

©Copyright 2026

Ethan Davis

Benefits of Being Bayesian: Motor Imagery Electroencephalogram Classification

Ethan Davis

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science in Computer Science & Software Engineering

University of Washington

2026

Committee:

Erika Parsons, Ph.D.

Elizabeth Field, Ph.D.

Michael Stiber, Ph.D.

Program Authorized to Offer Degree:
Computing and Software Systems

University of Washington

Abstract

Benefits of Being Bayesian: Motor Imagery Electroencephalogram Classification

Ethan Davis

Chair of the Supervisory Committee:
Erika Parsons, Ph.D.
Computing and Software Systems

Brain-computer interfaces (BCI) based on electroencephalogram (EEG) pattern recognition have demonstrated clinical value in motor rehabilitation and neuroprosthetics, yet accurate classification of EEG signals remains challenging due to poor signal-to-noise ratio and intrinsic nonstationarity. Bayesian machine learning offers a principled framework for uncertainty quantification that may address these limitations, though prior comparisons with frequentist methods have been limited to informal evaluations on single datasets. This thesis presents a large-scale meta-analysis of Bayesian versus frequentist machine learning for motor imagery EEG classification across 20 benchmark datasets. Six top-performing frequentist pipelines spanning spatial filtering, Riemannian geometry, and deep learning approaches were benchmarked against pairwise Bayesian variants. Pooled effects were estimated using a three-level meta-analytic model accounting for heterogeneity both within and between datasets. Results yielded statistically significant improvements in calibration metrics — expected calibration error and maximum calibration error — for Bayesian models, while discrimination metrics including AUROC and MCC showed no significant difference. Computational cost analysis further suggests that Bayesian methods are economically viable for future BCI research. These findings motivate a hypothesis that the primary advantage of Bayesian learning lies in long-term adaptability through sequential model updating and transferability across subjects, with probabilistic resolution and reliability proposed as key indicators for sustaining

high-performing BCI systems.

TABLE OF CONTENTS

	Page
List of Figures	v
List of Tables	ix
Chapter 1: Introduction	1
1.1 Interests	1
1.2 Motivation	2
1.3 Contribution	3
1.4 Bayesian Machine Learning	5
1.5 Motor Imagery Electroencephalogram	8
Chapter 2: Methods	11
2.1 Datasets	11
2.1.1 Background	11
2.2 Paradigm	20
2.2.1 Signal Preprocessing	20
2.2.2 Scoring Rules	28
2.3 Pipelines	31
2.3.1 Frequentist Baselines	36
2.3.2 Bayesian Models	45
2.3.3 Device Acceleration	62
2.3.4 Subprocess Isolation	64
2.4 Evaluation	66
2.4.1 Cross-Validation	68
2.4.2 Runtime Environment	69
2.5 Analysis	72
2.5.1 Data Wrangling	74

2.5.2	Moderator Analysis	74
2.5.3	Robust Testing	75
2.5.4	Metrics Reporting	76
Chapter 3:	Results	77
3.1	Posterior Sampling	77
3.1.1	Diagnostics Dispersion: Pipelines	78
3.1.2	Diagnostics Dispersion: Datasets	83
3.2	Raw Scores	89
3.2.1	Discrimination Metrics	91
3.2.2	Proper Scoring Rules	94
3.2.3	Calibration Metrics	95
3.3	Meta-Analysis	101
3.3.1	Intercept-Only	101
3.3.2	Meta-Regression	108
3.4	Computational Costs	119
3.4.1	Emissions Summary	119
3.4.2	Detailed Profiling	126
Chapter 4:	Discussion	136
4.1	Validity of Methods	136
4.1.1	Datasets	136
4.1.2	Paradigm	137
4.1.3	Pipelines	139
4.1.4	Evaluation	141
4.1.5	Analysis	141
4.2	Interpretation of Results	143
4.2.1	Posterior Sampling	143
4.2.2	Raw Scores	144
4.2.3	Meta-Analysis	144
4.2.4	Computational Costs	146
4.3	Limitations	147
4.4	Future Work	149

Chapter 5: Conclusion	151
Bibliography	153
Appendix A: Methods: Datasets	164
A.1 Electroencephalogram	164
A.1.1 Basic Principles	164
A.1.2 Experiment Protocol	167
A.1.3 Data Analysis	169
A.1.4 Feature Engineering & Selection	172
Appendix B: Methods: Paradigm	176
B.1 Metrics	176
B.1.1 Proper Scoring Rules	176
B.1.2 Discrimination	177
B.1.3 Calibration	179
Appendix C: Methods: Pipelines	181
C.1 Markov Chain Monte Carlo	181
C.1.1 Theoretical Foundations	181
C.1.2 Hamiltonian Monte Carlo	184
C.1.3 Runtime Diagnostics	188
C.1.4 Non-centered Parameterization	193
C.2 Common Spatial Pattern	196
C.2.1 Correctness Proof	196
C.3 Tangent Space	198
C.3.1 Definitions	199
C.3.2 Example	204
C.4 Linear Discriminant Analysis	206
C.4.1 Decision Boundary Derivation	206
C.5 Logistic Regression	208
C.5.1 Expressiveness Delineation	208
C.6 Support Vector Machine	212
C.6.1 Separating Hyperplane	212

C.6.2	Probability Calibration	216
C.6.3	Further Reading	217
C.7	Gaussian Process	219
C.7.1	Preliminaries	220
C.7.2	Classification	226
C.7.3	Covariance Functions	229
C.7.4	Sparse Approximation	229
C.8	Deep Neural Network	233
C.8.1	Convolutional Neural Network	237
C.8.2	Bayesian Neural Network	241
Appendix D:	Methods: Evaluation	244
D.1	Cross-Validation	244
D.1.1	Training Phase	244
D.1.2	Statistical Testing	246
Appendix E:	Methods: Analysis	250
E.1	Meta-Analysis	250
E.1.1	Motivation	250
E.1.2	Effect Sizes	252
E.1.3	Pooling Effect Sizes	256
E.1.4	Between-Study Heterogeneity	260
E.1.5	Forest Plots	264
E.1.6	Subgroup Analysis	264
E.1.7	Meta-Regression	268
E.1.8	Multilevel Meta-Analysis	272
Appendix F:	Results	277
F.1	Posterior Sampling	277
F.2	Raw Scores	295
F.3	Meta-Analysis	325
F.4	Computational Costs	335

LIST OF FIGURES

Figure Number	Page
1.1 PML Evolution	1
1.2 EEG general scheme	3
1.3 SPANER lab	5
1.4 Bias-variance tradeoff	7
1.5 Flat and sharp minima	8
1.6 Sensorimotor cortex	9
2.1 Channel montages	27
2.2 Datasets sample characteristics	30
2.3 Covariance estimation	38
2.4 ShallowConvNet backbone architecture	43
2.5 DeepConvNet backbone architecture	44
2.6 Bayesian logistic regression PGM	48
2.7 Bayesian LDA PGM	51
2.8 Beta distributions	52
2.9 Gaussian process with RBF kernel PGM	54
2.10 Gaussian process with linear kernel PGM	56
2.11 LogNormal distributions	58
2.12 Bayesian neural network PGM	60
2.13 Variational inference	63
2.14 MOABB evaluations	67
3.1 Pipelines bulk ESS	79
3.2 Pipelines tail ESS	80
3.3 Pipelines mean MCSE	81
3.4 Pipelines standard deviation MCSE	82
3.5 Datasets bulk ESS	83
3.6 Datasets tail ESS	86

3.7	Datasets mean MCSE	86
3.8	Datasets standard deviation MCSE	89
3.9	AUROC histogram	93
3.10	MCC histogram	94
3.11	NLL histogram	96
3.12	Brier score histogram	97
3.13	ECE histogram	99
3.14	MCE histogram	100
3.15	MCE intercept-only meta-analysis forest plot	102
3.16	MCE meta-regression forest plot	111
3.17	Average training duration	120
3.18	Average training emissions	123
3.19	Average training energy consumed	123
3.20	Average training CPU power	126
3.21	Average training GPU power	129
3.22	Average training CPU energy	131
3.23	Average training GPU energy	131
3.24	Average training RAM energy	134
3.25	Percentage of energy consumed	135
5.1	Prediction with uncertainty	151
A.1	Neural activity	165
A.2	P300 waves	166
A.3	Frequency bands	167
A.4	Experiment protocol	168
A.5	EEG signal artifacts	171
A.6	EEG data analysis	174
B.1	Confusion matrix	177
B.2	ROC curve plots	178
B.3	Calibration plot	179
C.1	MCMC circular leapfrog	187
C.2	MCMC funnel leapfrog	188
C.3	Trace plots	191

C.4	Autocorrelation plots	192
C.5	Rank plots	192
C.6	Neal’s funnel	194
C.7	Centered parameterization	194
C.8	Non-centered parameterization	195
C.9	Sample covariance matrices	196
C.10	Sample covariance matrices summed	197
C.11	Sample covariance matrices transformed	198
C.12	Tangent space, manifold, and geodesic	199
C.13	Exponential map and logarithmic map	202
C.14	Data preprocessing pipeline	205
C.15	Data preprocessing and classification pipeline	205
C.16	LDA decision boundary	208
C.17	Logistic regression decision boundary	212
C.18	Linear SVM	213
C.19	SVM affine subspace	214
C.20	SVM margin	216
C.21	SVM objective	217
C.22	SVM kernels	219
C.23	Gaussian process	220
C.24	Covariance functions	230
C.25	Additional covariance functions	230
C.26	Sparse Gaussian process	233
C.27	Multilayer perceptron	234
C.28	Voronoi diagram	236
C.29	1D discrete convolution	238
C.30	2D discrete convolution	239
C.31	Convolutions with padding	239
C.32	Convolutions with padding and strides	240
C.33	Max pooling	241
D.1	5-fold cross-validation	245
D.2	Validation set cross-validation	246

D.3	External validation	248
E.1	Hedges' g and Cohen's d	256
E.2	Random-effects model	259
E.3	Third Wave forest plot	265
E.4	Fixed-effects model	266
E.5	Meta-regression	269
E.6	Random-effects model and meta-regression	270
E.7	Multilevel meta-analysis	273
F.1	BNCI2014_001 BDCNN convergence plots	287
F.2	BNCI2014_001 BSCNN convergence plots	288
F.3	BNCI2014_001 CSPBLDA convergence plots I	289
F.4	BNCI2014_001 CSPBLDA convergence plots II	290
F.5	BNCI2014_001 CSPGP convergence plots I	291
F.6	BNCI2014_001 CSPGP convergence plots II	292
F.7	BNCI2014_001 TSBLR convergence plots	293
F.8	BNCI2014_001 TSGP convergence plots	294
F.9	AUROC intercept-only meta-analysis forest plot	325
F.10	MCC intercept-only meta-analysis forest plot	326
F.11	NLL intercept-only meta-analysis forest plot	327
F.12	Brier score intercept-only meta-analysis forest plot	328
F.13	ECE intercept-only meta-analysis forest plot	329
F.14	AUROC meta-regression forest plot	330
F.15	MCC meta-regression forest plot	331
F.16	NLL meta-regression forest plot	332
F.17	Brier score meta-regression forest plot	333
F.18	ECE meta-regression forest plot	334

LIST OF TABLES

Table Number	Page
2.1 Sample characteristics	13
2.2 Participant demographics	16
2.3 Equipment specifications	18
2.4 33 channels	21
2.5 Channel montages I	23
2.6 Channel montages II	24
2.7 Channel montages III	25
2.8 Channel montages IV	26
2.9 Paradigm preprocessing	29
2.10 MOABB motor imagery classification	33
2.11 MOABB motor imagery classification reduction	35
2.12 k -fold cross-validation	70
2.13 Sample splits	71
3.1 Pipelines bulk ESS summary statistics	78
3.2 Pipelines tail ESS summary statistics	79
3.3 Pipelines mean MCSE summary statistics	80
3.4 Pipelines std. MCSE summary statistics	81
3.5 Datasets bulk ESS summary statistics	84
3.6 Datasets tail ESS summary statistics	85
3.7 Datasets mean MCSE summary statistics	87
3.8 Datasets std. MCSE summary statistics	88
3.9 Raw scores summary statistics	91
3.10 AUROC histogram distribution	92
3.11 MCC histogram distribution	93
3.12 Scores worse than random classification	95
3.13 NLL histogram distribution	96

3.14	Brier score histogram distribution	97
3.15	ECE histogram distribution	99
3.16	MCE histogram distribution	100
3.17	Intercept-only pooled effect size	103
3.18	Intercept-only variance components	104
3.19	Intercept-only pooled effect size with RVE I	106
3.20	Intercept-only pooled effect size with RVE II	107
3.21	Average number of training samples	109
3.22	Meta-regression pooled effect size	110
3.23	Meta-regression variance components	112
3.24	Meta-regression pooled effect size with RVE I	113
3.25	Meta-regression pooled effect size with RVE II	114
3.26	Meta-regression pooled effect size with RVE III	115
3.27	Meta-regression slope estimate with CWB I	117
3.28	Meta-regression slope estimate with CWB II	118
3.29	Average training duration	121
3.30	Average training carbon emissions	122
3.31	Average training energy consumed	124
3.32	USA regional grid intensity	125
3.33	Average training CPU power	127
3.34	Average training GPU power	128
3.35	Average training CPU energy	130
3.36	Average training GPU energy	132
3.37	Average training RAM energy	133
A.1	EEG characteristics	165
C.1	Synthetic posteriors ESS	189
C.2	Synthetic posteriors $\hat{R}$	190
C.3	Synthetic posteriors MCSE	190
F.1	Posterior sampling I	277
F.2	Posterior sampling II	278
F.3	Posterior sampling III	279
F.4	Posterior sampling IV	280

F.5	Posterior sampling V	281
F.6	Posterior sampling VI	282
F.7	Posterior sampling VII	283
F.8	Posterior sampling VIII	284
F.9	Posterior sampling IX	285
F.10	Posterior sampling X	286
F.11	Average raw metrics BNCI2014_001	295
F.12	Average raw metrics BNCI2014_004	296
F.13	Average raw metrics Brandl2020	297
F.14	Average raw metrics Chang2025	298
F.15	Average raw metrics Cho2017	299
F.16	Average raw metrics Dreyer2023	300
F.17	Average raw metrics Forenzo2023	301
F.18	Average raw metrics GrosseWentrup2009	302
F.19	Average raw metrics GuttmanFlury2025_MI	303
F.20	Average raw metrics HefmiIch2025	304
F.21	Average raw metrics Kumar2024	305
F.22	Average raw metrics Lee2019_MI	306
F.23	Average raw metrics Liu2024	307
F.24	Average raw metrics PhysionetMI	308
F.25	Average raw metrics Schirrmeister2017	309
F.26	Average raw metrics Shin2017A	310
F.27	Average raw metrics Stieger2021	311
F.28	Average raw metrics Weibo2014	312
F.29	Average raw metrics Yang2025	313
F.30	Average raw metrics Zhou2020	314
F.31	Metrics differences I	315
F.32	Metrics differences II	316
F.33	Metrics differences III	317
F.34	Metrics differences IV	318
F.35	Metrics differences V	319
F.36	Metrics differences VI	320
F.37	Metrics differences VII	321

F.38 Metrics differences VIII	322
F.39 Metrics differences IX	323
F.40 Metrics differences X	324
F.41 Average carbon emissions BNCI2014_001	335
F.42 Average carbon emissions BNCI2014_004	336
F.43 Average carbon emissions Brandl2020	337
F.44 Average carbon emissions Chang2025	338
F.45 Average carbon emissions Cho2017	339
F.46 Average carbon emissions Dreyer2023	340
F.47 Average carbon emissions Forenzo2023	341
F.48 Average carbon emissions GrosseWentrup2009	342
F.49 Average carbon emissions GuttmannFlury2025_MI	343
F.50 Average carbon emissions HefmiIch2025	344
F.51 Average carbon emissions Kumar2024	345
F.52 Average carbon emissions Lee2019_MI	346
F.53 Average carbon emissions Liu2024	347
F.54 Average carbon emissions PhysionetMI	348
F.55 Average carbon emissions Schirrmeister2017	349
F.56 Average carbon emissions Shin2017A	350
F.57 Average carbon emissions V	351
F.58 Average carbon emissions Weibo2014	352
F.59 Average carbon emissions Yang2025	353
F.60 Average carbon emissions Zhou2020	354
F.61 Average power and energy usage BNCI2014_001	355
F.62 Average power and energy usage BNCI2014_004	356
F.63 Average power and energy usage Brandl2020	357
F.64 Average power and energy usage Chang2025	358
F.65 Average power and energy usage Cho2017	359
F.66 Average power and energy usage Dreyer2023	360
F.67 Average power and energy usage Forenzo2023	361
F.68 Average power and energy usage GrosseWentrup2009	362
F.69 Average power and energy usage GuttmannFlury2025_MI	363
F.70 Average power and energy usage HefmiIch2025	364

F.71 Average power and energy usage Kumar2024	365
F.72 Average power and energy usage Lee2019_MI	366
F.73 Average power and energy usage Liu2024	367
F.74 Average power and energy usage PhysionetMI	368
F.75 Average power and energy usage Schirrmeister2017	369
F.76 Average power and energy usage Shin2017A	370
F.77 Average power and energy usage Stieger2021	371
F.78 Average power and energy usage Weibo2014	372
F.79 Average power and energy usage Yang2025	373
F.80 Average power and energy usage Zhou2020	374

Chapter 1

INTRODUCTION

1.1 *Interests*

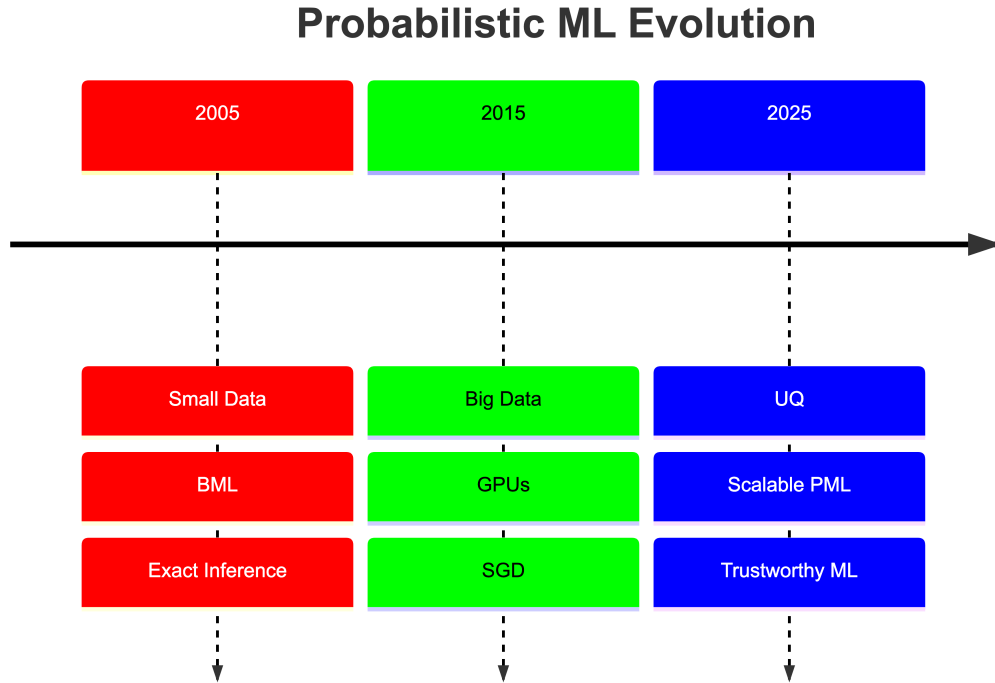


Figure 1.1: Historical events of probabilistic machine learning.

Circa 2005, Bayesian ML was popular. Practitioners worked in small data regimes where fitting models via maximum likelihood approaches systematically underestimates the variance of training samples. This results in bias related to overfitting. A counter to bias inherent with frequentist methods in small data regimes is to use Bayesian ML. Indeed, this approach offers principled uncertainty quantification (UQ) over the weight space of ML models [9].

By around 2015, big data was available to ML researchers and engineers. Training deep

learning models with GPUs was popular. Exact inference with probabilistic machine learning was intractable and interest turned to maximum likelihood point estimates solved by gradient descent. With large training sets, best effort approximations were performing better than ever. ML research was heavily focused on deep neural networks [34].

Today, about ten years later, LLM hallucinations are seen daily. This is one example of a scenario that is renewing interest in the principled frameworks of probabilistic ML [61]. New fields such as explainable AI (XAI) are growing with the interest of responsible and trustworthy AI. Uncertainty quantification offered by Bayesian ML lets practitioners answer difficult questions related to why a model returns certain predictions [58].

1.2 Motivation

Brain-computer interfacing (BCI) is active research for motor rehabilitation and neuroprosthetics [60]. Electroencephalogram (EEG) pattern recognition is a type of BCI that has been successfully applied in clinical settings. Examples include characterizing the onset and propagation of seizures in patients with epilepsy and eliciting biomarkers to provide a diagnosis as soon as possible of Alzheimer’s disease [24].

Advanced uses of BCIs consist of acquiring, analyzing, and translating brain signals into commands in real time for control or communication. BCI devices can be a valuable tool in the treatment of neurological disorders such as stroke and to provide assistive solutions for patients with spinal cord injury or the amyotrophic lateral sclerosis. Communication devices such as the P300 Speller allow users to communicate by selecting letters to form words and sentences [24].

Still, informative feature extraction and accurate classification of EEG patterns are considerably challenging due to the poor signal to noise ratio caused by volume conduction effects, interferences from various noises, and intrinsic signal nonstationarity. Various extensions of Bayesian learning offer an opportunity to explore more complex but potentially important properties of EEG towards improved performance of BCI systems. In the BCI literature, informal comparisons of frequentist and Bayesian ML evaluated on single datasets

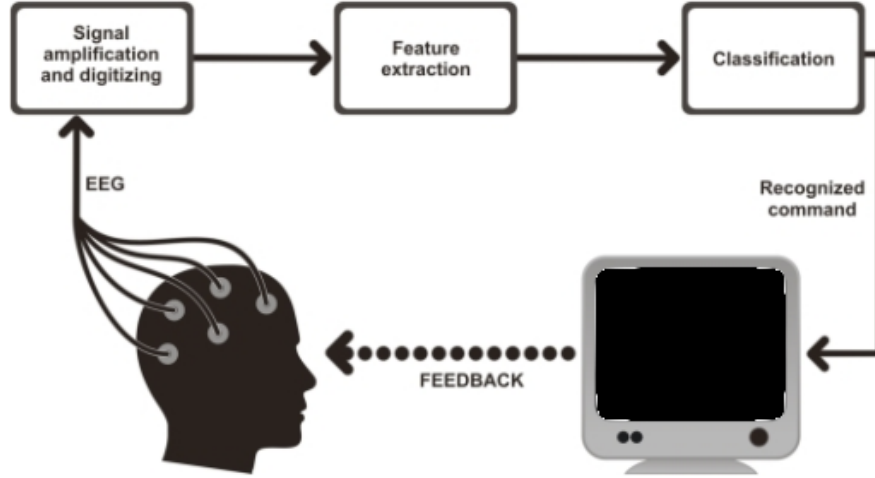


Figure 1.2: A general scheme of an EEG-based BCI [41].

have suggested greater discriminative feature extraction and robust classifier calibration by Bayesian learning to improve BCI performance [84].

1.3 Contribution

Our research analyzed the pooled effects of Bayesian ML across 20 different motor imagery EEG (MI-EEG) datasets using systematic hypothesis testing via meta-analysis instruments. Extending recent research published by Chevallier et al [21], we took 6 top performing frequentist ML models and benchmarked against pairwise Bayesian model variants evaluating performance in terms of ability to distinguish different outcomes and statistical consistency. The experiment design of our large scale study produced high quality Bayesian models generating statistically significant evidence for improved BCI performance.

Based on the BCI literature [81, 84, 47], our hypothesis was that cumulative results from frequentist and Bayesian classifiers trained on diverse EEG datasets would provide evidence that Bayesian methods offer improved prediction performance. Often in BCI research, reporting prediction performance is limited to AUROC or accuracy discrimination. However, in addition to standard discrimination, we were interested in measuring model reliability and

proper scoring (see section B.1). That is, we wanted to measure how well predicted probabilities matched true frequencies (scientific accuracy) and how concentrated the predictions were (scientific precision).

To test our hypothesis, we selected multiple measurements from discrimination, probability calibration, and proper scoring rules. We used proven frequentist ML models from the BCI literature as our baseline. As our experimental treatment, we reconfigured these baselines into Bayesian variants of themselves. Comparisons were measured by paired arithmetic differences in prediction scores. Although tests assessing performance for a given prediction metric were conducted through formal meta-analyses, conclusions drawn across tests per metric were narrative. Our null hypothesis was as follows.

H_0 : The precision-weighted pooled effect of paired differences in Bayesian minus frequentist ML predictions from left/right hand MI-EEG classification is not statistically significant for any discrimination, calibration, or proper scoring metric.

The broad scope of the empirical results from our evaluation showcased the benefits of Bayesian learning in MI-EEG classification and the advantage in BCI research. Furthermore, measurements from the computational costs of our experiment showed that future research into Bayesian learning for BCI is economical. Our research concluded with a newly generated hypothesis that the true benefits of Bayesian ML come from the long-term adaptability of Bayesian predictors in terms of both long sequences of model updates and wide transferability. Based on our study, we reasoned that quantitative metrics for probabilistic resolution and reliability are strong indicators for gauging and maintaining high performing BCI.

Our research was conducted as part of the Smart Platform for Augmented NEuro Rehabilitation (SPANER) lab of the University of Washington [26]. The SPANER lab involves the use of artificial intelligence, edge computing, and gamification to develop an environment that assists the rehabilitation of neurology patients while helping medical professionals collect and study data related to these activities. The focus of our research was on data analytics

and machine learning. Our work did not overlap with data acquisition, edge computing, or rehabilitation gamification for the SPANER lab.

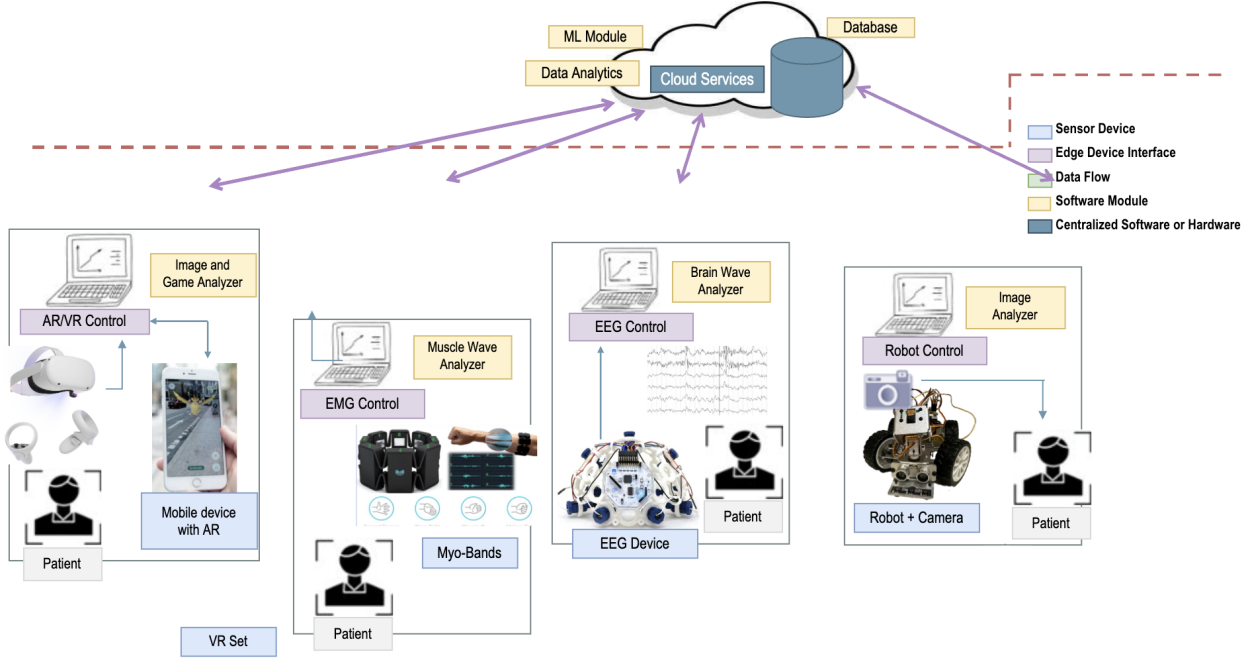


Figure 1.3: High level summary of the SPANeR Project.

1.4 Bayesian Machine Learning

The most common form of machine learning (ML) is supervised learning. In this problem, a task T is to learn a mapping f from inputs $\mathbf{x} \in \mathcal{X}$ to outputs $y \in \mathcal{Y}$. The inputs $\mathbf{x}$ are often a fixed dimensional vector $\mathbf{x} \in \mathbb{R}^D$, where D is the dimensionality of the vector. An experience E is given in the form of a set of N input-output pairs $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$, known as the training set, where N is the sample size. In classification problems, the output space is a set of C unordered and mutually exclusive labels known as classes $\mathcal{Y} = \{1, 2, \dots, C\}$. If there are just two classes, often denoted by $y \in \{0, 1\}$ or $y \in \{-1, 1\}$, it is called binary classification [57].

The natural starting point for classification is the joint probability $p(y, \mathbf{x})$, where y denotes the class label. Using Bayes' theorem, this joint probability can be decomposed as either $p(y)p(\mathbf{x}|y)$ or as $p(\mathbf{x})p(y|\mathbf{x})$. This gives rise to two different approaches to classification problems. The first, called the generative approach, models the class-conditional distribution $p(\mathbf{x}|y)$ for $y = \mathcal{C}_1, \dots, \mathcal{C}_C$ and also the prior probabilities of each class. The alternative approach, called the discriminative approach, focuses on modeling $p(y|\mathbf{x})$ directly [68].

The discriminative approach is attractive because it directly models what we want, namely $p(y|\mathbf{x})$. In addition, estimating the density for the class-conditional distributions is a difficult problem, particularly when $\mathbf{x}$ is high dimensional. So, if we are only interested in classification, then the generative approach may mean that we are trying to solve a harder problem than we need to. However, to deal with missing input values, outliers, and unlabeled data points in a principled way, it is helpful to have access to $p(\mathbf{x})$. This can be obtained by marginalizing the class label y from the joint distribution $p(y, \mathbf{x})$ [68].

The problem of training is to find a set of parameters that solves the maximum likelihood estimation (MLE) of the model [57].

$$p(y|\mathbf{x}, \hat{\theta}) \quad \text{such that} \quad \hat{\theta} = \arg \max_{\theta} \sum_{n=1}^N \log p(y_n|\mathbf{x}_n, \theta). \quad (1.1)$$

In the ML literature, it is also common for training to be expressed as a minimization of loss. However, minimizing the empirical risk of misclassification in the training set for some loss function $\ell(y_n, f(\mathbf{x}_n, \theta))$, is not our true goal. Instead, we want to minimize the expected loss in future data that we have not yet seen. That is, we want to generalize, rather than just do well on the training set. A model that perfectly fits the training data, but which is too complex, is said to suffer from overfitting. Similarly, classifiers that are too simple suffer from underfitting. This problem is known as the bias-variance tradeoff [57].

Indeed, overfitting is the result of using frequentist MLE-based methods to optimize point estimates of model parameters. Alternatively, a fully Bayesian treatment of machine learning

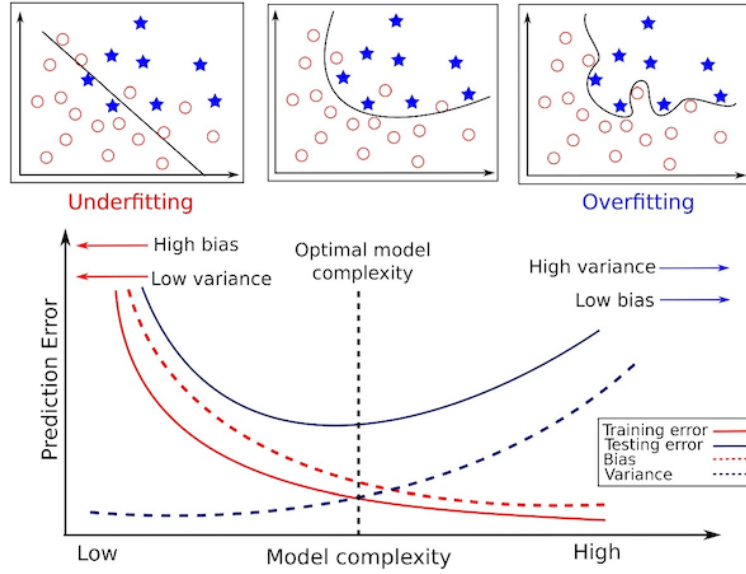


Figure 1.4: Example of the bias-variance tradeoff balancing underfitting and overfitting [76].

involves averaging over all possible models, with the contribution of each model weighted by its posterior probability. That is, the Bayesian perspective of ML allows us to represent uncertainty in our prediction, which can be important for various applications. For a training set $\mathcal{D}$, the target variable y , a new input value $\mathbf{x}$, and model weights θ , we have the following Bayesian treatment of machine learning [10].

$$p(y|\mathbf{x}, \mathcal{D}) = \int p(y|\mathbf{x}, \theta)p(\theta|\mathcal{D})d\theta. \quad (1.2)$$

However, there is a major drawback with the Bayesian framework which involves the integration over the space of parameters. Especially in modern deep learning, models can have millions or billions of parameters, and even simple approximations to such integrals are typically infeasible. In fact, given a limited compute budget and an ample source of training data, it will often be better to apply maximum likelihood techniques, generally augmented with one or more forms of regularization, to a large neural network than apply a Bayesian treatment to a much smaller model [10].

Striking a balance of frequentist and Bayesian techniques offers the greatest benefit. Bayesian can improve predictive accuracy and generalization performance. For example, certain optimization methods are capable of finding narrow but deep holes in the loss landscape known as sharp minima. Unfortunately, such minima correspond to overfitting the data and do not generalize as well as flat minima, which generally occupy a much greater volume of space. The Bayesian model averaging automatically weights regions with the highest volume, which corresponds to a better generalization [58].

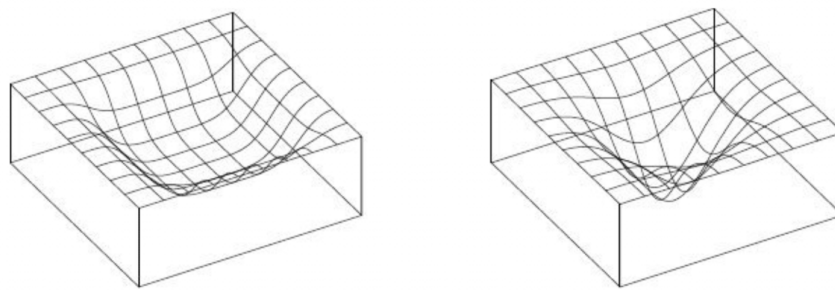


Figure 1.5: Examples of flat and sharp minima [58].

1.5 *Motor Imagery Electroencephalogram*

MI-EEG classification is a type of BCI research that aims to model imagined physical movement, such as with the left hand versus the right hand. It is well-established in BCI research and general neuroscience that brain signals related to physical movement and sensory perception originate from the sensorimotor cortex seen in Figure 1.6. Indeed, Figure 1.6 also shows the cortical homoculus which maps designated locations and amounts of brain tissue to certain parts of the body. A detailed background covering fundamental concepts of EEG signals, data acquisition protocols, and current analysis and modeling techniques is provided in section A.1.

For many years, early BCI research was focused on specialized knowledge in signal processing for feature engineering in the time and frequency domains [60]. Some of the first

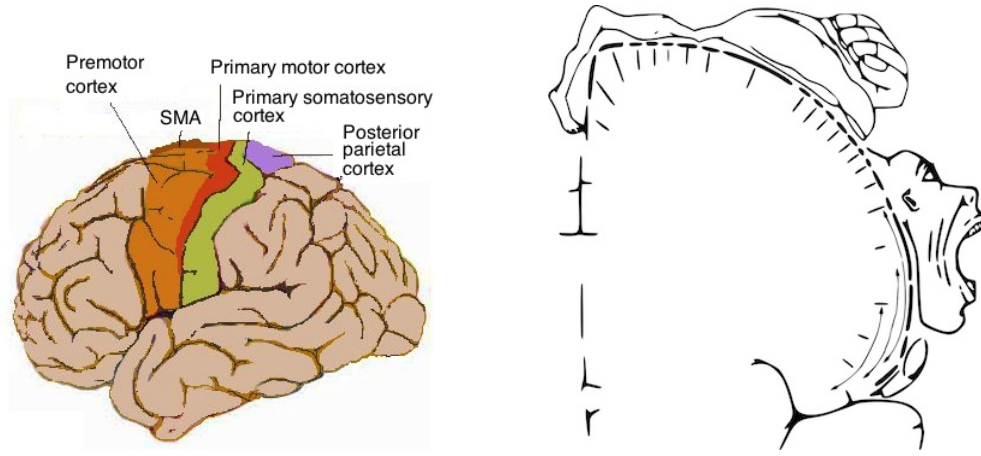


Figure 1.6: The motor and sensory cortex [43, 67].

and most widely used feature engineering and selection for EEG models included power and energy features from frequency bands and Hjorth parameters [65, 42]. These are derived features extracted from each channel, and therefore the total number of predictors scales proportional to the number of selected features and channels. When using time and frequency domain features, performing principal component analysis (PCA) is recommended before fitting ML models to avoid the curse of dimensionality.

The first EEG was recorded in 1929. However, BCI research did not take off until the late 1960s. Early EEG classification relied heavily on signal processing techniques from time and frequency domain feature extraction. Furthermore, the best performing classifiers were highly complex. A major shift to ML classification came in 2000 with spatial filters like the common spatial pattern (CSP). Not only did this feature engineering technique dramatically improve the performance of the EEG classification for BCI research, but it did so successfully with simple classifiers [60]. See section C.2 for a mathematically motivated explanation of CSP.

In the early 2010s, another major wave came to BCI research and EEG classification through Riemannian geometry. These achievements were built on theoretical advances made by researchers working on information geometry and its applications to signal processing [22].

Riemannian metrics correctly measure norms, distances, and other statistics for structures such as symmetric positive definite covariance matrices (SPD) unlike Euclidean metrics, which are approximations and are used by CSP [64]. For an in depth overview of Riemannian geometric statistics relevant to state of the art BCI research, see section C.3.

The success of deep learning (DL) is widely known from image, text, and speech recognition and generation. One of its defining characteristics that separates it from other ML algorithms is its automated feature extraction. Many complex and nonlinear layers of feature engineering and selection occur before a simple linear classification [10]. For EEG this means that the raw signals with potentially minimal to light preprocessing are used as input to the DL model.

However, DL performance for BCI research is modest at best, often outperformed by much cheaper and simpler spatial filter models using CSP or Riemannian metrics [21]. An explanation for this is that DL algorithms require massive datasets for training which is often unavailable for EEG. Regardless of its moderate success in EEG classification so far, DL is still highly active research. DL research for EEG includes temporal modeling, such as convolutional neural networks (CNNs), and spatial modeling, such as graph neural networks (GNNs) [24].

Chapter 2

METHODS

2.1 *Datasets*

We used the Mother of all BCI Benchmarks (MOABB) Python package PyPI version 1.5 to access all of the left hand and right hand motor imagery EEG datasets used in our experiment. For the purpose of unbiased dataset selection, we used all 20 available datasets, under the condition that, first, a dataset had at least 8 subjects to ensure a sufficiently sized training set, second, was not a subset of another dataset and, thirdly, was indeed an MI-EEG dataset as opposed to, for example, motor execution (ME). There were 10 datasets that did not meet these criteria. When all 20 datasets were downloaded, they were equivalent to 1 TB of storage on disk.

There is consistent terminology used in the BCI research concerning EEG signals that we establish with the following definitions. EEG experiments are designed with a composition of sessions, runs, and trials. A session is a series of runs in which EEG electrodes remain attached to the head of a subject. The term run refers to the period during which an experiment is conducted without interruption. Within a session, several runs are completed with potential intervals between them. A trial denotes a segment within a run where an atomic event occurs, such as left or right hand motor imagery [21].

2.1.1 *Background*

Certain of the datasets we used contain additional class labels in addition to left hand and right hand MI, such as feet and tongue MI, and resting state. Our paradigm included MI in the left hand and right hand, and excluded superfluous classes (see section 2.2 for more details). In the following, we briefly summarize each dataset and Table 2.1 – Table 2.3

summarize the key properties of the datasets. All descriptions assume the characteristics of the raw, unfiltered datasets. Certain datasets were collected for the purpose of creating a high-quality dataset. Others were acquired such that some types of noise were baked in or so that specific experimental interventions could be tested.

BNCI2014_001

This is one of the most widely used imagery datasets in BCI research, commonly referred to as “BCI Competition IV Dataset 2a”. It serves as a standard reference for 4-class motor imagery classification algorithms. The dataset is particularly useful for the following: multi-class motor imagery classification (left hand, right hand, both feet, and tongue), transfer learning studies (9 subjects, 2 sessions each), and cross-session variability analysis [78].

BNCI2014_004

This dataset is commonly referred to as “BCI Competition IV Dataset 2b”. It is widely used for binary left versus right hand motor imagery classification tasks. Two sessions were recorded on different days for each subject. Each session is comprised of 6 runs separated by short breaks. One run consists of 20 trials (10 for each of the two possible classes), which yields a total of 120 trials per session [78].

Brandl2020

Each subject completed 1 session with 7 runs. Run 0 was for calibration, with 72 trials, no feedback, and no distraction. Runs 0–6 included feedback, having 72 trials each with auditory feedback, under 6 distraction conditions (clean, eyes closed, news, numbers, flicker, and vibrotactile stimulation). Auditory cues indicated left and right hand imagery [13].

Table 2.1: Summary of the experiments for the datasets used in our evaluation. The total trials counts have not been filtered to left and right hand motor imagery. That is, all class labels from the original datasets are included.

Sample Characteristics					
Dataset	No. subjects	No. sessions / subject	No. runs / session	Trial length (s)	Total trials
BNCI2014_001	9	2	6	4.0	62208
BNCI2014_004	9	5	1	4.5	32400
Brandl2020	16	1	7	4.5	8064
Chang2025	28	6	1	4.0	13440
Cho2017	52	1	1	3.0	9800
Dreyer2023	87	1	6	5.0	20880
Forenzo2023	25	2	3	4.0	1875
GrosseWentrup2009	10	1	1	7.0	3000
GuttmannFlury2025_MI	31	3	1	4.0	2520
HefmiIch2025	37	6	1	10.0	3330
Kumar2024	18	6	4	5.0	7156
Lee2019_MI	54	2	1	4.0	11000
Liu2024	50	1	1	4.0	2000
PhysionetMI	109	1	3	3.0	69760
Schirrmeister2017	14	1	2	4.0	13440
Shin2017A	29	3	1	10.0	5220
Stieger2021	62	6	1	3.0	250000
Weibo2014	10	1	1	4.0	5600
Yang2025	51	3	1	4.0	39600
Zhou2020	8	7	9	5.0	33600

Chang2025

This dataset is from the multi-paradigm EEG dataset for studying upper limb rehabilitation. It contains EEG data from 28 healthy subjects. Six rehabilitation paradigms were tests. Namely, motor execution (grasp and release), motor imagery (imagine grasp and release), virtual reality assisted motor imagery, mirror therapy (left/right only), soft rehabilitation gloves, and mirror gloves (left / right only). Each paradigm has multiple sessions [20].

Cho2017

Subjects were asked to imagine the movement of the hand depending on the instructions given. Five or six runs were performed during the motor imagery experiment. After each run, the classification accuracy was calculated over one run and the subject was given feedback to increase motivation. Between each run, a maximum 4-minute break was given depending on the demands of the subject [23].

Dreyer2023

The experiment was designed to investigate the impact of the participant's and experimenter's gender on MI BCI performance. Demographic and biosocial information includes: gender, birth year, laterality, vision, vision assistance, familiarity with cognitive science or neurology, level of education, physical activity, meditation, and attentional, neurological and psychiatric symptoms. A personality and cognitive profile were evaluated using the fifth edition of the 16 Personality Factors (16PF5) test, the mental rotation test, and the learning style index. Pre and post experiment questionnaires were conducted for mood, mindfulness, and motivational states [66, 8].

Forenzo2023

This dataset is from the article on integrating simultaneous motor imagery and spatial attention for EEG-BCI control. It contains EEG data from 25 subjects recorded in 5 sessions on

different days. Multiple task conditions were tested: motor imagery alone (left/right hand), overt spatial attention alone, and combined MI and spatial attention [32].

GrosseWentrup2009

The Munich motor imagery dataset was conducted so that a trial started with the central display of a white fixation cross. After 3 seconds, a white arrow was superimposed on the fixation cross, either pointing to the left or right. Subjects were instructed to perform haptic motor imagery of the left or right hand during display of the arrow, as indicated by the direction of the arrow. Although subjects were explicitly instructed to perform haptic motor imagery with the specified hand (to imagine feeling instead of visualizing how their hands moved), the exact choice of the type of imagery movement was not specified [35].

GuttmannFlury2025_MI

This dataset is from the article on combining EEG, eye-tracking, and high-speed video for ocular activity analysis across BCI paradigms. Four paradigms were tested: MI, ME, SVEP, and P300. The trial structure was the following: 2 seconds of fixation, 4 seconds of imagery/execution, and then 1-1.5 seconds of rest [37].

HefmiIch2025

This EEG data are from 37 subjects: 17 healthy controls and 20 patients with intracerebral hemorrhage (ICH). The motor imagery task involved grasping the left and right hands. The trial structure was as follows: 2 seconds of cue, 10 seconds of MI and 15 seconds of rest. Sessions vary from 1 to 5 per subject [72].

Kumar2024

Session 1 was an offline calibration session with 4 bar-feedback runs. Sessions 2–6 were online sessions consisting of bar-feedback runs with continuous visual feedback followed by car rac-

Table 2.2: In the BCI literature, it is common to see several of these demographics reported. However, it is not standard practice to list each of these attributes. In cases where a characteristic was not reported, we use the placeholder “NR”.

Participant Demographics						
Dataset	No. subjects	Health status	Age	No. right handedness	No. naive experience	No. females
BNCI2014_001	9	healthy	NR	NR	NR	NR
BNCI2014_004	9	healthy	24.7 ± 3.3	9	9	NR
Brandl2020	16	healthy	26.3	NR	13	6
Chang2025	28	healthy	20–33	28	NR	14
Cho2017	52	healthy	24.7 ± 3.86	50	NR	19
Dreyer2023	87	healthy	29.0 (19–59)	87	87	41
Forenzo2023	25	healthy	25.5	24	19	10
GrosseWentrup2009	10	healthy	25.6 ± 2.5	8	NR	2
GuttmannFlury2025_MI	31	healthy	28.3 (20–57)	NR	NR	11
HefmiIch2025	37	ICH pt.	20–65	37	NR	8
Kumar2024	18	healthy	23.22 ± 2.59	NR	18	7
Lee2019_MI	54	healthy	24–35	50	NR	25
Liu2024	50	stroke pt.	56.7 ± 10.57	NR	NR	11
PhysionetMI	109	healthy	NR	NR	NR	NR
Schirrmeister2017	14	healthy	27.2 ± 3.6	12	NR	6
Shin2017A	29	healthy	28.5 ± 3.7	29	29	15
Stieger2021	62	healthy	18–63	NR	NR	49
Weibo2014	10	healthy	24.0 (23–25)	10	10	7
Yang2025	51	healthy	17–30	51	51	18
Zhou2020	20	healthy	23.2 ± 1.47	20	NR	9

ing games. In each bar-feedback run, subjects performed 20 trials in pseudo-random order. Two transfer learning training protocols were used on separate subjects: generic recentering with unsupervised domain adaption and personally assisted recentering for supervised recalibration [48].

Lee2019_MI

The first 3 seconds of each trial began with a black fixation cross that appeared in the center of the monitor to prepare subjects for the MI task. Afterwards, the subject performed the imagery task of grasping with the appropriate hand for 4 seconds when the right or left arrow appeared as a visual cue. After each task, the screen remained blank for 6 seconds (± 1.5 seconds) [50].

Liu2024

This dataset included data from 50 acute stroke patients admitted to the stroke unit of Xuanwu Hospital of Capital Medical University. The time after the stroke ranges from 1 day to 30 days. In the instruction stage, patients were asked to imagine grasping a spherical object with the left or right hand. In the MI stage, participants imagined performing this action, a video of the gripping motion was played on the computer, which led the patient to imagine grabbing the ball [52, 53].

PhysionetMI

This dataset consists of EEG recordings obtained from 109 volunteers. Each subject performed 14 experimental runs: two one-minute baseline runs (one with eyes open, one with eyes closed), and three two-minute runs of each of the four following tasks: open and close the left or right fist, imagine opening and closing the left or right fist, open and close both fists or both feet, and imagine opening and closing both fists or both feet [69, 33].

Table 2.3: Notch filtering at 50/60 Hz was also applied to all datasets, which is common practice to remove power line noise.

Equipment Specifications					
Dataset	No. channels	Frequency (Hz)	Reference	Sensors	Amplifier
BNCI2014_001	22	250	left mastoid	Ag/AgCl	BrainAmp
BNCI2014_004	3	250	left mastoid	Ag/AgCl	g.tec
Brandl2020	63	1000	nose	Ag/AgCl	BrainAmp
Chang2025	59	1000	NR	Ag/AgCl	Neuracle
Cho2017	64	512	CMS/DRL	active	Biosemi
Dreyer2023	27	512	left earlobe	active	g.tec
Forenzo2023	64	1000	between Cz–CPz	Ag/AgCl	Neuroscan
GrosseWentrup2009	128	500	Cz	NR	BrainAmp
GuttmannFlury2025_MI	64	1000	right mastoid	Ag/AgCl	Neuroscan
HefmiIch2025	32	256	NR	NR	g.tec
Kumar2024	22	512	CPz	NR	ANT Neuro
Lee2019_MI	62	1000	naison	Ag/AgCl	BrainAmp
Liu2024	29	500	CPz	Ag/AgCl	ZhenTec
PhysionetMI	64	160	mastoid	NR	NR
Schirrmeister2017	128	500	NR	NR	NR
Shin2017A	30	200	linked mastoids	active	BrainAmp
Stieger2021	62	1000	NR	NR	Neuroscan
Weibo2014	60	200	nose	Ag/AgCl	Neuroscan
Yang2025	59	1000	NR	Ag/AgCl	Neuracle
Zhou2020	41	500	Cz	Ag/AgCl	Neuroscan

Schirrmeister2017

This is a high-gamma dataset consisting of four classes of movements: left hand, right hand, both feet, and rest. Data were acquired in an EEG lab optimized for non-invasive detection of high-frequency movement-related EEG components. During each trial, the subjects had to repetitively clench their toes, perform sequential finger-tapping of their left or right hand, or relax [70].

Shin2017A

The EEG electrodes were placed according to the international 10-5 system. Nine sensors were placed around the frontal channels (Fp1, Fp2, and Fpz), twelve sensors each around the motor channels (C3 and C4), and three sensors around the visual areas (Oz). Note that sensors for these named channels were not placed for this dataset [73].

Stieger2021

The main goals of this study were to characterize how individuals learn to control SMR-BCIs and to test whether this learning can be improved through behavioral interventions such as mindfulness training. Participants were initially assessed for baseline BCI proficiency and then randomly assigned to an 8 week mindfulness intervention, or waitlist control condition where participants waited for the same duration as the MBSR class before starting BCI training. However, the control group was offered a comparable MBSR course after completing all experimental requirements. Following the 8 weeks, the participants returned to the lab for 6 to 10 sessions of BCI training [77].

Weibo2014

This dataset was used to investigate the differences in EEG patterns between simple limb motor imagery and compound limb motor imagery. Seven types of mental tasks were designed that involved three tasks of simple motor imagery of the limb (left hand, right hand, feet),

three tasks of compound motor imagery of the limb combining hand with hand/foot (both hands, left hand combined with the right foot, right hand combined with the left foot), and rest state [83].

Yang2025

This dataset contains data recorded on 62 subjects across 3 sessions on different days. Two paradigms were used: the 2C paradigm (subjects 1-51) with left hand versus right hand motor imagery, and the 3C paradigm (subjects 52-62) having left hand, right hand and foot motor imagery. The trial time was as follows: 1.5 seconds cue, 4 seconds of MI, and 2 seconds of rest [82].

Zhou2020

This dataset was recorded on 20 subjects over 7 sessions (one session every approximately 2 days over 2 weeks) with no feedback training. Two groups of subjects were recorded: S-subjects (subjects 1-12) using 41 EEG and 4 EOG channels, and A-subjects (subjects 13-20) with 26 EEG channels and 2 EOG channels. Four MI classes were recorded: left hand, right hand, both feet, and rest [85].

2.2 Paradigm

2.2.1 Signal Preprocessing

Preprocessing could be summarized as filtering and then transformation. Since we were only concerned with left hand and right hand motor imagery, our paradigm filtered any other classes. Additional filtering to the raw datasets was the inclusion of only sessions 1–6 for Stieger2021 and the inclusion of only subjects 13–20 with Zhou2020. In the case of Stieger2021, this session reduction was done in response to GPU memory bottlenecks specifically in our Bayesian neural networks (see section 2.3 and section 2.4 for details). Regarding Zhou2020, only A-subjects (see section 2.1) whose electrode information could be

parsed were included.

We also filter the sensors used in the evaluation for all datasets. Figure 1.6 shows the sensorimotor cortex of the brain, which stretches roughly from ear to ear, up and over the scalp. On the right of Figure 1.6, we also see how certain regions of the sensorimotor cortex correspond to specific parts of the body. For example, it is well-established in the BCI literature that the international 10-20 montage channels (see section A.1 for details) C3 and C4 correspond to the left and right hands.

In our experiment, we filtered the sensors to those from the sensorimotor cortex of the international 10-05 montage. These channels are shown in Table 2.4. In other words, we took all 33 channels from the sensorimotor cortex as a base set, and for each dataset we used the intersection of this base set and the available channels in the data set in our evaluation. See Table 2.5 – Table 2.8 for this intersection with the 33 channel base set and the sensors of each dataset. Also, see Figure 2.1 for a visualization of these filtered montages.

Table 2.4: The 33 channels from the international 10-05 montage covering the sensorimotor cortex. This table orders the sensors according to how they would be seen in the international 10-05 montage. See Figure A.4 in section A.1 for an example of where these electrodes are placed on a subject’s head in the international 10-20 montage.

Channel Montage Base
FC5, FC3, FC1, FCz, FC2, FC4, FC6, FCC5h, FCC3h, FCC1h, FCC2h, FCC4h, FCC6h, C5, C3, C1, Cz, C2, C4, C6, CCP5h, CCP3h, CCP1h, CCP2h, CCP4h, CCP6h, CP5, CP3, CP1, CPz, CP2, CP4, CP6

The filter on the electrodes used in our experiment removes extraneous noise from the data we evaluated with machine learning. Regions outside the sensorimotor cortex include

the frontal cortex associated with decision making, planning, and problem-solving, and the occipital cortex located in the back of the brain and primarily responsible for processing visual information. It is common knowledge in the BCI literature that signals corresponding to eye movements and invariable heartbeats add noise to EEG signals.

However, the original motivation for filtering the electrodes came from our response to processing failures that occurred when we combined our largest datasets and pipelines. We were initially curious to include all available sensors in our experiment, treating all datasets and their acquisition protocols as heterogeneous sources of information. Our solution to reduce channels to a subset of the sensorimotor cortex still maintains this spirit. It was also the correct way to perform dimensionality reduction within the BCI domain.

We also explored dimensionality reduction through unsupervised methods such as principal component analysis (PCA). However, even with Bayesian logistic regression, which was one of our smallest Bayesian models (see section 2.3 for background on the example), the Riemannian geometry tangent space features were too much for the system to handle. As the dimensionality of the tangent space feature space scales quadratically with the number of input channels, this created a feature space with thousands of dimensions before filtering the channels to the sensorimotor cortex.

Methods such as PCA could not be effectively applied, as the highest variance explained by any one principal component was less than 1%. Otherwise, we would have needed to make an indefensible compromise to accept some upper bound on the number of principal components to represent the feature space, yet at the same time only account for a percentage much less than 90% of explained variance. By performing dimensionality reduction of the electrodes during paradigm preprocessing such that we only included sensorimotor channels, we not only followed frequent practice from the BCI literature but also avoided any use of PCA.

Generally, unsupervised spatial filtering, such as PCA or independent component analysis (ICA), is outperformed by supervised spatial filtering methods such as common spatial pattern (CSP) for EEG datasets. Although unsupervised techniques are still common in

some cases for cleaning EEG datasets, without careful attention, these methods are known to scrub beyond the point of useful signals and create corruptive noise. For these reasons, we wanted to avoid PCA and ICA in our signal preprocessing and machine learning pipelines. Indeed, we prevented introduction of these unsupervised techniques into our experiment, and note that supervised algorithms (CSP) were a core method in certain pipelines.

Table 2.5: The channel montages.

Channel Montages I	
Dataset	Channels
BNCI2014.001	FC3, FC1, FCz, FC2, FC4, C5, C3, C1, Cz, C2, C4, C6, CP3, CP1, CPz, CP2, CP4
BNCI2014.004	C3, Cz, C4
Brandl2020	FC5, FC3, FC1, FCz, FC2, FC4, FC6, C5, C3, C1, Cz, C2, C4, C6, CP5, CP3, CP1, CPz, CP2, CP4, CP6
Chang2025	FC5, FC3, FC1, FCz, FC2, FC4, FC6, C5, C3, C1, Cz, C2, C4, C6, CP5, CP3, CP1, CP2, CP4, CP6
Cho2017	FC5, FC3, FC1, FCz, FC2, FC4, FC6, C5, C3, C1, Cz, C2, C4, C6, CP5, CP3, CP1, CPz, CP2, CP4, CP6

Since we were concerned with the motor imagery EEG paradigm, we performed 8 Hz and 32 Hz frequency high pass and low pass temporal filtering, respectively, within electrode signals (see section A.1 for an explanation). In addition, we downsampled the sampling

Table 2.6: The channel montages.

Channel Montages II	
Dataset	Channels
Dreyer2023	FC5, FC3, FC1, FCz, FC2, FC4, FC6, C5, C3, C1, Cz, C2, C4, C6, CP5, CP3, CP1, CPz, CP2, CP4, CP6
Forenzo2023	FC5, FC3, FC1, FC2, FC4, FC6, C5, C3, C1, C2, C4, C6, CP5, CP3, CP1, CP2, CP4, CP6
GrosseWentrup2009	FC5, FC3, FC1, FC2, FC4, FC6, FCC5h, FCC3h, FFC1h, FCC2h, FCC4h, FCC6h, C5, C3, C1, Cz, C2, C4, C6, CCP5h, CCP3h, CCP1h, CCP2h, CCP4h, CCP6h, CP5, CP3, CP1, CPz, CP2, CP4, CP6
GuttmannFlury2025_MI	FC5, FC3, FC1, FCz, FC2, FC4, FC6, C5, C3, C1, Cz, C2, C4, C6, CP5, CP3, CP1, CPz, CP2, CP4, CP6
HefmiIch2025	FC5, FC1, FC2, FC6, C3, Cz, C4, CP5, CP1, CP2, CP6

Table 2.7: The channel montages.

Channel Montages III	
Dataset	Channels
Kumar2024	FC5, FC1, FC2, FC6, C3, Cz, C4, CP5, CP1, CP2, CP6
Lee2019_MI	FC5, FC3, FC1, FC2, FC4, FC6, C5, C3, C1, Cz, C2, C4, C6, CP5, CP3, CP1, CPz, CP2, CP4, CP6
Liu2024	FC3, FCz, FC4, C3, Cz, C4, CP3, CP4
PhysionetMI	FC5, FC3, FC1, FCz, FC2, FC4, FC6, C5, C3, C1, Cz, C2, C4, C6, CP5, CP3, CP1, CPz, CP2, CP4, CP6
Schirrmeister2017	FC5, FC3, FC1, FCz, FC2, FC4, FC6, FCC5h, FCC3h, FCC1h, FCC2h, FCC4h, FCC6h, C5, C3, C1, Cz, C2, C4, C6, CCP5h, CCP3h, CCP1h, CCP2h, CCP4h, CCP6h, CP5, CP3, CP1, CPz, CP2, CP4, CP6

Table 2.8: The channel montages.

Channel Montages IV	
Dataset	Channels
Shin2017A	FCC5h, FCC3h, FCC4h, FCC6h, Cz, CCP5h, CCP3h, CCP4h, CCP6h
Stieger2021	FC5, FC3, FC1, FCz, FC2, FC4, FC6, C5, C3, C1, Cz, C2, C4, C6, CP5, CP3, CP1, CPz, CP2, CP4, CP6
Weibo2014	FC5, FC3, FC1, FCz, FC2, FC4, FC6, C5, C3, C1, Cz, C2, C4, C6, CP5, CP3, CP1, CPz, CP2, CP4, CP6
Yang2025	FC5, FC3, FC1, FCz, FC2, FC4, FC6, C5, C3, C1, Cz, C2, C4, C6, CP5, CP3, CP1, CP2, CP4, CP6
Zhou2020	FC5, FC3, FC1, FCz, FC2, FC4, FC6, C5, C3, C1, Cz, C2, C4, C6, CP5, CP3, CP1, CPz, CP2, CP4, CP6

frequencies for all datasets to 128 Hz. This sampling frequency is seen in other BCI MI-EEG benchmarks where it could be applied to reduce memory consumption during evaluation [46]. See Table 2.3 in section 2.1 where the sampling rate during data acquisition is as high as 500–1000 Hz. It is common to see modern BCI hardware capable of collecting EEG signals at these frequencies.

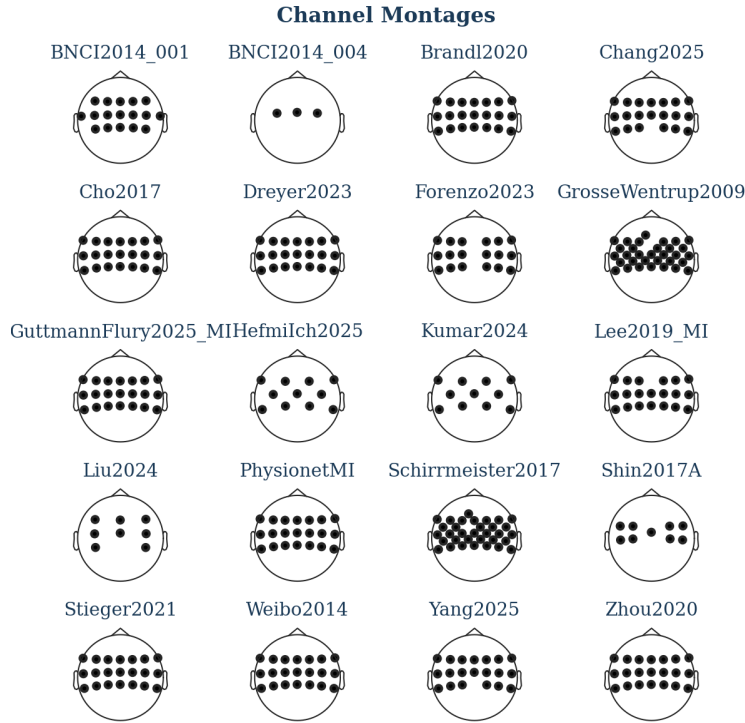


Figure 2.1: Visualization of the channel montages.

The dataset with the lowest sampling rate was PhysionetMI at 160 Hz. Rather than up-sampling some datasets and downsampling others, which would introduce inconsistency in data quality for our experiment, we uniformly downsampled all datasets to 128 Hz. We considered that it could be valuable to maintain this source of heterogeneity between datasets. However, we justified downsampling all datasets as it is common practice in the BCI literature. Indeed, downsampling reduced the number of time points per trial. This was another

way in which we reduced the dimensionality across datasets to prevent runtime failures during evaluation.

The results of signal preprocessing can be seen in Table 2.9. This shows the dimensions of each preprocessed dataset that was input to our machine learning pipelines. In addition, Figure 2.2 visualizes the relative scale of the preprocessed datasets. In all cases, the class balance between the labels in the left hand and in the right hand was at most within 1% of a 50:50 ratio. Indeed, this class balance ratio was expected from the design of the data acquisition experiments (see section 2.1).

In terms of how long preprocessing took, in most cases it was negligible. However, in particular with Stieger2021, preprocessing took almost 2 hours. Furthermore, signal preprocessing for Yang2025 took close to 24 hours. Therefore, with all datasets we preprocessed once and saved the results to disk. This reduced the load time to within seconds for all datasets. The preprocessed datasets also required less disk space. Whereas the raw datasets needed 1 TB of storage, their preprocessed results only required 20 GB of disk space.

2.2.2 *Scoring Rules*

We were interested in validating the predictive performance of our machine learning models both in terms of discrimination and calibration. Since our task was binary left hand and right hand motor imagery classification, our measurements stemmed from the four category confusion matrix. For our work, false positives and false negatives were equally problematic. Furthermore, we were dealing with balanced class splits. Given our goals and these circumstances, we used the area under the receiver operating characteristic (AUROC), Matthews correlation coefficient (MCC), negative log-likelihood (NLL), Brier score (BS), expected calibration error (ECE), and maximum calibration error (MCE) as scoring rules.

The metrics AUROC and MCC, NLL and BS, and ECE and MCE divide into distinct classes of prediction measurement (see section B.1 for a detailed description). In summary, AUROC and MCC strictly measure discrimination independently of calibration as summaries of the confusion matrix. NLL and BS are proper scoring rules that capture both discrimi-

Table 2.9: The three dimensional tensor of each dataset after paradigm preprocessing.

Paradigm Dimensions			
Dataset	No. trials	No. channels	No. timepoints
BNCI2014_001	2592	17	513
BNCI2014_004	6520	3	377
Brandl2020	8058	21	576
Chang2025	2260	20	512
Cho2017	10520	21	384
Dreyer2023	20792	21	640
Forenzo2023	3661	18	512
GrosseWentrup2009	3000	32	896
GuttmannFlury2025_MI	2520	21	512
HefmiIch2025	2940	11	1280
Kumar2024	7156	11	640
Lee2019_MI	10800	20	512
Liu2024	2000	8	512
PhysionetMI	4918	21	385
Schirrmeister2017	6742	33	512
Shin2017A	1740	9	1281
Stieger2021	43422	21	384
Weibo2014	1580	21	513
Yang2025	30591	20	512
Zhou2020	6666	21	640

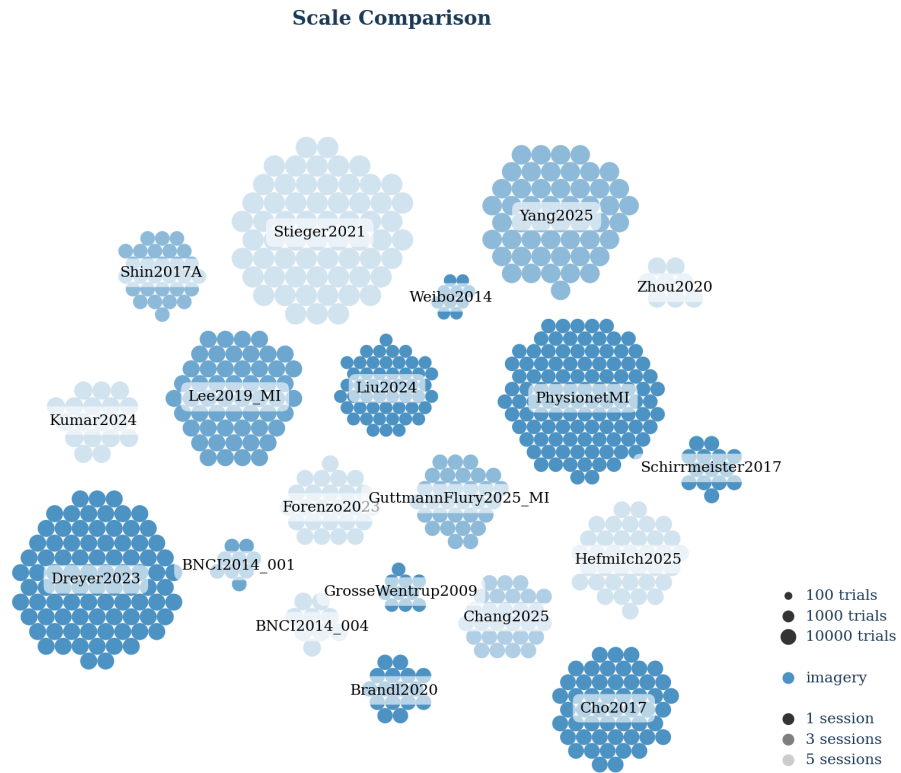


Figure 2.2: Visualization of the sample characteristics for the datasets we used.

nation and calibration. Training a classifier to optimize a proper scoring rule automatically results in a well-calibrated classifier [58]. Furthermore, ECE and MCE offer separate perspectives on summaries of the probability calibration curve.

Our choice of metrics provides us with a range of prediction performance scores, from purely discriminative to purely calibration based. These paired scores offer similar perspectives for each of the classes of classification metrics such that taking additional measurements would offer diminishing returns. For all pipeline evaluations, each prediction measurement was taken during the testing phase of k -fold cross-validation. In section 2.5 we describe how we combined these scores to aggregate the comparison results of frequentist and Bayesian models from all datasets or studies.

To collect multiple model evaluation measurements when using MOABB, we forked the open-source GitHub repository and implemented a feature request to allow configurable scoring rules. Our branch was merged into development and public facing documentation was updated with instructions for how to program evaluations accordingly. This feature was released in MOABB PyPI version 1.5 [5]. A workable but higher risk and more costly post hoc alternative solution could have been to manage model pickling. However, this would also have required reloading datasets into memory and recreating cross-validation splits.

2.3 Pipelines

Our work can be seen as an extension of previous research by the mother of all BCI benchmarks (MOABB), characterized as the largest EEG-based benchmark. In its study, MOABB identified three categories for the most commonly researched BCI classifiers today: those based on raw signals, algorithms relying on covariance matrices seen as elements of a Riemannian manifold, and deep learning (DL) approaches. The MOABB benchmark included 19 unique classifiers associated with the motor imagery paradigm, 7 for raw signals, and 6 each using Riemannian and deep learning techniques [21].

All pipelines included in the MOABB evaluation are known to perform well in predictive classification in their respective category throughout the BCI literature [21]. Moreover,

the models that MOABB benchmarked are frequentist, being based on optimized point estimates of their parameters. We selected the top performing estimators of the MOABB benchmark from each model family for our own evaluation involving the comparison of frequentist baselines versus their Bayesian analogs in terms of predictive performance.

The MOABB literature consisted of 10 MI-EEG datasets that were evaluated using a binary left versus right hand paradigm: BNCI2014_001, BNCI2014_004, Cho2017, GrosseWentrup2009, Lee2019_MI, PhysionetMI, Schirrmeister2017, Shin2017A, Weibo2014 and Zhou2016 [21]. Among all datasets and pipelines, we rank pipelines within their categories according to the average AUROC reported across the datasets. For our experiment, we used the two best performing raw signal and deep learning classifiers: CSP+SVM and CSP+LDA, and ShallowConvNet and DeepConvNet, respectively.

The two best performing Riemannian pipelines reported by MOABB were TS+EL (AUROC: 82.129%) and ACM+TS+SVM (AUROC: 81.574%). The next two ranked models were then TS+LR (AUROC: 81.479%) and TS+SVM (AUROC: 81.352%). Both TS+EL and TS+LR are similar to each other in that elastic net (EL) is logistic regression (LR) with the addition of L1 (lasso) and L2 (ridge) regularization penalties in the loss function [57]. Similarly, ACM+TS+SVM is similar to TS+SVM, the difference being its use of an augmented covariance matrix (ACM) based on the Yule-Walker equations of an autoregressive model [19].

Rather than select the top two averaging Riemannian geometry pipelines for our experiment, we chose the third and fourth ranked models. The difference in AUROC between TS+EL and TS+LR was 0.65%, and for ACM+TS+SVM and TS+SVM it was 0.222%. From a narrative perspective of these raw differences, a less than 1% improvement in EEG classification is hardly notable in both cases. However, our final decision to take TS+LR and TS+SVM for our own evaluation was motivated by estimator characteristics and consistency across other categories of classifiers.

The design of a fully Bayesian LR model is straightforward, requiring prior probability distributions being placed on the parameters. However, EL belongs to a penalized maximum

Table 2.10: Summary of MOABB results for the classification of motor imagery of the left versus right hand [21]. The AUROC scores were averaged across all ten datasets and sorted in descending order. The yellow, red, and blue color rows correspond to Riemannian geometry, deep learning, and raw signal based classifiers, respectively.

Classifier Performance	
Pipeline	AUROC percentage (avg.)
TS+EL	82.129
ACM+TS+SVM	81.574
TS+LR	81.479
TS+SVM	81.352
FgMDM	80.405
ShallowConvNet	78.029
CSP+SVM	77.72
CSP+LDA	77.629
TRCSP+LDA	77.256
LDCSPauto+shLDA	77.202
FilterBank+SVM	76.243
DeepConvNet	75.402
LogVariance+SVM	74.165
LogVariance+LDA	73.056
EEGNet_8_2	72.117
MDM	68.604
EEGTCNet	64.781
EEGITNet	63.724
EEGNeX	58.654

likelihood estimation (MLE) framework, lacking a clean mapping to a Bayesian formulation. This property decided our use of TS+LR over TS+EL. Furthermore, we considered the closeness of feature extraction methods for CSP+SVM and CSP+LDA, and also ShallowConvNet and DeepConvNet. By choosing TS+SVM over ACM+TS+SVM, we again analyzed models (based on Riemannian geometry) with similar feature engineering and selection.

Our reason for taking only two models from each category for our experiment was partly practical. Every frequentist or Bayesian pipeline that we wanted to evaluate would also require its analog pair to be developed for our experiment. Then, for every additional model pair, we would have needed to evaluate another 40 (both frequentist and Bayesian) pairwise classifiers, with 20 being the number of datasets we analyzed.

For Riemannian pipelines specifically, we also considered that minimum distance to the mean (MDM) based classifiers do not map to probabilistic scoring since their estimations are geometric. Furthermore, as for raw signal based pipelines, we saw that all predictors were either SVM or LDA, with varying feature extractors. Given that our interests were Bayesian prediction as opposed to feature engineering, this supported our selection of just CSP+SVM and CSP+LDA.

Beyond ShallowConvNet and DeepConvNet, the remaining deep learning pipelines were all different versions of the same EEGNet. Today, deep learning methods are generally outperformed in terms of both predictive performance and computational cost by spatial filtering models in BCI research. Therefore, we felt that it would not be particularly interesting to include EEGNet variants in our evaluation and instead focused on trends between categories of model families.

Therefore, we selected a uniform two number of pipelines from each category in our own evaluation, as shown in Table 2.11. Note that although our experiment was defined by cross subject evaluation, the MOABB study was conducted under within session evaluation [21]. See section 2.4 for a description of the difference between these two types of evaluations. It is possible that the ranking of classifiers would have different if MOABB had completed the analysis with cross subject evaluation. However, to define our experiment a priori, we

Table 2.11: Reduction of Table 2.10, again with rows in descending order of average AUROC percentage, with only those pipelines used in our experiment.

Filtered Classifier List	
Pipeline	AUROC percentage (avg.)
TS+LR	81.479
TS+SVM	81.352
ShallowConvNet	78.029
CSP+SVM	77.72
CSP+LDA	77.629
DeepConvNet	75.402

accepted the scores reported by MOABB as the best evidence that we had to select our pipelines.

Generally, we did not perform grid search or any other hyperparameter space optimization algorithm. The reason for this was practical. We were already fitting 12 pipelines by 20 datasets ($12 \times 20 = 240$ models) with cross-validation, and we did not want to multiply this computation further. Without hyperparameter search, our evaluation measuring the predictions performance scores took 10 days to complete. In the following, we explain our rationale behind the hyperparameters that we fixed for all evaluations.

For the purpose of reproducibility of our results, we fixed random seeding across all of our pipelines, setting the initial random state to 1. There are many avenues by which randomness is introduced into scikit-learn, PyTorch, and PyMC models. For example, with SVMs, random seeding is used by Platt scaling, in deep learning, validation splits are drawn randomly, and with Bayesian models, MCMC is defined by random sampling. We also set

environment variables to control randomness in the way libraries allocate GPU resources. Continue reading to see how these examples apply to our pipelines.

2.3.1 *Frequentist Baselines*

Generally, the structure of our frequentist pipelines mirrored that of the MOABB study [21]. Since we used the MOABB software package for evaluation [46], all of our models were based on scikit-learn pipelines [63]. Both our raw signals and Riemannian geometry based classifiers used covariance estimation that was implemented by the PyRiemann software package [6]. Respectively, these classifiers also used PyRiemann for the computation of common spatial pattern (CSP) and tangent space (TS). In terms of runtime, PyRiemann is grounded to CPU based workloads.

We used the scikit-learn implementation of linear discriminant analysis (LDA), logistic regression, and support vector machine (SVM) for our frequentist raw signals and Riemannian predictors. As we describe in more detail later, these are the only pipelines in our experiment that were bound to CPU execution for evaluation because they relied on scikit-learn. Specifically for training SVMs we considered using the Rapids ecosystem CuML software package that wraps the scikit-learn interface over GPU functionality. However, the calibration cross-validation loop for our SVMs made the development cost with CuML more than we were willing to accept.

Our deep learning models were implemented with PyTorch, which is built so that it easily maps to GPU based workloads [62]. For the backbone architecture of these classifiers, we used BrainDecode [4]. This software package is highly active for BCI DL research. It is built with Skorch [79], another package that wraps PyTorch functionality with the scikit-learn standardized interface. This allows BrainDecode DL pipelines to seamlessly fit into MOABB evaluations.

Additionally, frequentist pipelines also used centering and scaling according to the z -score transformation, after feature extraction and before fitting classification. This was done to match the Bayesian pipelines where centering and scaling is required to simplify the prior

specification (see below for more details). If we were not comparing the frequentist and Bayesian prediction, we would not have applied the z -score transformation in the frequentist pipelines. This is a point of divergence between our frequentist pipelines and those used in the MOABB benchmark [21].

Although we generally applied the z -score transformation for our frequentist estimators, we did not do so for the frequentist deep learning pipelines. Indeed, DL feature extraction weights and classification weights are coupled, since they are fitted together. Inserting centering and scaling into the DL backbones between feature extraction and classification would break its parameter optimization algorithm. In subsection 2.3.2, we describe how we applied centering and scaling to our Bayesian deep learning pipelines.

Covariance Estimation

Both the CSP and TS based classifiers, respectively belonging to the raw signal and Riemannian geometry model families, correspond to manual feature engineering and selection. This is in contrast to ShallowConvNet and DeepConvNet, which are deep learning pipelines and therefore characterized by automated feature extraction. Since both the CSP and the TS classifiers depend on the covariance matrix of the EEG electrodes at the time points of each trial as input features, these models could be seen to belong to a subgroup distinct from the DL models in a moderator analysis.

For all covariance estimation, we used the oracle approximating shrinkage (OAS) estimator. The calculation of the covariance matrix of the raw sample $\hat{\Sigma} = (1/n-1)X^\top X$ is generally numerically unstable. Additionally, despite being an asymptotically unbiased estimator of the covariance matrix, the maximum likelihood estimator is not a good estimator of the eigenvalues of the covariance matrix, so the precision matrix obtained from its inversion is not accurate. To avoid such an inversion problem, a transformation of the empirical covariance matrix is introduced: the shrinkage [63].

Mathematically, shrinkage consists of reducing the ratio between the smallest and the largest eigenvalues of the empirical covariance matrix. It can be done by shifting every

eigenvalue according to a given offset, and this is equivalent to finding the L2-penalized MLE of the covariance matrix. In 2004, Ledoit and Wolf proposed a formula to compute the optimal shrinkage coefficient that minimizes the mean squared error between the estimated and the real covariance matrix [63].

Then, under the assumption that the data are Gaussian distributed, Chen et al derived a formula aimed at choosing a shrinkage coefficient that yields a smaller mean squared error than the one given by Ledoit-Wolf. Their result is known as the oracle approximating shrinkage (OAS). The Ledoit-Wolf and OAS closed-form solutions to covariance estimation can be sensitive to outliers, and therefore in these scenarios iterative estimators can be used for robust covariance estimation [63].

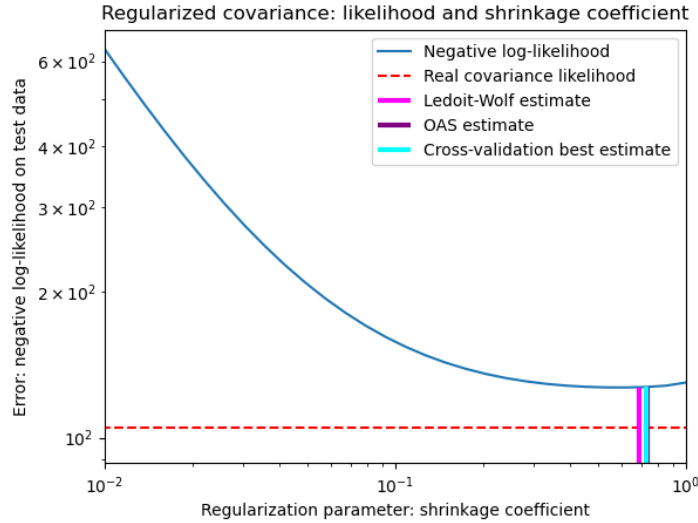


Figure 2.3: Comparison of the bias-variance tradeoff when setting shrinkage [63].

In all cases of our CSP and TS classifiers, we used OAS for covariance estimation. A robust covariance estimator is better at outlier detection, but is more computationally expensive. Between the experimental conditions under which our datasets were acquired (see section 2.1) and our preprocessing (see section 2.2), we felt that outlier detection was not necessary. Furthermore, in BCI scenarios for motor imagery in real-time, a closed-form co-

variance estimation is almost surely the better choice. Lastly, MOABB used OAS as its benchmark [21].

Spatial Filters

With CSP we used the Euclidean metric for the estimation of mean covariance matrices. The use of this metric was reasonable since CSP is a linear operation, which strictly kept the extraction of feature of the raw signals in vector space. Programmatically, the Euclidean metric was also the default setting. Additionally, we decomposed 6 components or orthogonal discriminative axes. A common number of components to decompose ranges between 4–8 in the BCI literature, including MOABB, which used 6 for its study [21].

Then, for TS feature engineering, we applied the Riemannian metric for reference matrix estimation and for the tangent space map. As TS feature selection operates with respect to the Riemannian manifold for symmetric positive definite (SPD) matrices, using the Riemannian metric was the consistent choice. For a detailed background behind the CSP and TS spatial filters that we used in our raw signals and Riemannian geometry pipelines, see section C.2 and section C.3.

Linear Discriminant Analysis

We used singular value decomposition (SVD) to fit a Gaussian density to both classes, assuming that they shared the same covariance matrix. This created a classifier with a linear decision boundary, generated by fitting class conditional densities to the data and using Bayes' rule. SVD is the recommended solution because it avoids computing the covariance matrix, which must be done with covariance estimation. An overview of the theory of LDA can be found in section C.4.

Logistic Regression

Our logistic regression configuration used the limited-memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) solver with default regularization. This is a second-order optimization algorithm, but it uses limited memory to approximate the Hessian matrix. The LBFGS solver works reasonably well for a wide class of problems and is the default used by scikit-learn [63]. To accommodate convergence with complex datasets, we increased the maximum number of iterations from the default limit of 100 to 1000. See section C.5 for further reading on logistic regression.

Support Vector Machine

Support vector machines are generally not interpreted with probability calibration, since their decision boundary is geometrically optimized. However, a common post-hoc method for measuring probabilistic metrics with SVMs is to use Platt scaling. This involves internal calibration cross-validation, producing scores that can be aggregated to find empirical probabilities. Platt scaling with 5-fold cross-validation is the default method used by scikit-learn for probabilistic calibration with SVMs. For a more detailed description of how SVMs are geometrically optimized and the use of Platt scaling for calibration, see section C.6.

The scikit-learn SVM classifier implementation has a fit time that scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. Since the largest datasets we used in our experiment trained with tens of thousands of trials per fold of cross-validation, we considered options besides the scikit-learn CPU bound computation. As we mentioned above, we were interested in using CuML from the Rapids ecosystem to move training phase computation to GPU devices.

Unfortunately, this calibration cross-validation introduced additional challenges related to our specific runtime environment. In subsection 2.3.4, we explain why we used a separate spawned subprocess to build each model, with the disk being used to communicate the estimated weights between processes. However, as a consequence of scikit-learn controlling

the inner cross-validation loop for calibration, we found that these fitted calibration weights were not in our direct access when writing to disk the trained weights of the SVM pipelines that we had programmed.

Note that the fitted calibration weights not being saved to disk was a problem specific to our use of CuML to train SVMs. That is, we did not experience this problem when using scikit-learn to fit SVM classifiers. The problem first arose when we upgraded our Python environment to MOABB PyPI version 1.5 from version 1.43, and we may have considered staying at version 1.43 so that we could continue to use CuML to train SVMs. However, this decision was a tradeoff, since version 1.5 offered additional datasets that we could use in our evaluation.

In the end, we felt it was more important to proceed with the evaluation than to debug our implementation using CuML. Therefore, we reverted our SVM (and logistic regression) classifiers back to scikit-learn from CuML. Although we note that this decision notably increased the time needed to train SVMs for our largest datasets with tens of thousands of samplers per fold of cross-validation. The change in implementation itself from CuML to scikit-learn was simple, as both packages have an identical interface, requiring only a correction of package imports in our software.

Regarding our SVM configurations, with common spatial pattern features, we trained SVM classifiers with a radial basis function (RBF) kernel, also known as the Gaussian kernel [63]. In the above, we described how we decomposed 6 components for our CSP feature engineering. This is a relatively low-dimensional feature space which makes nonlinear RBF computation inexpensive. The RBF kernel is also the default kernel for SVM classification in scikit-learn, due to a combination of its flexibility, smoothness, and comparative tractability over other nonlinear kernels such as the polynomial kernel.

Given tangent space features, we applied a linear kernel to train SVM estimators. The dimensionality of the tangent space for the Riemannian manifold of $n \times n$ SPD (covariance) matrices scales quadratically with the number of independent parameters $\sum_{i=1}^n i = n(n+1)/2$. This makes the use of an RBF kernel less attractive. However, there is also geometric

reasoning to prefer a linear kernel. That is, a decision boundary in the tangent space of the reference matrix at the geometric mean for all trials can be tuned so that it linearly approximates the best fit geodesic separating both classes.

Convolutional Neural Network

An in-depth overview of the mechanisms from the forward pass of deep neural networks (DNNs) and convolutional neural networks (CNNs) is found in section C.8. There, we describe the high-level components and connectors used to build the ShallowConvNet and DeepConvNet architectures that were designed specifically for the low latency and heterogeneous characteristics of EEG signals. Figure 2.4 visualizes the backbone of ShallowConvNet and Figure 2.5 shows the deeper architecture of DeepConvNet. These estimators are configured by the number of channels in the BCI device, time points per trial, and classification labels associated with the tensor of raw EEG signals given as input.

To train ShallowConvNet and DeepConvNet we use a uniform evaluation loop for both models. The loss function used was the cross entropy loss which combines log softmax and negative log likelihood loss. By combining these operations, it avoids the potential for exploding or vanishing gradients and precision loss that can occur when calculating softmax and log separately. The use of cross entropy loss is motivated for probabilistic ML since it minimizes the distance between the true and predicted distributions.

The gradient update algorithm we used was AdamW. Unlike standard Adam, this method decouples the learning rate and weight decay which improves model generalization. Whereas Adam treats weight decay as regularization, adding it to the gradient, AdamW applies weight decay directly to the weights. This prevents large gradients from disproportionately reducing the effect of regularization. AdamW is highly competitive at balancing the speed and final accuracy of modern deep learning architectures. We configured the learning rate and weight decay to be their default settings at $1e-3$ and $1e-2$, respectively.

We carefully configured our evaluation loop to ensure reliable convergence across all datasets. Our deep learning pipelines were set to repeat a maximum of 300 epochs with a

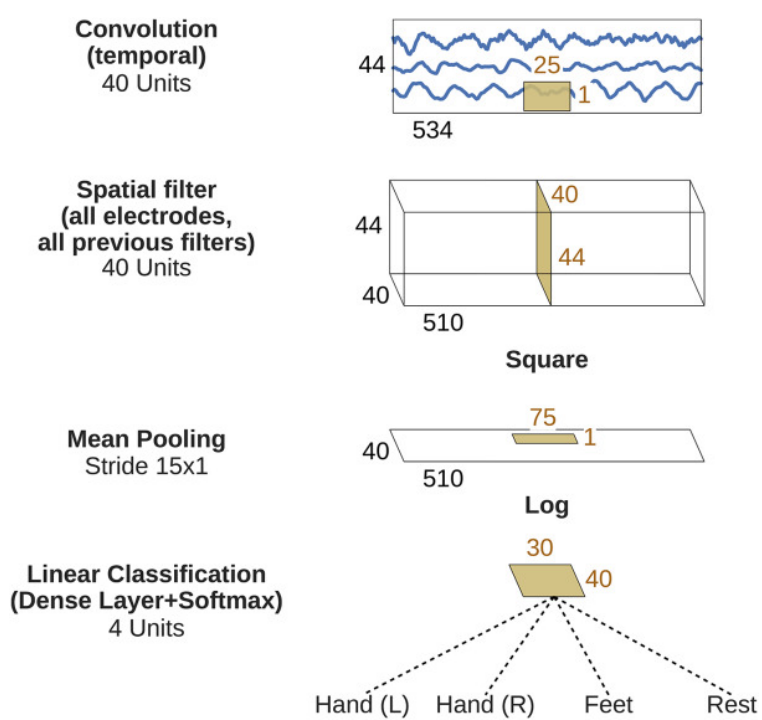


Figure 2.4: The ShallowConvNet backbone architecture [70].

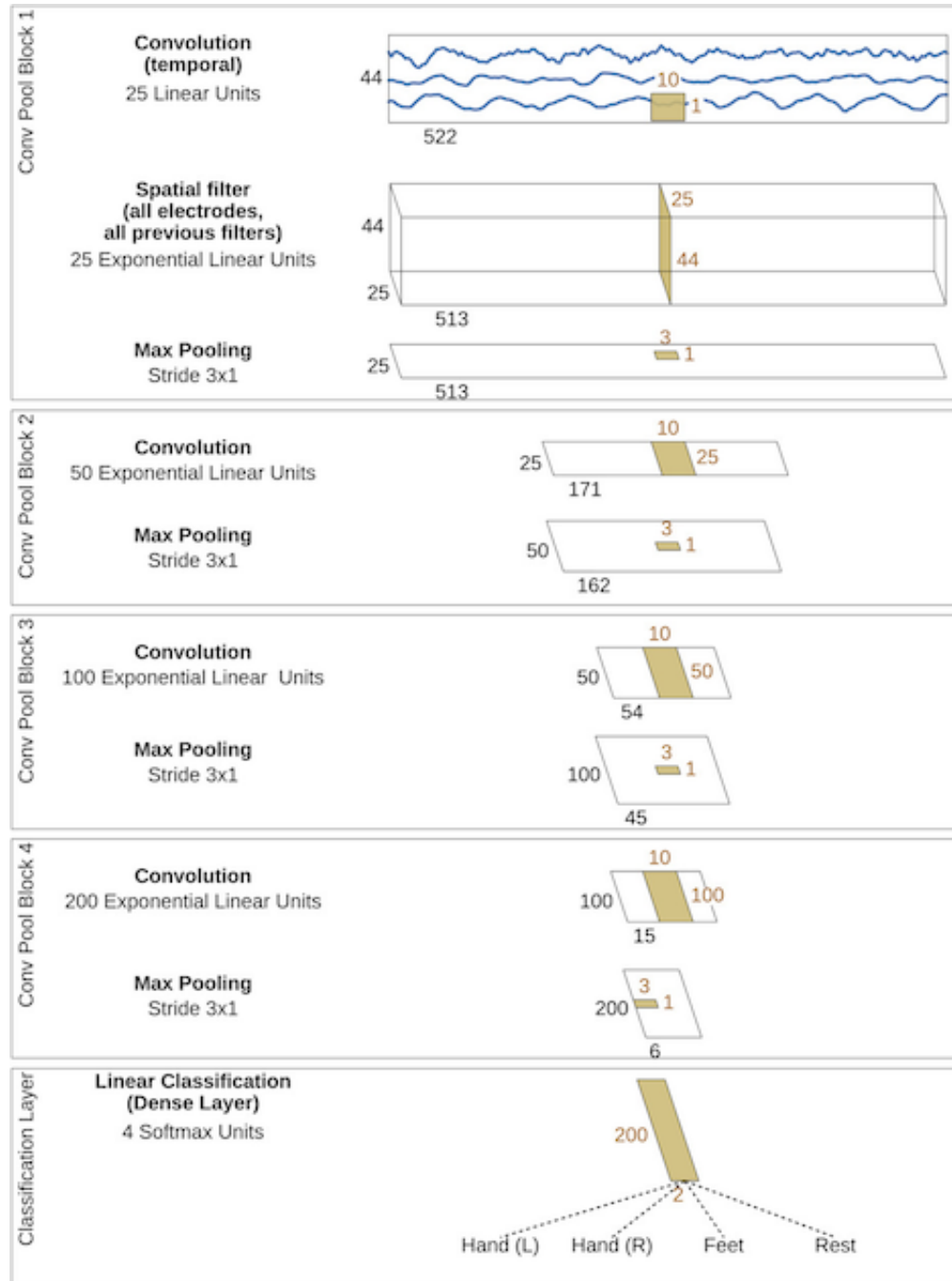


Figure 2.5: The DeepConvNet backbone architecture [70].

standard batch size of 64 and 80-20 validation split. Early stopping based on validation loss was used with 150 epochs of patience and a $1e-4$ threshold for improvement. A learning rate scheduler was also used, with 50 patience epochs before halving the learning rate, but not below $1e-6$, if the validation loss did not decrease.

2.3.2 Bayesian Models

Our Bayesian machine learning pipelines were built so that they were close analogs of the frequentist pipelines that we used. In all cases, the feature engineering and selection remained fixed. That is, covariance estimation, spatial filters, and also deep learning feature extraction was the same in our Bayesian classifiers as it was for our frequentist models.

In our experiment design, frequentist predictors were the control group and each was paired with a Bayesian predictor that was the intervention. Fundamentally, the focused comparison being made was frequentist point estimate optimization $p(y|x, \hat{\theta})$ versus Bayesian parameter space modeling of the posterior weights $p(\theta|\mathcal{D})$ with Bayesian model averaging for prediction $p(y|x, \mathcal{D}) = \int p(y|x, \theta)p(\theta|\mathcal{D})d\theta$. Indeed, all feature extraction was deterministic, whereas class prediction was experimented with as being probabilistic.

Exploring probabilistic prediction is most commonly seen as placing a probability distribution on classification weights. Standard practice assumes that many parameters are generated by a Gaussian based distribution prior to seeing the observed likelihood of class labels from the training set. These probability distributions of parameters and predictions create a probabilistic graphical model (PGM). A detailed overview of PGMs can be found in Bishop’s 2006 Pattern Recognition and Machine Learning [9].

We provide PGM diagrams for each of our Bayesian pipelines which aid in understanding their structure including non-centered parameterization (see section C.1) and bottlenecks. The intuition for PGMs is that they describe dependencies between parameters and their data generating distributions. In our diagrams, white ellipses wrap latent, indirectly observed random variables. White rectangles represent latent yet deterministic variables such as those used in non-centered parameterization.

Furthermore, gray ovals and ellipses denote directly observed variables. The ovals are the data we see from the training and testing sets. In general, ellipses surround random variables. In all cases, these components are labeled with their corresponding data generating mechanism, though, in the case of probability distributions, hyperparameters are not shown. Our PGM diagrams also use plate notation to summarize one-to-many and many-to-many relationships.

When plate notation is used, we select the dimensions (576, 22) from the BNCI2014_001 dataset that is widely seen as a gold standard for BCI research. This choice of dimensions is made without loss of generality. Specifically, 576 is the number of trials or data points from one subject, and 22 is the number of electrode channels, where these dimensions are taken from the raw, unfiltered 4-class BNCI2014_001 dataset with a motor imagery paradigm apply only 8–32 Hz high and low band pass filtering.

Generally, probabilistic learning is easier when the features are centered and scaled. Applying z -score normalization to features before training allows distributions with their hyperparameters to be fitted more accurately to otherwise heterogeneous scales of features. Therefore, as part of all our Bayesian pipelines, we applied z -score normalization to features before they were passed on to classifier prediction. Like we mentioned above in our discussion of the frequentist pipelines from our experiment, this centering and scaling was also applied to features before frequentist classification. However, in the case of deterministic point estimate prediction, this was done to maintain consistency of features across all pipelines.

For all of our Bayesian models, we used PyMC for implementation [1]. In addition to the widely public facing PyMC, we also used PyMC Extras, offering a wrapper to PyMC with the scikit-learn interface. This made it simple for us to integrate PyMC probabilistic programming with our scikit-learn-based ML pipelines and benefit from a standardized baseline implementation for many operations. Using this wrapper, we configured a uniform Markov chain Monte Carlo (MCMC) sampling rule set for all our Bayesian models.

We used the default number of warmup draws and tuning samples for MCMC with PyMC which is 1000 for both parameters. For robustness of posterior sampling, we increased the

target of accepted accuracy for posterior distribution samples from the default of 0.8 to 0.95. The cost of increasing the target accuracy is often smaller steps per iteration and slower convergence, and the benefit is more accurate posterior approximations. Given that we had access to GPU devices, we also overrode the default PyMC sampling algorithm.

In section 2.4 we describe all the resources that we had access to our server for the evaluation of our experiment. However, we note that the server had $4\times$ networked NVIDIA RTX A5000 GPUs. This hardware configuration influenced how we designed our MCMC sampler. For example, when using MCMC, it is recommended to use multiple chains which can be used to validate the correctness of convergence (see section C.1 for more details). We used one chain per GPU, which allowed us to compare the speed and predictive accuracy of the posterior distribution across all chains.

The fastest sampler available through PyMC is based on NumPyro which uses JAX. NumPyro is a low-level probabilistic programming package that requires software instructions to be written explicitly. In contrast, PyMC is a general purpose probabilistic programming package that allows developers to define models implicitly. A familiar motto of PyMC is that developers should say yes to automating inference and not to automated model building. JAX is a library for array-oriented numerical computation that provides a unified NumPy-like interface to workloads that run on a CPU, GPU, TPU, and in local or distributed settings.

Bayesian Logistic Regression

The Bayesian logistic regression model we built appears both in one of our tangent space features models, and in both of our Bayesian deep learning classifiers since these architectures use simple linear classification after multi-layer automated feature extraction. See below for more details on how we implemented Bayesian neural networks. We note that since the predictor is identical in both cases, it could be interesting to do a separate moderator analysis focusing on the difference in predictive performance and similarity in features between tangent space features and DL learned features.

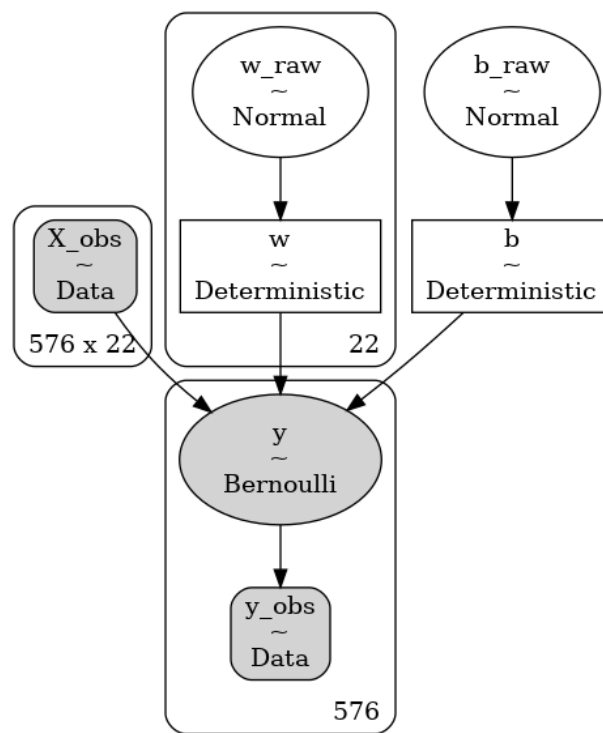


Figure 2.6: The Bayesian logistic regression PGM.

Figure 2.6 shows that we used non-centered parameterization with our Bayesian logistic regression model. That is, we decorrelated the weights and bias parameter distributions from the label distribution (see section C.1 for more information about non-centered parameterization). The motivation for this choice came from a desire to maintain consistency between Bayesian logistic regression and deep learning predictors. Our DL models had many more parameters leading into the posterior, and we note that we observed thousands of sampling divergences before applying non-centered parameterization (see below for details).

We used standard normal distributions for both the weights and the bias parameters. The standard normal distribution $\mathcal{N}(0, 1)$ was reasonable to assume since all features were centered and scaled prior to posterior sampling. In other words, it was justified to believe that the parameters would be zero-centered with a standard deviation of one. Though, specifically for our weights standard deviation, it was more reasonable that we assume the combination of all weights parameters had a standard deviation of one, and therefore that each parameter have standard deviation $1/\sqrt{D}$ where D was the number of trials in the training set.

That is, to handle the variance accumulation problem we set $\sigma_w = 1/\sqrt{D}$, similar to LeCun initialization [58]. Our goal was to keep the pre-activation (logit) in a reasonable range regardless of input dimensionality. The logit for a single sample is as follows.

$$z = \mathbf{w}^\top \mathbf{x} = \sum_{i=1}^D w_i x_i.$$

Assuming the inputs $\mathbf{x}$ are z -score normalized, and the weight are IID with $w_i \sim \mathcal{N}(0, \sigma_w^2)$, then we have the following by the law of total variance.

$$\begin{aligned} \text{var}(w_i x_i) &= \text{var}(w_i) \text{var}(x_i) \\ &= \text{var}(w_i) (\mathbb{E}[x_i^2] - \mathbb{E}[x_i]^2) \\ &= \text{var}(w_i) \mathbb{E}[x_i^2] \\ &= \sigma_w^2 \cdot 1. \end{aligned}$$

Then, summing the D independent terms gives the following.

$$\text{var}(z) = D \cdot \sigma_w^2.$$

We wanted $\text{var}(z) \approx 1$ since the sigmoid $\sigma(z) = 1/(1+e^{-z})$ saturates rapidly if $|z| \gg 1$ creating an exploding or vanishing gradient, and therefore we used the following updated prior on the variance of weights.

$$D \cdot \sigma_w^2 = \text{var}(z) = 1 \implies \sigma_w^2 = 1/D \implies \sigma_w = 1/\sqrt{D}.$$

Bayesian Linear Discriminant Analysis

Linear discriminant analysis (LDA) is a generative model by design. That is, when using LDA we directly find the conditional likelihood of observations $p(x|y)$ and then derive the conditional likelihood of labels $p(y|x)$ by applying Bayes' rule. LDA was the only generative classifier we worked with in our BCI experiment, as all other models were discriminative since they were directly optimized for the conditional likelihood of labels. To read about the mechanics of LDA, see section C.4.

To implement Bayesian LDA, we assumed a Beta probability distribution on the prior of left and right hand motor imagery trials. The Beta distribution is the conjugate before the Bernoulli distribution that applies to binary classification [9]. Although our Bernoulli posterior is not solved in closed-form, the Beta distribution is a natural choice of prior since these two distributions make a coherent generative pair. We knew that all the datasets we used in our evaluation had equal class balance, and so we parameterized our Beta distribution with $\alpha = \beta = 2$ creating a weakly informative prior with regularization towards balanced labels.

LDA is characterized by the assumption that classes are generated by Gaussian probability distributions with separate means and a shared covariance matrix. That is, LDA is a special case of Gaussian mixture models (GMMs) with supervised learning. Therefore, we

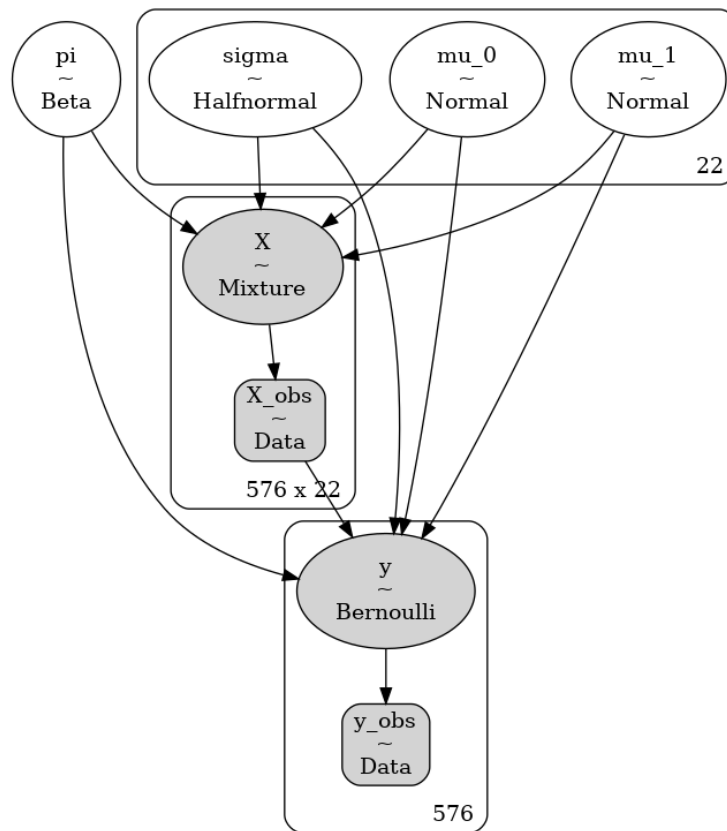


Figure 2.7: The Bayesian LDA PGM.

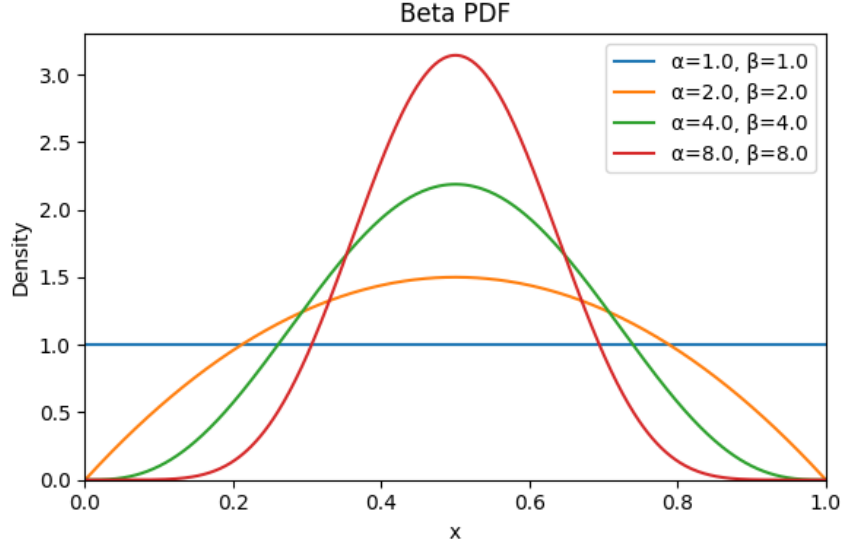


Figure 2.8: Examples of the Beta distribution with equal parameters $\alpha = \beta$ that approaches the Gaussian distribution as $\alpha = \beta \rightarrow \infty$ [57].

modeled Bayesian LDA as a mixture of Gaussians. This mixture of Gaussians defined the generative model for the likelihood of observations. To parameterize these Gaussians, we used location prior distributions and a scale prior distribution.

Our inputs were centered and scaled by z -score normalization before fitting the posterior distribution. Therefore, we believed that the mean of the class distributions would be centered at zero and would have a standard deviation of one. This mean $\mu = 0$ and standard deviation $\sigma = 1$ were used as hyperparameters for two standard normal distributions defining the location prior to each class label. Using separate distributions for the location priors was reasonable since LDA assumes that the means of class probability distributions can be different from each other.

For a shared scale prior distribution we applied a half normal distribution. The half normal distribution is strictly non-negative, corresponding to real variance of the covariance matrix shared by the Gaussian distributions model in LDA. Our use of the half normal

distribution for the scale prior was reasonable, since trials were additively spread around their mean. That is, the distance between trials was measured by a difference rather than a ratio. Since the trials were centered and scaled before being input, we assumed that the half normal probability distribution was parameterized by $\sigma = 1$.

Given our prior beliefs about class distributions $p(y)$, and our observed motor imagery trials with the left and right hand $p(x|y)$, we constructed the log-likelihood for the conditional likelihood of Bernoulli distributed labels $p(y|x)$ by applying Bayes' rule, completing the generative to discriminative conversion.

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}.$$

The derivation of the decision boundary for the LDA of binary classification which can be written as a linear function of x (see section C.4) allows us to cancel out the estimation of Bayesian evidence $p(x)$ which is generally intractable.

$$\begin{aligned} \log \frac{p(y = 1|x)}{p(y = 0|x)} &= \log \frac{p(x|y = 1)p(y = 1)}{p(x|y = 0)p(y = 0)} \\ &= \log p(x|y = 1) + \log p(y = 1) \\ &\quad - \log p(x|y = 0) - \log p(y = 0). \end{aligned}$$

Figure 2.7 shows the probabilistic graphical model for our Bayesian LDA classifier. We note the proportion of parameters (576×22) used to describe the generative components of the posterior compared to the parameters (576) for the discriminative component. This generative distribution of observations dominates the posterior compared to the discriminative distribution of labels. Indeed, as the only generative classifier in our study, the Bayesian LDA is the only estimator with competing objectives that define its posterior. In turn, when the Gaussian class conditional assumption of LDA is imperfect, the posterior can produce poorly calibrated probabilities.

Gaussian Process

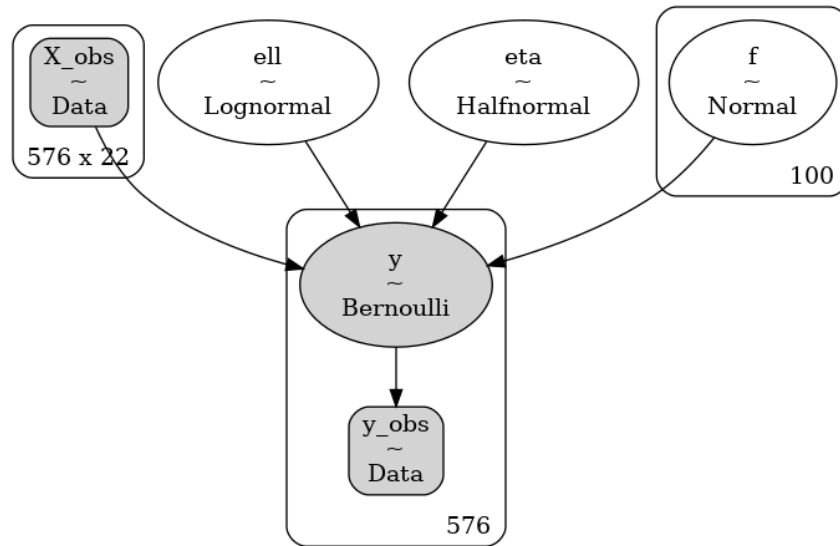


Figure 2.9: The GP with RBF kernel PGM.

In our frequentist model set we had two pipelines with support vector machine predictors. Namely, CSP features and an RBF kernel SVM, and TS features with a linear kernel SVM. We also explained that SVM decision boundaries do not naturally map to probabilistic calibration, as they are based on geometric optimization. Platt scaling is a solution, however, it is a post-one one. To see the learning algorithm for SVMs, visit section C.6.

The support vector machine belongs to the category of kernel based machine learning algorithms. This is a class of pattern recognition techniques in which the training points, or a subset of them, are kept and used during the prediction phase. The intuition of these methods is that kernels are formulated as an inner product in possibly infinite dimensional feature space where we substitute (known as the kernel trick) explicit calculation of the inner product with a kernel function whose output denotes the similarity between two inputs in the feature space [9].

Although support vector machines do not map to Bayesian modeling in a straightforward

manner, Gaussian processes (GPs) are a fully Bayesian kernel method. Indeed, there are properties of the inductive bias of both SVMs and GPs that make these kernel algorithms directly similar to each other [68]. Therefore, as a Bayesian analog to support vector machines in our experiment, we chose to implement Gaussian processes.

An in-depth description of Gaussian processes is found in section C.7. Generally, Gaussian process learning scales on a cubic scale $\mathcal{O}(N^3)$ where N is the number of trials in the case of MI-EEG data. For this reason, we needed to use a sparse approximation for our Gaussian processes. This was done using a fixed number of 100 inducing points, approximating the original dataset. We used the k -means algorithm, a popular choice to calculate the inducing points.

The literature offers little guidance on how to choose a fixed value of k in a principled fashion. For our evaluation, we generally used the PhysionetMI and BCNI2014.001 datasets consisting of 109 and 9 subjects, respectively, to assess the correctness of our Gaussian process implementation. Indeed, we used these two datasets to assess that all our Bayesian estimators were working. Specifically, we inspected that the posterior sampling graphs and MCMC metrics were reasonable and that the prediction scores were not unexpectedly below 50% or so as a result of software bugs.

These results did influence certain hyperparameters, prior distributions, and general implementation details. However, these design decisions were intended to ensure that our implementation of Bayesian pipelines was based on principles. That is, we did not observe the results of the evaluation with datasets besides PhysionetMI and BCNI2014.001, because we wanted to maintain that our pipelines were generally configured a priori, rather than tuning the parameterization to optimize for statistical significance in our meta-analysis.

In the case of our Gaussian processes, we found that 100 inducing points did not appear to grossly undershoot prediction results and at the same time kept computation tractable. Based on our review of the literature, existing software libraries, and open-source repositories available, we found examples with a fixed number of inducing points as low as 10–20. We also took into account that all inputs were centered and scaled with z -score normalization,

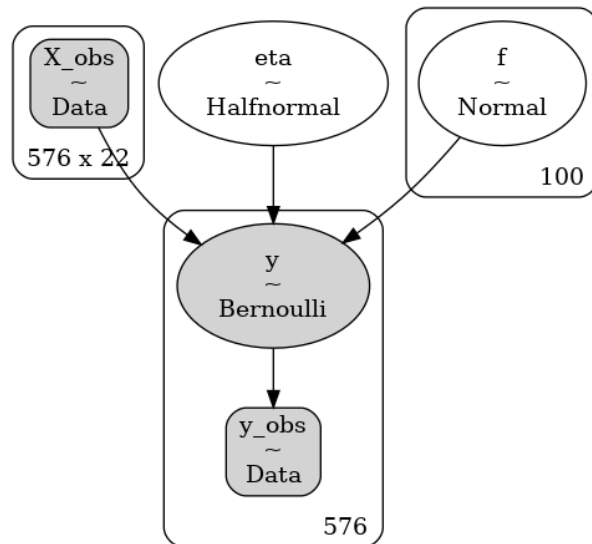


Figure 2.10: The GP with linear kernel PGM.

conditioning the input space to be isotropic. Given these conditions, we believed that $k = 100$ inducing points for all evaluations to be reasonably justified.

Our sparse approximation was based on the deterministic training conditional (DTC) method. It is common to find this algorithm implemented in probabilistic programming packages that offer Gaussian processes. DTC is a simple improvement over the deterministic inducing conditional (DIC) or subset of regressors (SOR) approximation which is known to be overconfident and underestimates the predictive variance. The fully independent training conditional (FITC) approximation further improves on DTC as it removes less uncertainty. However, FITC is more computationally demanding and more complicated to implement.

Furthermore, we also implemented non-centered parameterization. Given that Gaussian processes directly operate on the covariance matrix of observations, we had direct access to the full correlation structure. Using Cholesky factorization to reduce computational cost, we reconstructed the covariance matrix to be diagonal, so that observations were decorrelated. This diagonalization further approximates our Gaussian processes in addition to the use of

inducing points, however, in practice we found that it dramatically increased the speed of posterior sampling. Intuitively, this non-centered parameterization can be seen as smoothing the posterior distribution, and potentially increasing generalization.

With common spatial pattern features, we maintain an RBF kernel with our Gaussian process as we did with our support vector machine. Shown in Figure 2.9, we had separate length (**e11**) and signal (**eta**) prior distributions. The covariance function (**f**) reflects the inducing points and standard normal distribution on observation correlation with non-centered parameterization. The length scale **e11** is in ratio with observation distances, controlling how quickly covariance decays with distance. Our signal scale **eta** control the overall magnitude of the covariance.

$$k(x, x') = \eta^2 \exp \left(-\frac{(x - x')^2}{2\ell^2} \right).$$

For the prior probability distribution of our length scale **e11** we used a log-normal distribution. This is a common choice of prior distribution for distance scales controlled by ratios rather than differences. We considered multiple hyperparameter configurations for our logarithmic normal distribution, knowing that the input was z -score normalized. The 95% confidence intervals $[0.38, 2.66]$ around the log-normal centered at $\mu = 0$ and having $\sigma = 0.5$ were a good match for our use case. Furthermore, the centering and scaling of inputs led us to consider that a non-negative $\sigma = 1$ parameterized half normal distribution was a reasonable choice for the prior on our **eta** signal variance.

Our Gaussian with a linear kernel for tangent space features had inputs centered at zero and with a standard deviation of one. We included a prior probability distribution **eta** on the signal variance, as seen in Figure 2.10. This signal scale prior had the same role with our linear kernel GP as with our RBF kernel GP. As such, our reasoning on the suitability of a non-negative half normal distribution with a standard deviation $\sigma = 1$ was consistent.

$$k(x, x') = \eta^2 (x \cdot x').$$

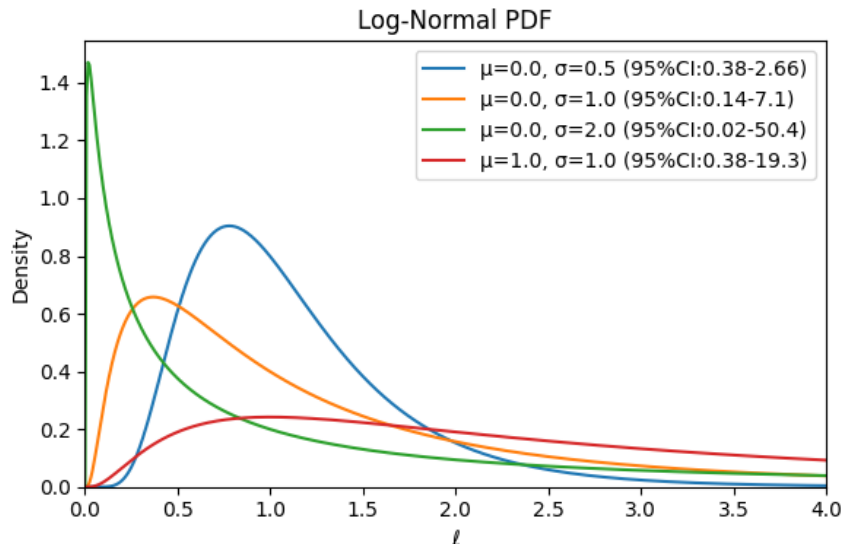


Figure 2.11: Sample of different parameterizations for a LogNormal distribution.

Bayesian Neural Network

There are many ways to implement Bayesian neural networks (BNNs). Perhaps the least tractable approach is to place probability distributions on all the backbone and classification parameters. For modern deep architectures with millions or billions of weights, this approach quickly becomes impractical. However, research from the BNN literature has shown that Bayesian last layer classifiers, that is, with deterministic feature extraction and probabilistic classification, often achieve prediction results that are comparable to fully Bayesian networks (see section C.8 for more details).

Our experiment used the same ShallowConvNet and DeepConvNet deep learning pipelines in our Bayesian evaluation as in our frequentist evaluation. We adapted these convolutional neural networks to have Bayesian last layer classification. In other words, our BNNs and all other Bayesian estimators maintained deterministic feature engineering and selection, and Bayesian prediction. However, deep learning generally encapsulates the optimization of feature extraction and class estimation. Therefore, we implemented our BNNs in two separate

steps.

In the first step, we fit the CNN fully deterministically. This stage optimized point estimate parameterization of backbone feature extraction before linear and deterministic classification with these features. That is, these features were tuned according to each dataset and the fold of cross-validation. This phase resulted in valuable learned information about the transformation and representation of raw left and right hand motor imagery EEG signals.

Our second major step in implementing Bayesian neural networks was to refit the linear classification layer with the optimized features acquired from the first step. However, we did not refit the classification layer with a standard gradient descent optimization algorithm characteristic of frequentist machine learning performing some type of maximum likelihood estimation. Rather, in keeping with the practice of Bayesian machine learning, we fit the classification weight space posterior probability distribution using Markov chain Monte Carlo sampling (see section C.1 for details).

Our Bayesian linear classification model is represented by Bayesian logistic regression in Figure 2.12. In section 2.3.2, we noted that this Bayesian model is identical to our Bayesian model used with Riemannian geometry tangent space features. In summary of this model, we used standard normal distributions as priors on our weights and bias parameters. However, specifically for our standard deviation initialization of the weights, we used $\sigma_w = 1/\sqrt{D}$ for all weights w_i where D was the number of trials in the training set.

When referring to Bayesian neural networks and our Bayesian logistic regression model above, we also noted that non-centered parameterization was necessary to fit our BNNs while preventing thousands of divergences during gradient based NUTS posterior sampling. Before reducing dimensionality at the paradigm level (see section 2.2), extracting tangent space features would result in predictors with up to thousands of dimensions. Our deep learning architectures would result in predictors with even more dimensions being extracted.

Especially high dimensional posteriors can create spaces that are difficult even for modern, sophisticated MCMC methods to sample. This is because the combination of dimensions can

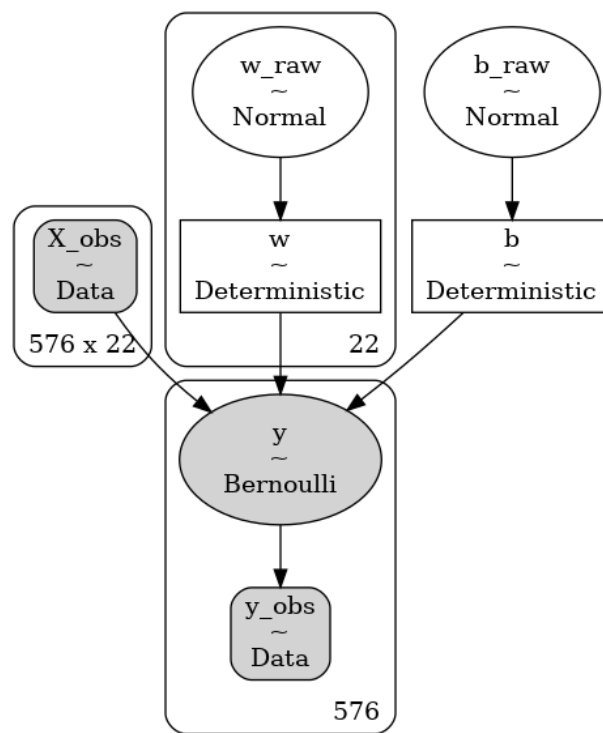


Figure 2.12: The Bayesian neural network PGM.

create regions that have a high curvature as the posterior extends throughout the space. The high curvature is numerically challenging to sample, as a small step in one direction can mean a divergent step reaching beyond the posterior surface in another direction.

In section C.1 we summarize the case of hierarchical models creating posteriors that can benefit from non-centered parameterization (NCP). Another scenario that benefits from NCP is high dimensional feature space, as we experienced when implementing our Bayesian neural networks. Before NCP, features could be characterized as anisotropic, with dimensions scaled according to their measurement. However, with non-centered parameterization, feature space can be made to be isotropic, and this is potentially helpful to MCMC samplers. When we applied NCP to our BNNs, we observed that thousands of sampling divergences vanish to zero.

Similarly to all other Bayesian models in our evaluation, we centered and scaled the features. In the case of our Bayesian neural networks, this looked like z -score normalization being applied to the feature extracted deterministically from the architecture backbone. Then, for Bayesian logistic regression, we decoupled the probability distributions of the weights and biases from other distributions. The result of moving from thousands of sampling divergences to zero tells us that non-centered parameterization helped to smooth the posterior distribution.

The Bayesian neural networks that we implemented were different from our other Bayesian pipelines in that our BNNs had two stages of fitting parameters: frequentist MLE gradient descent and Bayesian posterior sampling. If we had applied the optimized weights from point estimation of the classification layer as a starting point for Bayesian sampling, then we would have committed circular analysis, also known as statistical double dipping. That is because we would have fit the weights of the classification layer and then used these point estimates as hints for where to center the priors for fitting the Bayesian models.

Circular analysis results in overconfident models with higher rates of false positives (Type I errors). This results in a poorer calibration of the prediction models. Rather than reuse the fitted weights from the classification layer of deterministic optimization, we fit our Bayesian

logistic regression for BNNs starting with standard normal prior distributions. That is, centering and scaling the features extracted by the backbone, let us begin posterior sampling with standard normal priors on the weights, altogether avoiding circular analysis.

2.3.3 *Device Acceleration*

In general, we moved as much computation as we could out of our CPU host environment and into the GPU device configuration that we had access to. In section 2.4 we describe the resources available for our experiment. For our smallest datasets, the overhead of copying data between the host and the device may have cost more than its benefit. However, we believed that this cost would be relatively limited, with the gain being computation for our largest datasets and consistent evaluation conditions for all datasets.

Ineed, as we discussed above in the description of our SVM pipelines, we would have liked to include CuML in the evaluation of our point estimate support vector machine and logistic regression classifiers. Incidentally, CuML does not offer an implementation for GPU accelerated linear discriminant analysis. This means that we were always going to use scikit-learn to fit at least one of our pipelines. However, given the simplicity of fitting LDA with low computational demand, CPU bound execution is almost certainly a reasonable tradeoff in all practical cases.

In the absence of CuML, our pipelines trained with GPU resources were our frequentist deep learning classifiers, and all of our Bayesian models, or 8/12 (66.66%) of our pipelines. For our point estimate CNNs, PyTorch was the primary package on which we built and trained with GPU acceleration. With our Bayesian estimators, we wrote software with PyMC, and at runtime these models were compiled into computation graphs for GPU workloads with JAX. With our Bayesian neural networks, frequentist optimization was performed with PyTorch, and Bayesian sampling was done with PyMC, in a combination of GPU based computation.

The server we used for our evaluation had $4\times$ networked NVIDIA RTX A5000 GPUs. Our PyTorch based estimators were configured such that only one GPU was ever used for

calculation. Distributing work across multiple processors does not always result in faster completion, especially in cases when data or tasks are not easy to partition. Naively splitting work between processors can look like communication heavy, network bound execution. Rather than investigate whether our workloads could benefit from partitioning, we simply accepted the default settings of PyTorch, and in practice we found the execution time to be sufficient.

However, in the case of MCMC sampling, it is recommended to fit multiple chains so that the resulting posterior approximations can be compared. This presents an embarrassingly parallel scenario, where each chain can be fit simultaneously. Therefore, with our Bayesian models, we split weight space posterior sampling across 4 chains, one for each of the available GPU devices. During implementation, we did test runs with the default CPU bound PyMC sampler, the optimized but still CPU bound Nutpie sampler, and the GPU accelerated NumPyro sampler based on JAX. We found Nutpie to be surprisingly fast, though, for our largest datasets, NumPyro was still preferable.

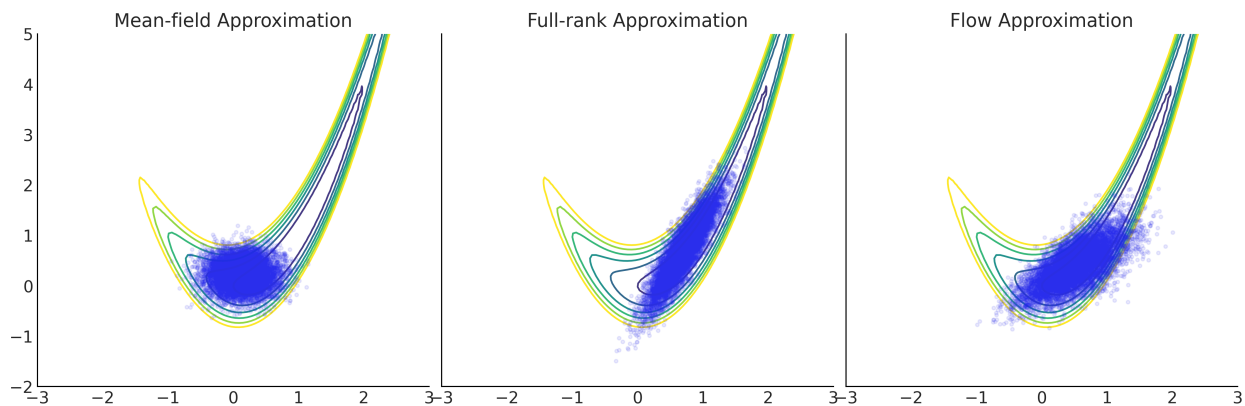


Figure 2.13: Three methods of variational inference that have been optimized to approximate a probability distribution and show various levels of precision [54].

PyMC primarily offers sampling methods for posterior approximations. These range across different base packages and implementations, and runtime assumptions. However, if

the developer wants, PyMC also offers various variational inference (VI) algorithms. However, these VI options do not come packaged with as many diagnostic tools and their implementations are strictly CPU bound. In general, MCMC is a method for finding a posterior probability distribution that is asymptotically exact, while at best VI is always an approximation [9].

MCMC is generally better understood than VI as MCMC has been explored in the literature since before VI [9]. However, at the same time, VI is increasingly attracting interest from researchers as it is more scalable than MCMC [58]. Modern deep learning architectures commonly produce parameter space distributions with millions or billions of dimensions, and this can be slow for MCMC methods to accurately sample. In summary, this is possible since VI does not attempt to directly approximate the true distribution, instead it chooses a tractable distribution and minimizes the Kullback-Leibler divergence (degree of separation) between the true and predicted distributions [9].

Early in the implementation of our Bayesian pipelines, we considered that using VI could be preferable over MCMC for posterior approximation since in theory it is more scalable. Our initial thoughts were that VI could be a better match for our larger models: Gaussian processes and Bayesian neural networks. That is, we think computation would be faster and that posterior approximation could be sufficient. We were interested to find that GPU accelerated NUTS was faster than CPU bound VI. Given that MCMC/NUTS is generally more accurate than VI, it was a win-win decision for us to adopt sampling algorithms for all of our Bayesian models.

2.3.4 Subprocess Isolation

Our initial implementation of our pipelines was to train all folds of all cross-validation loops in the same parent process. Each data set would be used one-by-one to train each pipeline in a sequence. The purpose of this configuration was to isolate our measurements of resource requirements for training with each pair of dataset and pipeline. However, when we started full-scale evaluation with all datasets and pipelines for the first time, we came across memory

leaks.

On inspection, we observed that after each fold of cross-validation from fitting Bayesian models, subsequent folds would require a few extra tens of MiBs of GPU memory. In total, training all pipelines with all datasets took over a week, but with this steady increase in GPU memory usage, execution would crash with out-of-memory errors after about one day. On rebooting the evaluation from the last checkpoint (paired dataset and pipeline) before exiting, we would then observe that the GPU memory continued to climb again.

These circumstances told us that there was a GPU memory leak. In addition to being inconvenient to restart evaluation every day for more than a week, we did not like the idea that this memory bottleneck could affect resource usage measurements of evaluation between folds and across datasets and pipelines. It is possible that the memory leak was a remnant of caching using JAX just-in-time (JIT) compilation and/or directed acyclic graph (DAG) computation. That is, we believed that the memory leak was a fragment of PyMC, NumPyro, or JAX execution, or some combination of their usage.

In response, we tried various programmatic changes at the level of Python sourcecode and environment configurations in attempts to force the release of GPU memory. Finally, and somewhat last resort, we considered spawning separate child processes for each fold to be fitted in as this would ensure that the operating system would take back all memory when processing returned to the parent process. This option was not ideal because it also meant that we would have to build a way to communicate specific information between processes.

In the end, spawning an isolated subprocess in which each fold was fitted solved our GPU memory leak. That is, forking a subprocess that shares certain addresses and data with its parent process was insufficient to solve our memory leak. A spawned child process with new resources was needed. To implement this without forking a potentially messy feature branch of the MOABB evaluation loop, we wrapped our pipelines with a scikit-learn interface and internal logic that encapsulated child processing around fitting the ML models.

Given that model estimation and scoring occur in separate steps, we needed to be able to pass fitted model weights across processes. For this purpose, we saved this information

to disk, with additional details like sampling traces for Bayesian models also being saved to disk. In a separate subprocess, the classifiers would be reconstructed for scoring. Spawning child processes for testing models in addition to training was done in interest of defensive programming rather than in response to any software bugs.

Using disk to communicate across processes is not efficient. However, it was easy. However, it also gave us a way to persist our learned models beyond scoring once and then releasing. If we ever wanted to collect additional scores, we would be able to do so without refitting the models. In general, spawning child processes was a fine solution because the purpose of our experiment was to benchmark frequentist and Bayesian pipelines, not optimize evaluation speed, even though training and testing speed was a practical nonissue anyway with this solution.

2.4 Evaluation

Our experiment used cross subject evaluation to measure the performance of frequentist and Bayesian pipelines. In general, EEG signals have a low signal-to-noise ratio (SNR), making classification a difficult task (see section A.1 for an explanation). The prediction of a single subject within one session offers the highest SNR. This often leads to higher classification scores, such as AUROC or accuracy. However, within session evaluation is also the least practical application of machine learning for BCI [21].

Many factors can lead to heterogeneity in EEG signals between sessions and also between subjects. For a given subject, the EEG signals can vary between sessions due to differences in cognitive states, emotions, or slight changes in experimental conditions. Cross session evaluation is seen as a middle ground in terms of SNR for EEG BCI research. It is also more realistic than within session classification. For example, in online learning scenarios, BCI ML algorithms can be adapted between sessions so that classifiers are personalized to individual subjects [56].

The most general form of ML research for EEG is cross subject evaluation. It offers widely applicable BCI algorithms, since its potential is that classification models fit to the

global population. This broad generalization task also makes cross subject evaluation the most difficult type of EEG evaluation to build with accurate and reliable prediction scores. On the other hand, cross subject evaluation is the easiest evaluation for which to collect ample data since this involves pooling trials from available sessions and subjects.

The potential of foundation models for EEG classification trained in a cross subject evaluation fashion is highly appealing to BCI researchers. That is because it is relatively low cost to fit one large classifier that can then be widely distributed to the general population and deliver high performing personalization with some form of transfer learning. Indeed, Bayesian ML is a theoretically promising technique for adapting foundation models to new environments based on model space learning regularization that accounts for prior and posterior beliefs [47].

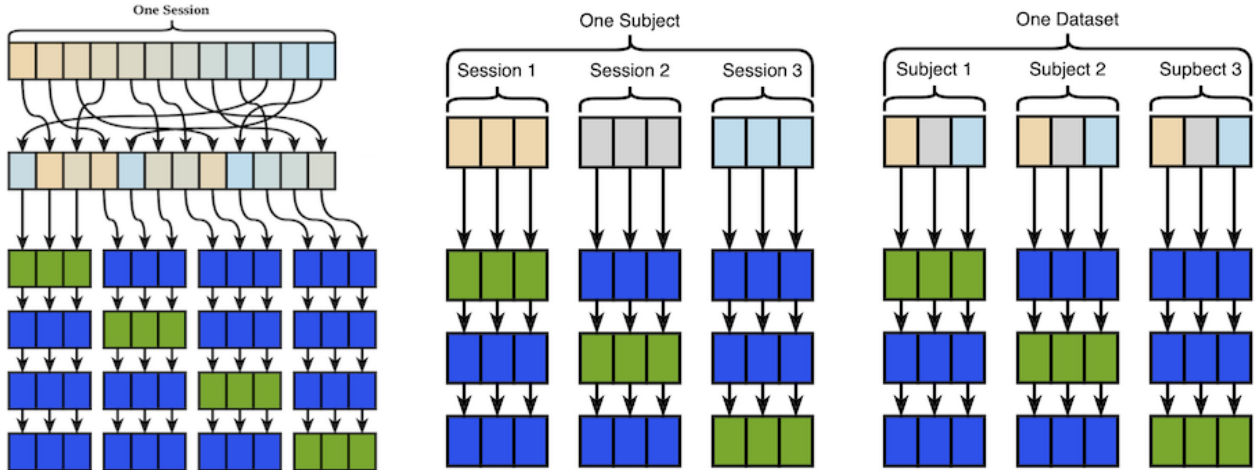


Figure 2.14: All evaluations are available through MOABB [21].

MOABB offers all three types of evaluations: within session, cross session (also known as within subject), and cross subject. These are visualized in Figure 2.14 according to how MOABB implements these separate evaluations. Since our experiment used a cross subject evaluation, our models learned in high variation, low SNR space. This is the most complex environment in which to fit BCI classifiers, and we assumed that successful results in this

setting would also apply to within and cross session evaluations.

2.4.1 Cross-Validation

The MOABB evaluation loop uses k -fold cross-validation. According to its open-source software repository, when k is configured to be the number of subjects, then a leave-one-group-out (LOGO) evaluation is applied. LOGO is generally only used when a dataset is especially small. In this case, it looks like one subject being left of training, and then used as the only subject during testing. For all evaluations in MOABB, this specifically looks like all the trials belonging to subjects either being assigned to the training or testing sets but never both.

Alternatively, when the number of folds in a MOABB evaluation is configured to be less than the number of subjects in a dataset, then k -fold cross-validation with non-overlapping groups is applied. For MOABB this again looks like the trials of one subject being held out of training and used for testing. However, unlike LOGO, not all subjects are tested against. Furthermore, in all evaluations, MOABB uses outer k -fold cross-validation and inner 3-fold cross-validation.

The inner cross-validation built into MOABB is used for hyperparameter optimization, where the highest performing hyperparameters are returned to the outer k -fold cross-validation. For this purpose, inside the inner cross-validation, an additional subject is removed from the training set and used for validation. This nested cross-validation is recommended for small datasets (see section D.1 for details on cross-validation and a discussion including nested cross-validation).

For our experiment, we chose to use LOGO when the number of subjects in a dataset was less than ten, 5-fold (outer) cross-validation for less than fifty subjects, and 10-fold cross-validation otherwise. This set of rules is summarized in the following formula. We chose the first constraint for datasets with less than ten subjects because it means that all our evaluations are bounded between $5 \leq k \leq 10$ for k -fold cross-validation.

The second condition determining whether the datasets would be evaluated with 5-fold

or 10-fold cross-validation was motivated by the following observation. There were several datasets with the number of subjects being between 20–40. However, there were no datasets where the number of subjects was between 40–50. Therefore, we made the opinionated decision that large datasets should be defined as those with more than 50 subjects, leaving a gap of at least 10 subjects between datasets categorized as medium sized and large sized.

$$\text{No. folds} = \begin{cases} \text{No. subjects} & \text{if No. subjects} < 10, \\ 5 & \text{if No. subjects} < 50, \\ 10 & \text{Otherwise.} \end{cases}$$

2.4.2 Runtime Environment

We tracked the duration, CPU power and energy usage, GPU power and energy usage, and the converted carbon emissions of our server for each fold of cross-validation through the CodeCarbon PyPI package. During our evaluation, we were the only researchers using the server, and we performed the experiment on bare metal resources. MOABB integrates the optional use of carbon emissions tracking with CodeCarbon. To collect the additional fine grained metrics for CPU and GPU usage, we implemented a feature request on a fork of the MOABB GitHub repository. The relevant MOABB documentation was also updated. These changes for fine grained compute tracking were released in MOABB PyPI version 1.5.

Our instance of CodeCarbon was configured to convert CPU and GPU usage from our server to carbon emissions according to emissions reports from Washington, USA, which CodeCarbon records. Although the server used in our experiment was offered through the University of Washington and was located in Seattle, WA, the finest grain emissions reporting that CodeCarbon records for its emissions calculations is recorded at the state level. CodeCarbon was a convenient tool to perform compute profiling of our ML workloads.

The operating system used by our server during the experiment was a release of Oracle Linux 8 Yum Server. Our Python version was 3.12.12. We had 72× Intel Xeon Gold 6354

Table 2.12: The number of folds used in cross-validation according to the size of the dataset.

No. Folds per Dataset		
Dataset	No. subjects	No. folds
BNCI2014_001	9	9
BNCI2014_004	9	9
Brandl2020	16	5
Chang2025	28	5
Cho2017	52	10
Dreyer2023	87	10
Forenzo2023	25	5
GrosseWentrup2009	10	5
GuttmannFlury2025_MI	31	5
HefmiIch2025	37	5
Kumar2024	18	5
Lee2019_MI	54	10
Liu2024	50	10
PhysionetMI	109	10
Schirrmeister2017	14	5
Shin2017A	29	5
Stieger2021	62	10
Weibo2014	10	5
Yang2025	51	10
Zhou2020	8	8

Table 2.13: The training and testing sample splits averaged over k -fold cross-validation.

Mean Splits of Cross-Validation		
Dataset	Avg. No. training samples	Avg. No. testing samples
BNCI2014_001	2304	144
BNCI2014_004	5795.56	144.89
Brandl2020	6446.4	1611.6
Chang2025	1808	105.67
Cho2017	9468	1052
Dreyer2023	18712.6	2079.2
Forenzo2023	2928.8	366.1
GrosseWentrup2009	2400	600
GuttmannFlury2025_MI	2016	168
HefmiIch2025	2352	105.6
Kumar2024	5724.8	238.53
Lee2019_MI	9720	540
Liu2024	1800	200
PhysionetMI	4426.2	491.8
Schirrmeister2017	5393.6	1348.4
Shin2017A	1392	116
Stieger2021	39079.8	723.7
Weibo2014	1264	316
Yang2025	27531.9	1019.7
Zhou2020	5832.75	121.48

CPUs at 3 GHz clock rate available to us. Additionally, there were $4\times$ Nvidia RTX A5000 GPUs available. These specifications were recorded by CodeCarbon. A full record of all packages with version used in our Python environment for evaluation can be found in our GitHub repository [26].

2.5 Analysis

Our evaluation trained 6 pairs of frequentist and Bayesian models on 20 different datasets. For each dataset and pipeline combination, we collected 6 metrics from the testing. Given that the datasets had heterogeneous qualities, we could not have pooled all the datasets into one primary study. Qualities that varied between datasets included, but were not limited to, the number of sessions, runs, and length of trials, the number of sensors and their locations, and the hardware and software of the BCI device. This variability across datasets allowed the results of our experiment to be widely generalized across the domain of left hand and right hand MI-EEG classification.

To accumulate the results of the evaluation on each dataset, we used meta-analysis instruments (we recommend section E.1 for a detailed foundation on meta-analysis hypothesis testing). Each dataset in our experiment was treated as an independent primary study. Within studies, the raw difference in prediction performance metrics between paired Bayesian and frequentist pipelines would contribute to an effect size. To be clear, within each dataset the scores from individual folds of cross-validation were averaged to remove between fold correlation. The resulting average score was used to compute the arithmetic difference between Bayesian and frequentist performance.

Therefore, we had 6 data points (one per pair of ML models) within the studies. That is, the scoring was done separately across the 6 metrics that we collected. For each prediction performance metric, we conducted an isolated meta-analysis, which gave us 6 completed meta-analyses. This experiment design gave us 6 data points per study that were independent in terms of cross-validation fold-level correlation. Our averaging across k -fold cross-validation scores is common practice in ML research, and simplified the structure of our meta-analyses.

Within the datasets, we further grouped data points by their corresponding pipeline family (see section 2.3). Each pipeline family in our experiment shared similar feature engineering and selection. This similarity created a natural clustering of effects. Therefore, our experiment was structured as 3-level meta-analyses. At level-3, we model the heterogeneity between studies that correspond to variability between datasets. Modeling for level-2 measured heterogeneity across pipeline families within datasets. Then, level-1 modeled sampling error which is assumed to always be present.

Indeed, MOABB offers meta-analysis tools which we could have relied on for our experiment. However, to accumulate results across datasets, MOABB relies on Stouffer’s z -score method to combine p -values of paired model comparisons [21]. Generally, Stouffer’s z -score method should only be considered for use when an effect size is not directly available [12]. That is because a (combined) p -value only tells us whether there was an effect, but does not tell us anything about the effect size. In the end, it is the effect size rather than the existence of an effect that drives interest in an alternative hypothesis.

Furthermore, the approach taken by MOABB that combines p -values across datasets for single pairs of pipelines would have offered more limited conclusions than we could infer from our experiment design. Using the statistical analysis offered by MOABB currently, our conclusion about the benefits of Bayesian ML would have been more narrative, as the pairwise p -values of ML model comparisons would have been spread over one meta-analysis per pair for each prediction performance metric.

With our approach, in addition to directly analyzing effect sizes, we were able to pool the effects of paired ML model comparisons. This allowed us to quantitatively answer a more general question about Bayesian ML versus frequentist ML for left hand and right hand MI-EEG classification, as opposed to a qualitative description that would have grouped the effects of separate meta-analyses for each pair of ML models per metric. Our experiment design marked a major divergence from the tooling offered by MOABB.

2.5.1 Data Wrangling

Given the results of k -fold cross-validation of MOABB, we averaged the training and testing sample sizes, duration and carbon emissions, and scores of the prediction performance metrics across all folds when grouped by dataset and pipeline. Then we computed the difference in duration, carbon emissions, and scores between paired Bayesian and frequentist pipelines within the datasets. Finally, grouping by dataset and pipeline family, we calculated the average duration, carbon emissions, and scores, along with the sample variance of the scores. This generated the data that we used for our meta-analyses.

2.5.2 Moderator Analysis

In addition to intercept-only models, we computed meta-regressions as part of our analysis. Our intercept-only models simply considered the effect of Bayesian ML versus frequentist ML for left and right hand MI-EEG classification. As a meta-regression, we also considered how the effect changed with respect to the training sample size. In section 2.2 and section 2.4 we showed that the number of trials for training sets ranged from a little more than a thousand to tens of thousands, which is proportional to a gain of $10\times$.

Although we had data to conduct a meta-regression based on training cost such as duration or carbon emissions, we felt that it could misrepresent the effect of Bayesian and frequentist ML. That is because these measurements are largely dependent on the silicon implementation. For example, certain algorithms and software packages had access to run on GPUs rather than CPUs, greatly reducing the duration and correlated carbon emissions. Recall our discussion in section 2.3 about not using CuML to fit SVMs on GPUs as we originally planned, which results in SVMs being trained with scikit-learn using CPUs. The result was much longer training durations for our largest datasets.

Rather than performing any meta-regression with respect to computation cost metrics, we simply recorded these measurements. Interested readers are able to see the duration, carbon emissions, and fine grained power and energy usage of CPUs and GPUs by our

server. However, analysis with these measurements is narrative and interpretation is left largely to the reader. In the current state of green ML research, comparisons are often made between apples and oranges.

2.5.3 Robust Testing

Given a multi-level meta-analysis, there are certain cases where modeling correlation between sampling errors should be considered. Indeed, this so-called correlated and hierarchical effects (CHE) model can make some experiments more realistic. For our evaluation, we considered that prediction performance scores within datasets relied on the same fold data to be generated. That is, although we averaged the correlation within k -fold cross-validation, the resulting average scores still came from evaluating the same training and testing sets. Therefore, we believe it to be worthwhile to model the 1-level sampling error correlation within datasets.

With 20 datasets in our experiment, we were at the edge of it being reasonable to apply the analytical framework of robust variance estimation (RVE) to our results. We used the bias-reduced linearization or CR2 method for small sample adjustment in the sandwich estimator. As a sensitivity analysis, we iterated over $\rho = 0.2, 0.4, 0.6, 0.8$ as inputs for the constant sampling correlation assumption. Consistent results from a sensitivity analysis would validate the robustness of our conclusions on the sign, significance, and magnitude of effects.

In addition to robust variance estimation, we also applied cluster wild bootstrapping (CWB) for our meta-regression. For sample sizes less than 40 or so, the CWB can offer better estimates from its numerical framework. As a robustness check, we also used $\rho = 0.2, 0.4, 0.6, 0.8$ in separate tests for sensitivity analysis. With 20 datasets in our experiment, quantitative CWB results would be more reliable than RVE, but seeing similar qualitative explanations about the sign and significance of results would be reassuring. We used 2000 iterations for each of our CWB estimations.

2.5.4 Metrics Reporting

We used R to program all of our meta-analyses. The `metafor` package was the backbone for each of our intercept-only and meta-regression models. For robustness checks with correlated and hierarchical models, we used the `clubSandwich` package for robust variance estimation and the `wildmeta` package for cluster wild bootstrapping. Considering that we had only 20 studies for our estimations, we used the Knapp-Hartung adjustment based on the t -distribution for all of our meta-analyses, rather than Wald tests which assume a normal distribution. Furthermore, we used the restricted maximum likelihood (REML) estimator for aggregating effect sizes, given its robust performance with continuous outcomes.

The p -value of the pooled effect and the size of the pooled effect with 95% confidence intervals were recorded for all meta-analyses. We also recorded the 2-level and 3-level Higgins and Thompson I^2 statistic to measure the percentage of variability in the pooled effect sizes that was not explained by sampling error. Additionally, prediction intervals, which are insensitive to the number of studies and their precision, were measured giving us a range into which we could expect the effects of future studies to fall based on our acquired evidence. Additionally, we collected the pooled standard error and degrees of freedom from our meta-analyses.

Chapter 3

RESULTS

3.1 *Posterior Sampling*

In Table F.1 – Table F.10 of section F.1 we summarize the diagnostics from the MCMC sampling of the posterior probability distributions of weights for our Bayesian models. Grouping by dataset and pipeline, we computed worst case diagnostics across k -fold cross-validation. See subsection C.1.3 for a description of the effective sample size of the bulk and tail (ESS), the potential scale reduction factor $\hat{R}$, the Monte Carlo standard error (MCSE), and the divergence. Across all combinations of datasets and pipelines, 113/120 (94.17%) and 119/120 (99.17%) of minimum bulk and tail ESS values were at least 400, respectively. Furthermore, 118/120 (98.33%) of maximum $\hat{R}$ results are at most 1.01.

In addition, we created trace plots, autocorrelation plots, and rank plots (see subsection C.1.3 for an explanation of these plots). The results of these graphs for all pipelines are found in section F.1. Interestingly, only the plots for BNCI2014_001 are shown. As a golden standard dataset for MI-EEG research, BNCI2014_001 is a representative choice among all datasets we used. Since summary diagnostics were largely consistent across all datasets, the convergence plots were also broadly similar. An analogous statement can be made for parameters in matrix form, for which we plotted the zeroth index vector. Note that the convergence plots for all datasets can be found in our Figshare [25].

Our summary diagnostics and convergence plots tell complementary stories. In general, the trace plots showed good mixing across chains. The fuzzy caterpillars from trace plots of Bayesian logistic regression, Bayesian ShallowConvNet, and Bayesian DeepConvNet were notably straight and tight, showing that the posterior was well-characterized. Chains for these Bayesian models were able to explore the posterior efficiently with each sample rela-

tively independent from the last (low autocorrelation) and steps being appropriately sized. Together, this translates into a large effect sample size.

However, the trace plots of our Gaussian processes were more zigzag. This is a characteristic of slower mixing between chains. In these cases, chains may get stuck more often or reverse direction frequently, meaning consecutive samples are more correlated (high autocorrelation), and the ESS is lower relative to total samples. Common causes include difficult posterior geometry and complex models. The following sections quantitatively analyze the summary of our sampling diagnostics from Table F.1 to Table F.10. In subsection 3.1.1 we consider the results when grouped by pipeline, and then subsection 3.1.2 inspects our summary of diagnostics when grouped by dataset.

3.1.1 Diagnostics Dispersion: Pipelines

Table 3.1: The summary statistics of bulk ESS across all datasets.

Pipeline Summary Statistics: Bulk ESS					
Pipeline	Min.	Max.	Median	P05	P95
BDCNN	2856.0	5859.0	4264.0	2908.25	5710.8
BSCNN	3550.0	6163.0	4502.0	3590.85	6060.4
CSPBLDA	959.0	3190.0	1572.0	1100.55	3104.5
CSPGP	230.0	1305.0	635.0	299.35	1204.3
TSBLR	2371.0	6531.0	4727.0	3223.15	6527.2
TSGP	349.0	2391.0	871.5	351.85	1478.05

Figure 3.1 and Figure 3.2 visualize the outcomes of Table 3.1 and Table 3.2 for bulk and tail ESS when the summary of our sampling diagnostics is grouped by pipelines. These plots and tables quantify the same narrative description that can be seen in our convergence

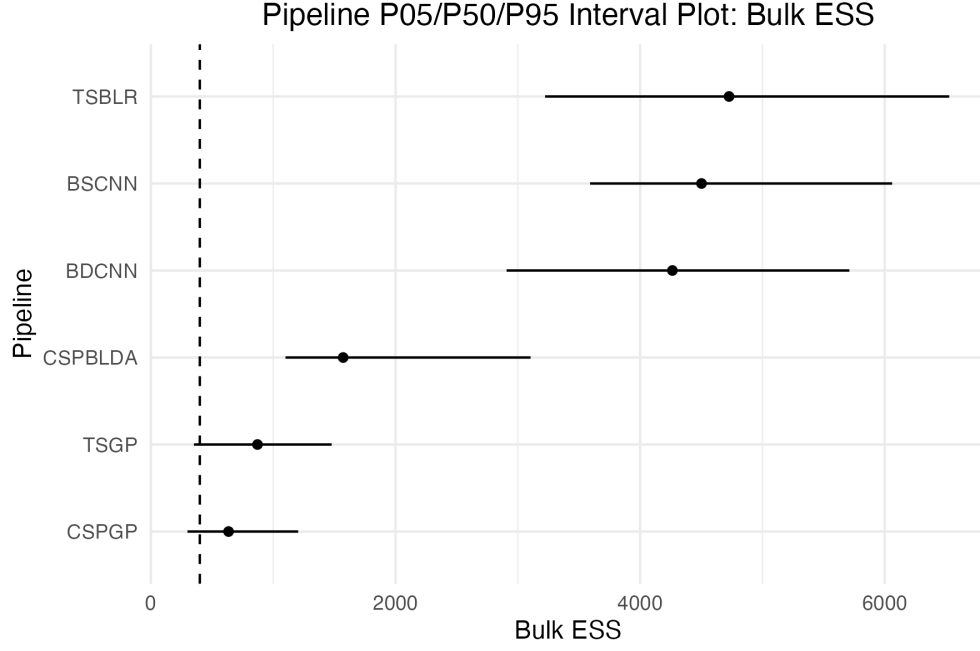


Figure 3.1: The interval plot of bulk ESS across all datasets.

Table 3.2: The summary statistics of tail ESS across all datasets.

Pipeline Summary Statistics: Tail ESS					
Pipeline	Min.	Max.	Median	P05	P95
BDCNN	1398.0	2352.0	1986.0	1509.15	2189.55
BSCNN	1743.0	2106.0	1957.5	1743.0	2033.8
CSPBLDA	914.0	2601.0	2092.0	1420.35	2553.5
CSPGP	198.0	1848.0	1158.0	437.4	1830.9
TSBLR	1975.0	2614.0	2194.0	2036.75	2600.7
TSGP	526.0	2282.0	1589.0	653.3	2009.35

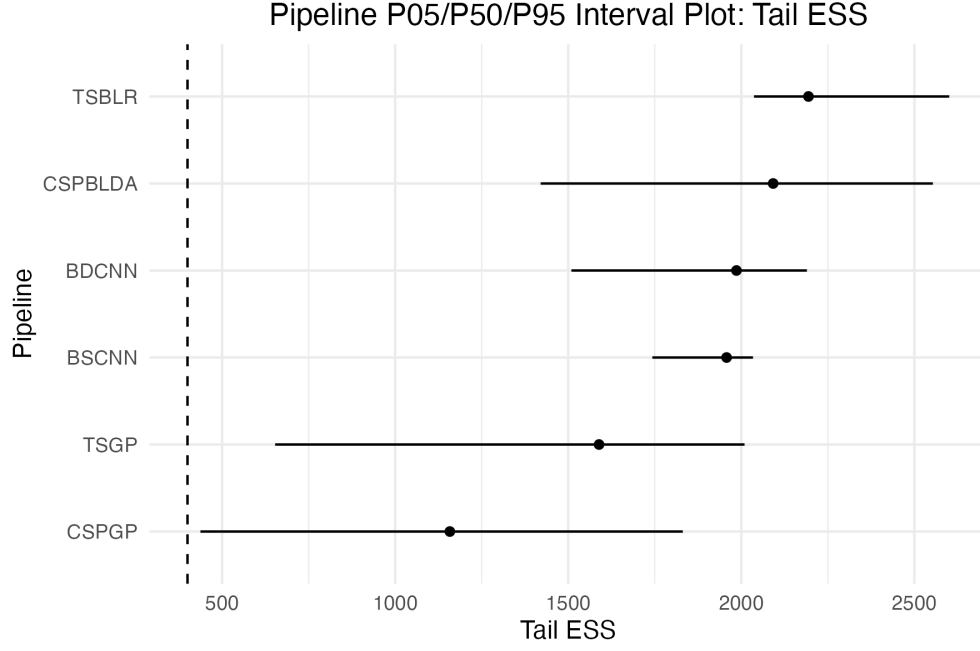


Figure 3.2: The interval plot of tail ESS across all datasets.

Table 3.3: The summary statistics of the mean MCSE in all datasets.

Pipeline Summary Statistics: Mean MCSE					
Pipeline	Min.	Max.	Median	P05	P95
BDCNN	0.0	0.002	0.001	0.0	0.002
BSCNN	0.0	0.001	0.001	0.0	0.001
CSPBLDA	0.0	0.003	0.001	0.0	0.002
CSPGP	0.012	0.059	0.017	0.012	0.038
TSBLR	0.0	0.002	0.001	0.0	0.002
TSGP	0.011	0.042	0.013	0.011	0.024

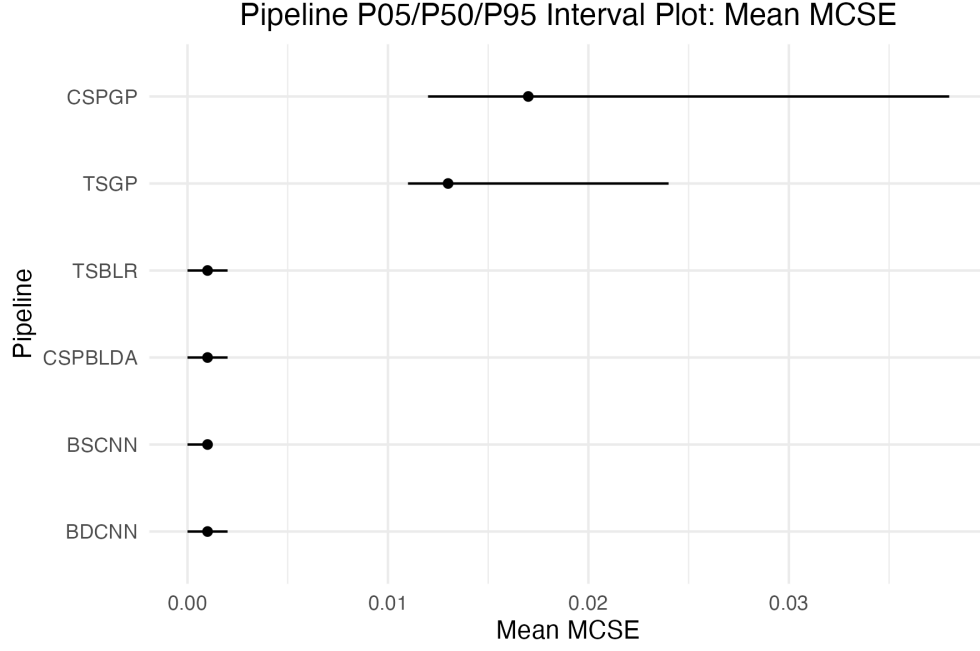


Figure 3.3: The interval plot of mean MCSE across all datasets.

Table 3.4: The summary statistics of std. MCSE across all datasets.

Pipeline Summary Statistics: Std. MCSE					
Pipeline	Min.	Max.	Median	P05	P95
BDCNN	0.0	0.003	0.001	0.001	0.002
BSCNN	0.001	0.001	0.001	0.001	0.001
CSPBLDA	0.0	0.002	0.001	0.0	0.001
CSPGP	0.015	0.029	0.02	0.016	0.028
TSBLR	0.001	0.002	0.001	0.001	0.002
TSGP	0.018	0.027	0.022	0.019	0.025

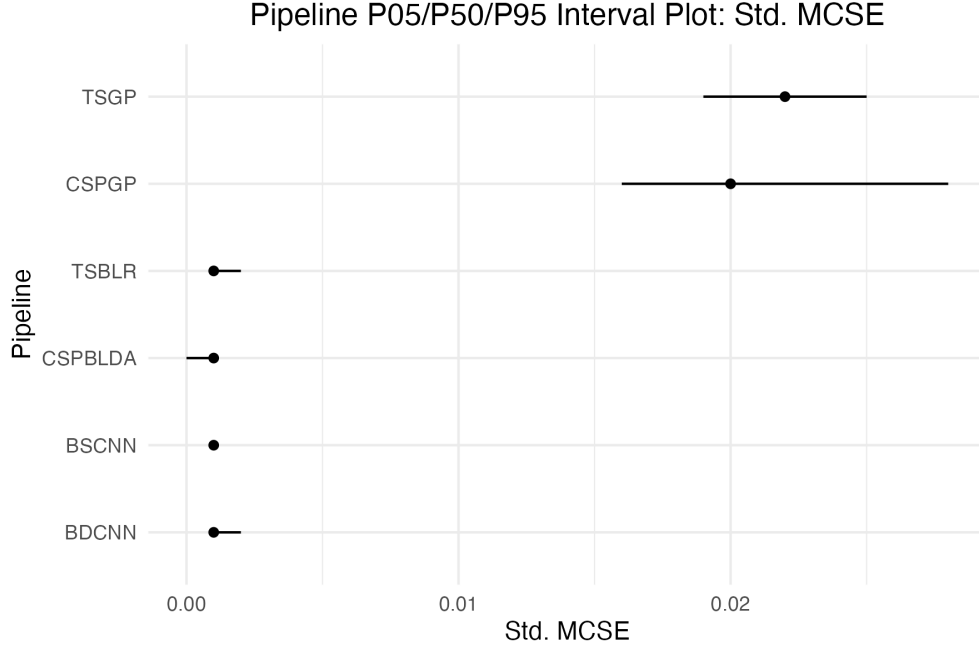


Figure 3.4: The interval plot of std. MCSE across all datasets.

plots from section F.1. Indeed, the 95% percentile interval of bulk ESS for Bayesian logistic regression based models and our Gaussian processes do not overlap. Furthermore, the 5% percentile of our Gaussian processes slightly extends below the standard lower threshold of 400 for ESS.

The tail ESS performance of Gaussian processes also ranks last among our pipelines when worst case sampling diagnostics are pooled across all datasets. For tail ESS grouped by pipelines, we also see that Bayesian logistic regression was the highest performer, but that our Bayesian deep learning classifiers fell below Bayesian LDA. This was perhaps a product of the Bayesian deep learning models having a higher dimensional parameter space that was more difficult for MCMC NUTS samplers to efficiently explore.

Considering that well over 90% of our bulk and tail ESS scores were at least 400 and similarly that $\hat{R}$ scores were at most 1.01, it was reasonable for us to consider the MCSE results. We again saw that our Gaussian processes were the lowest performers with respect to

mean and standard deviation MCSE. Figure 3.3 and Figure 3.4 show that the 95% percentile intervals of these aggregated diagnostics for Gaussian processes did not overlap with any other pipelines. However, at the same time, mean and standard deviation MCSE results less than 0.05 are low, regardless of their difference relative to other estimators.

3.1.2 Diagnostics Dispersion: Datasets

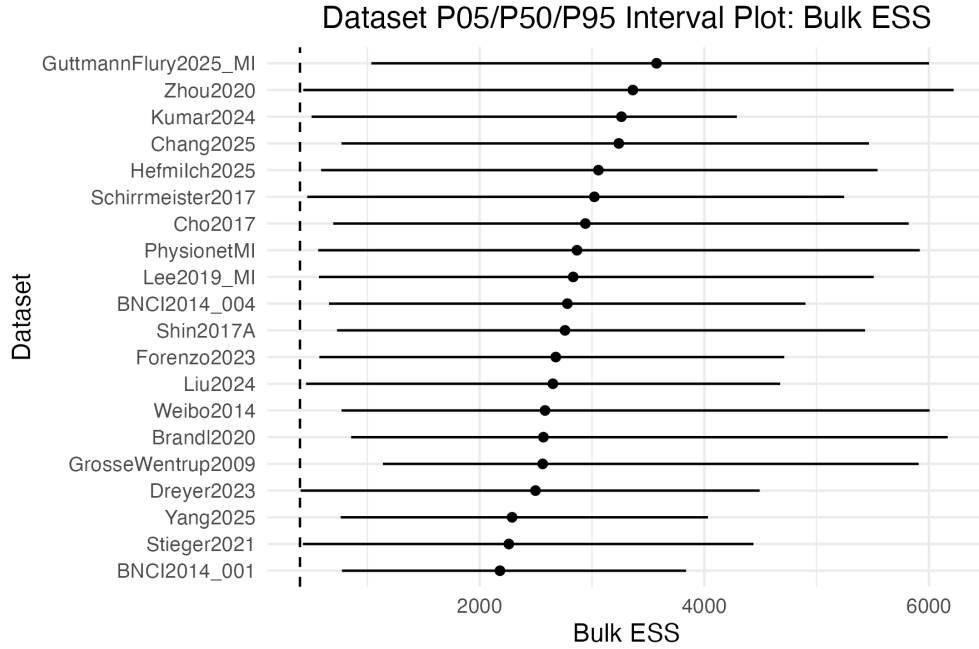


Figure 3.5: The interval plot of bulk ESS across all pipelines.

Generally, grouping worst case sampling results by pipelines revealed clear quantitative patterns as discussed in subsection 3.1.1. Indeed, we saw qualitative representations of these patterns in the convergence plots of section F.1. Likewise, the convergence plots for all datasets found in our Figshare, qualitatively suggested what we quantitatively derived in Figure 3.5 – Figure 3.8 plotting the results of Table 3.5 – Table 3.8 [25]. In other words, these plots and tables quantitatively summarize that more or less the convergence plots of

Table 3.5: The summary statistics of bulk ESS across all pipelines.

Dataset Summary Statistics: Bulk ESS					
Dataset	Min.	Max.	Median	P05	P95
BNCI2014_001	712.0	3863.0	2180.5	773.75	3837.50
BNCI2014_004	549.0	5231.0	2780.5	658.25	4900.75
Brandl2020	777.0	6168.0	2567.5	855.50	6166.75
Chang2025	651.0	5489.0	3238.5	770.00	5466.00
Cho2017	568.0	6527.0	2941.0	695.75	5819.50
Dreyer2023	378.0	4793.0	2497.0	405.75	4493.00
Forenzo2023	350.0	4755.0	2677.5	572.75	4711.50
GrosseWentrup2009	1118.0	6531.0	2561.0	1138.25	5907.75
GuttmannFlury2025_MI	584.0	6514.0	3574.5	1035.75	6001.00
HefmiIch2025	349.0	5703.0	3057.0	588.00	5542.00
Kumar2024	432.0	4443.0	3262.0	503.75	4289.75
Lee2019_MI	547.0	6128.0	2832.5	568.50	5508.00
Liu2024	303.0	4797.0	2651.5	454.00	4675.25
PhysionetMI	380.0	6258.0	2866.0	562.00	5918.25
Schirrmeister2017	443.0	5448.0	3021.0	464.75	5244.75
Shin2017A	696.0	5531.0	2759.5	730.25	5431.75
Stieger2021	352.0	4631.0	2260.0	427.25	4437.50
Weibo2014	637.0	6055.0	2582.0	770.25	6006.00
Yang2025	738.0	4125.0	2288.5	762.50	4032.25
Zhou2020	230.0	6400.0	3363.5	429.50	6219.50

Table 3.6: The summary statistics of tail ESS across all pipelines.

Dataset Summary Statistics: Tail ESS					
Dataset	Min.	Max.	Median	P05	P95
BNCI2014_001	914.0	2190.0	1988.0	1034.25	2165.00
BNCI2014_004	680.0	2340.0	1944.0	903.50	2308.75
Brandl2020	1571.0	2139.0	1869.0	1615.75	2101.00
Chang2025	1357.0	2601.0	2134.5	1518.00	2521.25
Cho2017	1126.0	2320.0	1893.5	1240.50	2266.50
Dreyer2023	523.0	2222.0	2015.0	580.75	2179.25
Forenzo2023	198.0	2228.0	1926.0	567.50	2216.25
GrosseWentrup2009	1470.0	2092.0	1877.0	1550.50	2066.00
GuttmannFlury2025_MI	700.0	2485.0	2010.5	928.25	2378.50
HefmiIch2025	526.0	2174.0	1840.5	744.00	2125.00
Kumar2024	1032.0	2600.0	1921.0	1102.25	2560.50
Lee2019_MI	1064.0	2367.0	2065.5	1096.50	2310.00
Liu2024	790.0	2614.0	1859.5	983.00	2493.75
PhysionetMI	737.0	2040.0	1716.5	914.50	2013.75
Schirrmeister2017	775.0	2336.0	1892.5	854.50	2260.00
Shin2017A	1160.0	2551.0	1717.5	1268.50	2523.75
Stieger2021	660.0	2429.0	1862.5	798.75	2323.50
Weibo2014	1156.0	2237.0	1807.0	1286.50	2174.25
Yang2025	1256.0	2352.0	2020.0	1349.00	2342.00
Zhou2020	450.0	2151.0	1862.0	797.25	2109.50

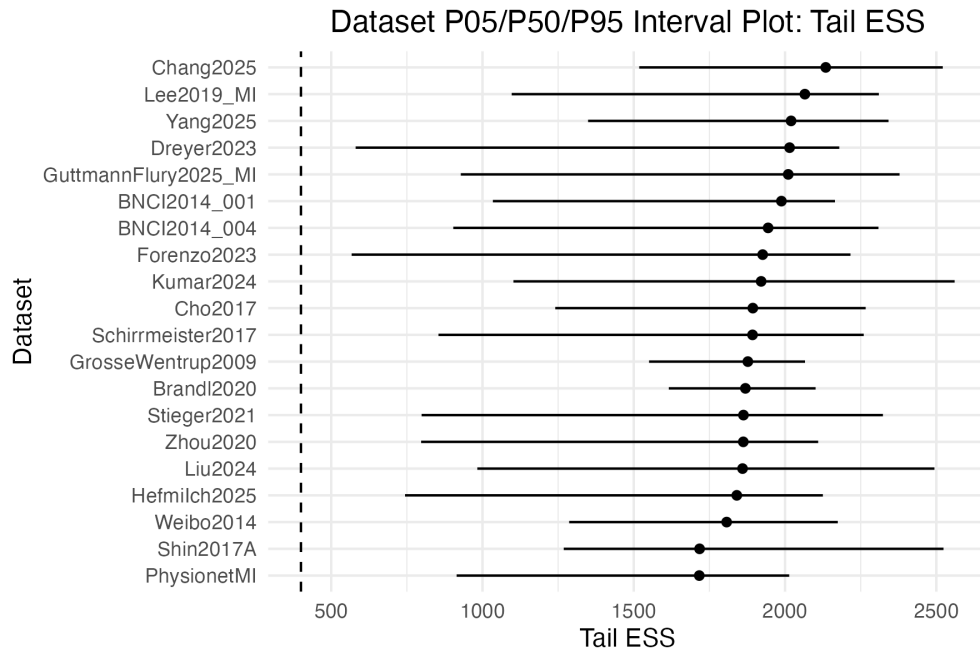


Figure 3.6: The interval plot of tail ESS across all pipelines.

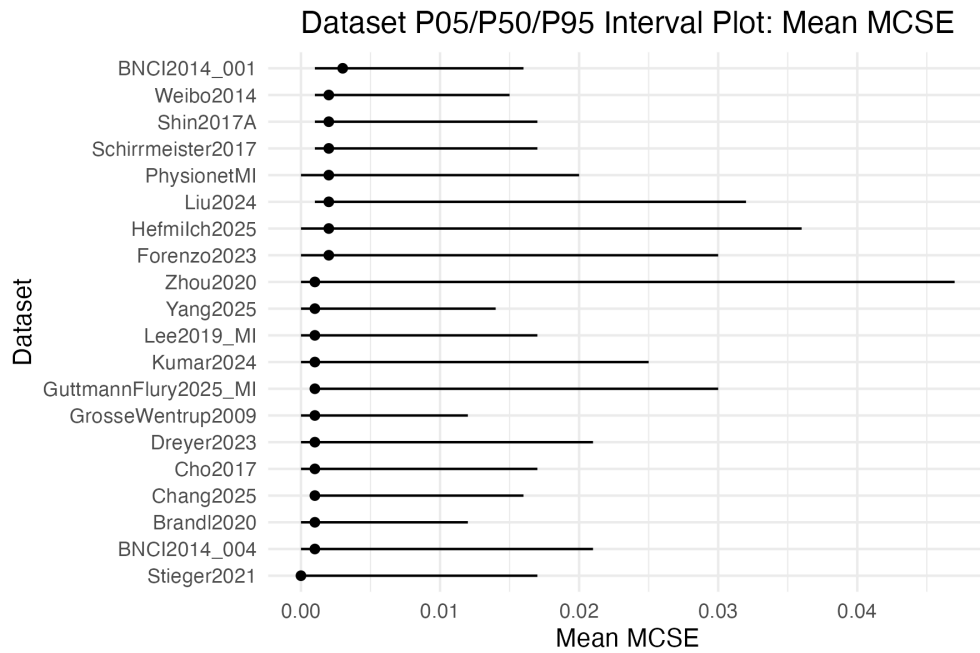


Figure 3.7: The interval plot of mean MCSE across all pipelines.

Table 3.7: The summary statistics of mean MCSE across all pipelines.

Dataset Summary Statistics: Mean MCSE					
Dataset	Min.	Max.	Median	P05	P95
BNCI2014_001	0.001	0.017	0.003	0.001	0.016
BNCI2014_004	0.000	0.023	0.001	0.000	0.021
Brandl2020	0.000	0.012	0.001	0.000	0.012
Chang2025	0.001	0.017	0.001	0.001	0.016
Cho2017	0.000	0.019	0.001	0.000	0.017
Dreyer2023	0.000	0.022	0.001	0.000	0.021
Forenzo2023	0.000	0.036	0.002	0.000	0.030
GrosseWentrup2009	0.000	0.012	0.001	0.000	0.012
GuttmannFlury2025_MI	0.001	0.035	0.001	0.001	0.030
HefmiIch2025	0.000	0.042	0.002	0.000	0.036
Kumar2024	0.000	0.029	0.001	0.000	0.025
Lee2019_MI	0.000	0.017	0.001	0.000	0.017
Liu2024	0.001	0.037	0.002	0.001	0.032
PhysionetMI	0.000	0.023	0.002	0.000	0.020
Schirrmeister2017	0.001	0.019	0.002	0.001	0.017
Shin2017A	0.001	0.017	0.002	0.001	0.017
Stieger2021	0.000	0.018	0.000	0.000	0.017
Weibo2014	0.001	0.016	0.002	0.001	0.015
Yang2025	0.000	0.014	0.001	0.000	0.014
Zhou2020	0.000	0.059	0.001	0.000	0.047

Table 3.8: The summary statistics of std. MCSE across all pipelines.

Dataset Summary Statistics: Std. MCSE					
Dataset	Min.	Max.	Median	P05	P95
BNCI2014_001	0.001	0.022	0.002	0.001	0.021
BNCI2014_004	0.000	0.027	0.001	0.000	0.026
Brandl2020	0.001	0.021	0.001	0.001	0.020
Chang2025	0.001	0.018	0.002	0.001	0.018
Cho2017	0.000	0.019	0.001	0.000	0.019
Dreyer2023	0.000	0.023	0.001	0.000	0.023
Forenzo2023	0.001	0.023	0.002	0.001	0.022
GrosseWentrup2009	0.001	0.023	0.001	0.001	0.022
GuttmannFlury2025_MI	0.001	0.029	0.002	0.001	0.027
HefmiIch2025	0.001	0.024	0.002	0.001	0.024
Kumar2024	0.000	0.023	0.001	0.000	0.023
Lee2019_MI	0.000	0.022	0.001	0.000	0.021
Liu2024	0.001	0.028	0.002	0.001	0.027
PhysionetMI	0.001	0.025	0.001	0.001	0.024
Schirrmeister2017	0.001	0.023	0.002	0.001	0.022
Shin2017A	0.001	0.021	0.002	0.001	0.020
Stieger2021	0.000	0.020	0.001	0.000	0.019
Weibo2014	0.001	0.022	0.002	0.001	0.021
Yang2025	0.000	0.021	0.001	0.000	0.021
Zhou2020	0.001	0.021	0.001	0.001	0.021

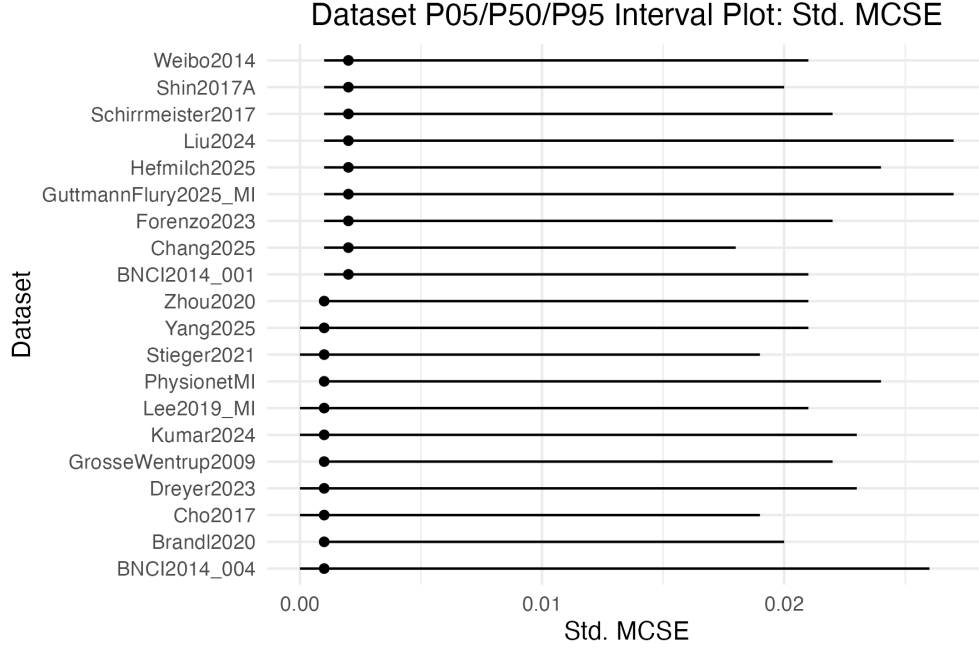


Figure 3.8: The interval plot of std. MCSE across all pipelines.

BNCI2014_001 in section F.1 are qualitatively similar to the convergence plots of all other datasets from our experiment and vice versa.

In all datasets, the 95% percentile intervals of bulk and tail ESS are highly overlapped. The same can be said about the 95% percentile intervals of the mean and standard deviation MCSE for worst case sampling diagnostics grouped by datasets. This tells us that, unlike in subsection 3.1.1 where certain pipelines clearly have better or worse sampling, no dataset was especially easier or harder to sample for all pipelines. This narrative understanding of the datasets we used is reassuring, as it suggests that all datasets were on a level playing field for our experiment.

3.2 Raw Scores

All prediction performance metrics for the dataset and pipeline combinations from our experiment are summarized in section F.2. That is, the metrics averaged across k -fold cross-

validation are shown in Table F.11 – Table F.30. The rows in these tables are ordered so that the frequentist and Bayesian model analogs within the datasets are contiguous. Subsequently, Table F.31 – Table F.40 also in section F.2 show the Bayesian scores minus the frequentist scores. In section 3.3 we describe the results of using these differences in metrics in our meta-analyses.

To gain a descriptive summary of averaged raw metrics, we grouped these scores by their type of pipeline: frequentist or Bayesian. Summary statistics of these aggregations are shown in Table 3.9. Furthermore, Figure 3.9 – Figure 3.14 visualize these aggregations from another perspective by plotting histograms for the empirical distributions of pooled frequentist and Bayesian classifier scores per metric. Additionally, Figure 3.9 – Figure 3.12 draw a dashed vertical line at the point of chance guesses corresponding to their prediction metric.

These aggregations are an informal way to pool and explore the data we generated from our experiment. We did not try to draw conclusions, but rather looked for indications that results of a formal study, correctly modeling for multiple forms of correlation across datasets, might tell us. In the histograms, we checked whether the empirical frequentist and Bayesian distributions roughly appeared to be the same, or whether they looked to be different. After all, the results of formal hypothesis testing essentially boil down to whether two (or more) distributions are different.

In section 3.3, we present the results of a multi-level meta-analysis carried out using formal instruments. The fact that the datasets were generated under heterogeneous conditions, in addition to within cluster pipeline families and correlated sampling error as described in section 2.5, explains why formal hypothesis testing was necessary to draw valid conclusions. Indeed, we completed our meta-analyses before building the following histograms. Therefore, our findings from the empirical distributions and how they related within metrics domains possibly giving hints about the results of our meta-analyses were entirely retrospective.

To build histograms, we started with the summary statistics shown in Table 3.9. Given lower and upper bounds of values, we divided the scores into 5–7 bins of uniform width. To make visualizations more useful, we made sure that boundary lines denoting chance guesses

were drawn at the edge of bins. In all cases, we found bins that included all 120 data points for building the histograms. In addition to the plotted histograms, Table 3.10 – Table 3.11 and Table 3.13 – Table 3.16 show the total of metrics that fall into each bin. The caption of these tables includes the width of the corresponding bins.

Table 3.9: The summary statistics of raw scores across all datasets.

Summary Statistics of Raw Scores						
Metric	Type of pipeline	Min.	Max.	Median	P05	P95
AUROC	Frequentist	0.471	0.85	0.69	0.539	0.81
AUROC	Bayesian	0.495	0.848	0.69	0.524	0.812
MCC	Frequentist	-0.034	0.54	0.267	0.057	0.455
MCC	Bayesian	-0.018	0.532	0.278	0.027	0.451
NLL	Frequentist	0.497	1.538	0.685	0.575	0.95
NLL	Bayesian	0.536	0.971	0.682	0.59	0.808
BS	Frequentist	0.161	0.322	0.236	0.188	0.284
BS	Bayesian	0.165	0.287	0.233	0.191	0.255
ECE	Frequentist	0.019	0.285	0.114	0.036	0.229
ECE	Bayesian	0.023	0.222	0.106	0.037	0.191
MCE	Frequentist	0.026	0.613	0.341	0.155	0.514
MCE	Bayesian	0.027	0.554	0.321	0.1	0.457

3.2.1 Discrimination Metrics

We know from the result of our meta-analyses presented in section 3.3 that Bayesian models did not offer a uniformly significant effect on discrimination metrics: AUROC and MCC.

Indeed, between Figure 3.9 and Figure 3.10, we see that the distributions look more or less the same. Without loss of generality, take, for example, the histograms of MCC. Given a score generated from one of these distributions, we would not be able to find a test statistic such that with great probability we could attribute the data point to one of these distributions. These histograms overlap too much to claim that one is especially different from the other.

Furthermore, both the AUROC and MCC plots show that the classifiers from our experiment were well-fitted to average above random chance prediction. The frequentist AUROC and Bayesian AUROC histograms show that 118/120 (98.33%) and 117/120 (97.5%) of the dataset and pipeline combinations averaged scores greater than 50%, respectively. Similarly, 118/120 (98.33%) and 116/120 (96.67%) of frequentist and Bayesian scores respectively averaged above 0.0 for MCC. Matthews correlation coefficient ranges between -1 to 1, where a negative correlation indicates that prediction would be better if binary left hand and right hand MI-EEG labels were reversed.

Table 3.10: The AUROC histogram distribution with bin width 0.1.

AUROC Histogram Distributions		
Bin	Frequentist	Bayesian
	total	total
0.4–0.5	2	3
0.5–0.6	22	21
0.6–0.7	42	42
0.7–0.8	45	45
0.8–0.9	9	9

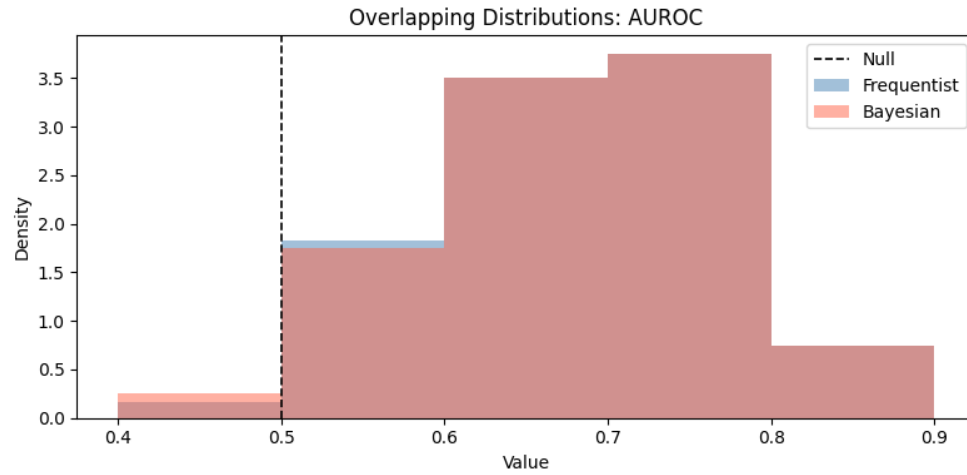


Figure 3.9: The histogram of pooled AUROC scores.

Table 3.11: The MCC histogram distribution with bin width 0.1.

MCC Histogram Distributions		
Bin	Frequentist	Bayesian
	total	total
-0.1–0.0	2	4
0.0–0.1	15	14
0.1–0.2	18	16
0.2–0.3	38	35
0.3–0.4	32	37
0.4–0.5	14	12
0.5–0.6	1	2

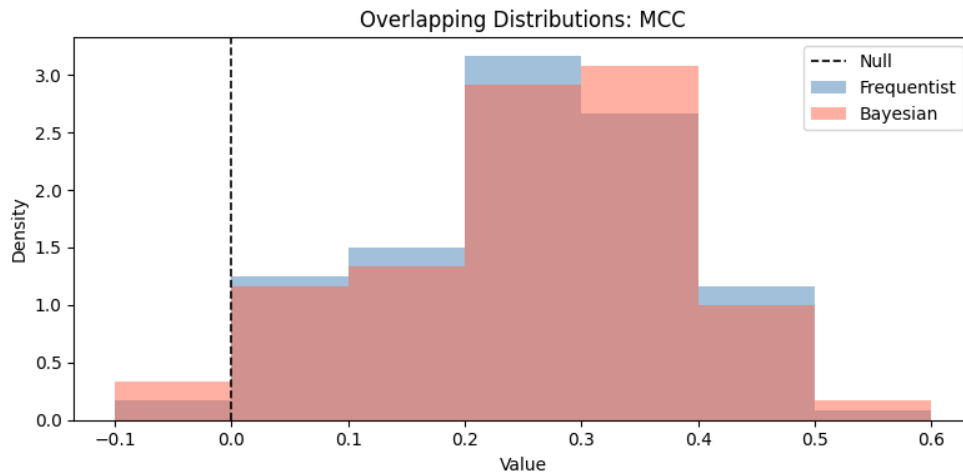


Figure 3.10: The histogram of pooled MCC scores.

3.2.2 Proper Scoring Rules

Proper scoring rules are influenced by both discrimination performance and probability calibration. However, the weighting of discrimination and calibration in metrics such as NLL and BS is unclear. Our formal meta-analysis results from section 3.3 tell us that the difference in these proper scoring rules between the Bayesian and frequentist pipelines was insignificant, similar to AUROC and MCC. However, one question, which relates to the balance of discrimination and calibration in proper scoring rules, is whether NLL and BS had different performances against their corresponding random classification thresholds.

section B.1 provides the formulas to calculate NLL and BS. For NLL, this gives us $-\ln(0.5) = 0.693$ as the upper bound threshold for chance prediction. For BS, its formula gives us $(0.5 - 1)^2 = (0.5 - 0)^2 = 0.25$. Similarly, Figure 3.11 and Figure 3.12 draw these thresholds as dashed lines. In the case of NLL, the threshold $0.693 \approx 0.7$ has been rounded so that the edges of the histogram bin start and end at the upper bound. Inspecting the totals of the bins below the AUROC and MCC thresholds, and above the NLL and BS thresholds, we find the following in Table 3.12.

Table 3.12: The total of metrics beyond random classification thresholds with corresponding percentages.

Avg. Raw Scores Beyond Thresholds			
Threshold	Frequentist total	Bayesian total	Percentage
AUROC < 0.5	2	3	1.7–2.5%
MCC < 0.0	2	4	2.5–3.3%
NLL > 0.7	39	28	23.3–32.5%
BS > 0.25	32	19	15.8–26.7%

Given the percentage of average metrics below or above the random classification thresholds of Table 3.12, we can generally infer that our pipelines were poorly calibrated. That is because pure discrimination measurements had a much lower percentage of scores beyond their threshold as opposed to proper scoring rules. At the same time, the difference in the percentage of scores above the upper bounds of NLL and BS for the Bayesian and frequentist pipelines was roughly 10%. From this we can infer that discrimination was largely unchanged, while calibration was improved with Bayesian ML. In chapter 4 we consider the explanations for why the calibration bounds were generally worse than the discrimination bounds.

3.2.3 Calibration Metrics

ECE and MCE do not have standard random performance thresholds in the same way as our discrimination metrics and proper scoring rules did. Such thresholds let us analyze the overall performance according to certain measurements. Therefore, our interest becomes the relative performance of Bayesian and frequentist pipelines in terms of pure calibration. A

Table 3.13: The NLL histogram distribution with bin width 0.2.

NLL Histogram Distributions		
Bin	Frequentist total	Bayesian total
0.3–0.5	1	0
0.5–0.7	80	92
0.7–0.9	30	26
0.9–1.1	5	2
1.1–1.3	2	0
1.3–1.5	1	0
1.5–1.7	1	0

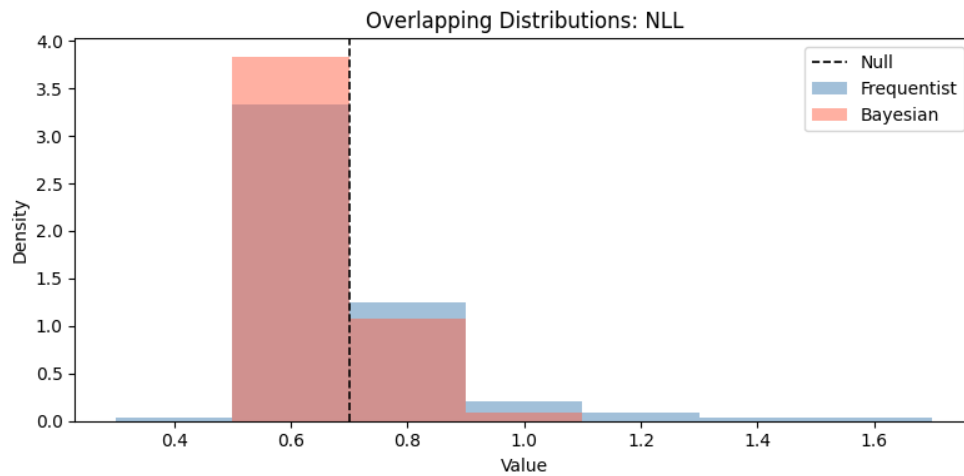


Figure 3.11: The histogram of pooled NLL scores.

Table 3.14: The Brier score histogram distribution with bin width 0.03.

Brier Score Histogram Distributions		
Bin	Frequentist	Bayesian
	total	total
0.16–0.19	7	6
0.19–0.22	24	31
0.22–0.25	57	64
0.25–0.28	25	18
0.28–0.31	5	1
0.31–0.34	2	0

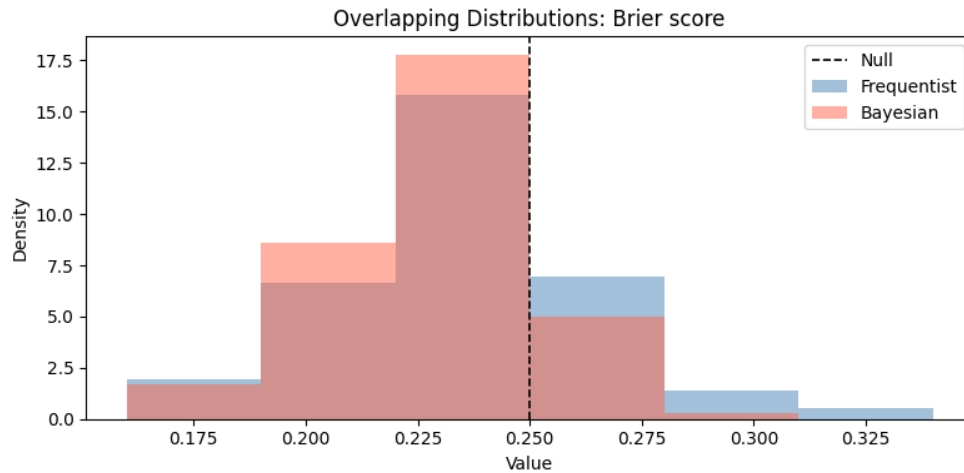


Figure 3.12: The histogram of pooled Brier scores.

question we can ask is whether the distribution of naively aggregated Bayesian scores was different from that of frequentist metrics. Indeed, from section 3.3 we know that the Bayesian classification of ECE and MCE was significantly different from the frequentist prediction.

Note that in Figure 3.13 and Figure 3.14 corresponding to the aggregated average scores of ECE and MCE, respectively, we can describe the Bayesian histograms as looking like they are shifted left of the frequentist histograms. That is, there exists a distinct bin b_k , where $1 \leq k \leq N$ and N is the number of bins, such that for all indices $1 \leq i < k$ we have all Bayesian bins greater than or equal to frequentist bins, $b_{iB} \geq b_{iF}$, and for all indices $k \leq j$ we see $b_{jB} \leq b_{jF}$. The same cannot be said for the discrimination and proper scoring rule histograms.

This visual finding, although informal, may indeed hint at Bayesian machine learning offering significantly different calibration compared to frequentist classification in our experiment. Furthermore, our finding in the previous section showing fewer Bayesian NLL and BS metrics than frequentist scores performing worse than random prediction further supports this suggestion. Given that pure discrimination measured by AUROC and MCC was largely unaffected by Bayesian ML, and measurements for calibration showed differences with Bayesian classification, we could summarize our data exploration as pointing to the formal results from our meta-analysis.

Table 3.15: The ECE histogram distribution with bin width 0.05.

ECE Histogram Distributions		
Bin	Frequentist	Bayesian
	total	total
0.00–0.05	10	13
0.05–0.10	38	43
0.10–0.15	33	37
0.15–0.20	24	24
0.20–0.25	11	3
0.25–0.30	4	0

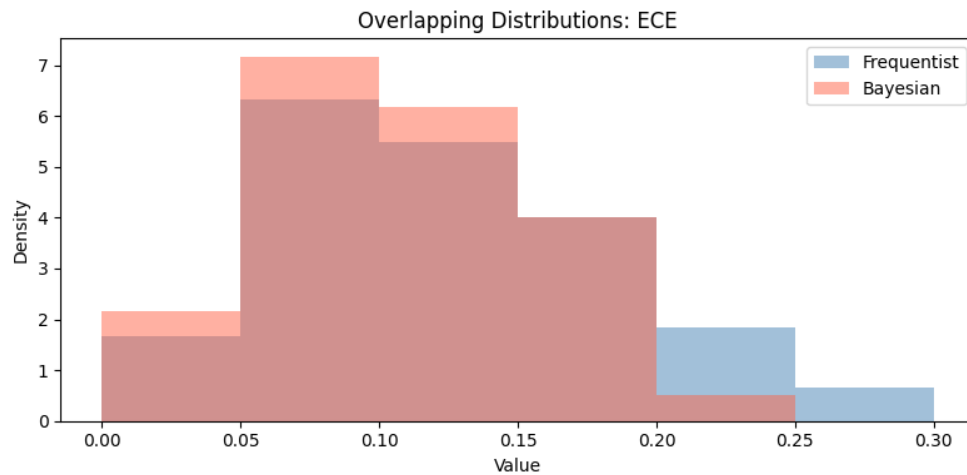


Figure 3.13: The histogram of pooled ECE scores.

Table 3.16: The MCE histogram distribution with bin width 0.1.

MCE Histogram Distributions		
Bin	Frequentist total	Bayesian total
0.0–0.1	3	6
0.1–0.2	15	16
0.2–0.3	32	32
0.3–0.4	29	42
0.4–0.5	34	22
0.5–0.6	6	2
0.6–0.7	1	0

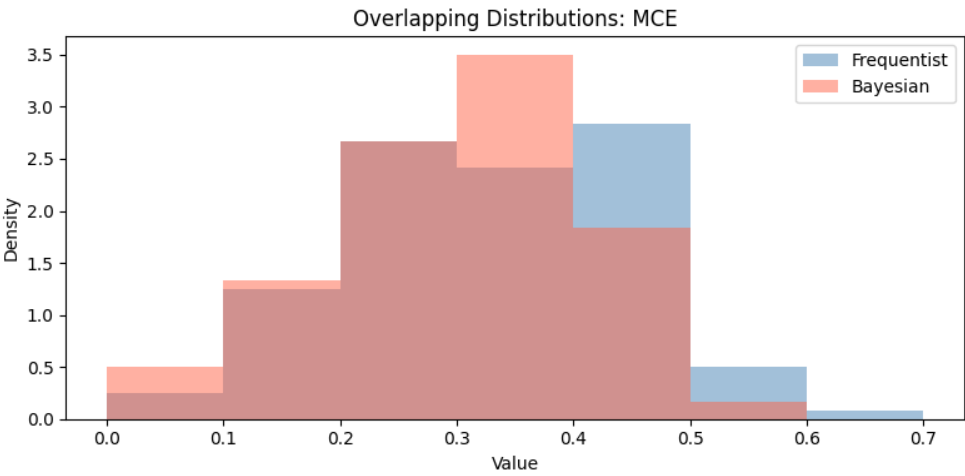


Figure 3.14: The histogram of pooled MCE scores.

3.3 Meta-Analysis

In section F.3 we have forest plots for our meta-analyses with the prediction performance metrics we measured in our experiment. Given the average performance across k -fold cross-validation shown in Table F.11 – Table F.30, and the paired differences found in Table F.31 – Table F.40, we grouped differences firstly by their dataset and secondly by their pipeline family. With these groupings, we computed the sample mean and variance, and used the results in our multi-level meta-analyses. Figure F.9 – Figure F.13 and Figure 3.15 show the intercept-only forest plots for each metric. Then, Figure F.14 – Figure F.18 and Figure 3.16 offer the meta-regression forest plots conditioned on the z -score normalized number of training samples.

3.3.1 Intercept-Only

Table 3.17 shows the results of our 3-level meta-analyses where significant effect sizes are highlighted in boldface. These models did not assume correlated and hierarchical effects for the sampling error within datasets. The Knapp-Hartung method based on a t -distribution was used instead of the Wald test with normal distribution. Assuming a significance threshold of $p = 0.05$ and a two-sided, zero-constrained test, the calibration measurements from Bayesian pipelines were significantly different compared to frequentist classifiers. the effect on AUROC with Bayesian machine learning was almost significant.

The estimate of the size of the pooled effect size was based on the average difference in the raw prediction performance score. That is, we saw an average difference of 0.28%, -0.58% and -2.72% in AUROC, ECE, and MCE, respectively. See Figure 3.15 for the forest plot of MCE. Although the average difference across left hand and right hand MI-EEG datasets and Bayesian versus frequentist pipelines was significant for calibration measurements, these effect sizes are negligible. In section 3.4, we consider these results from the perspective of the computational cost needed to train our ML models. Then chapter 4 discusses how these results may apply to future work.

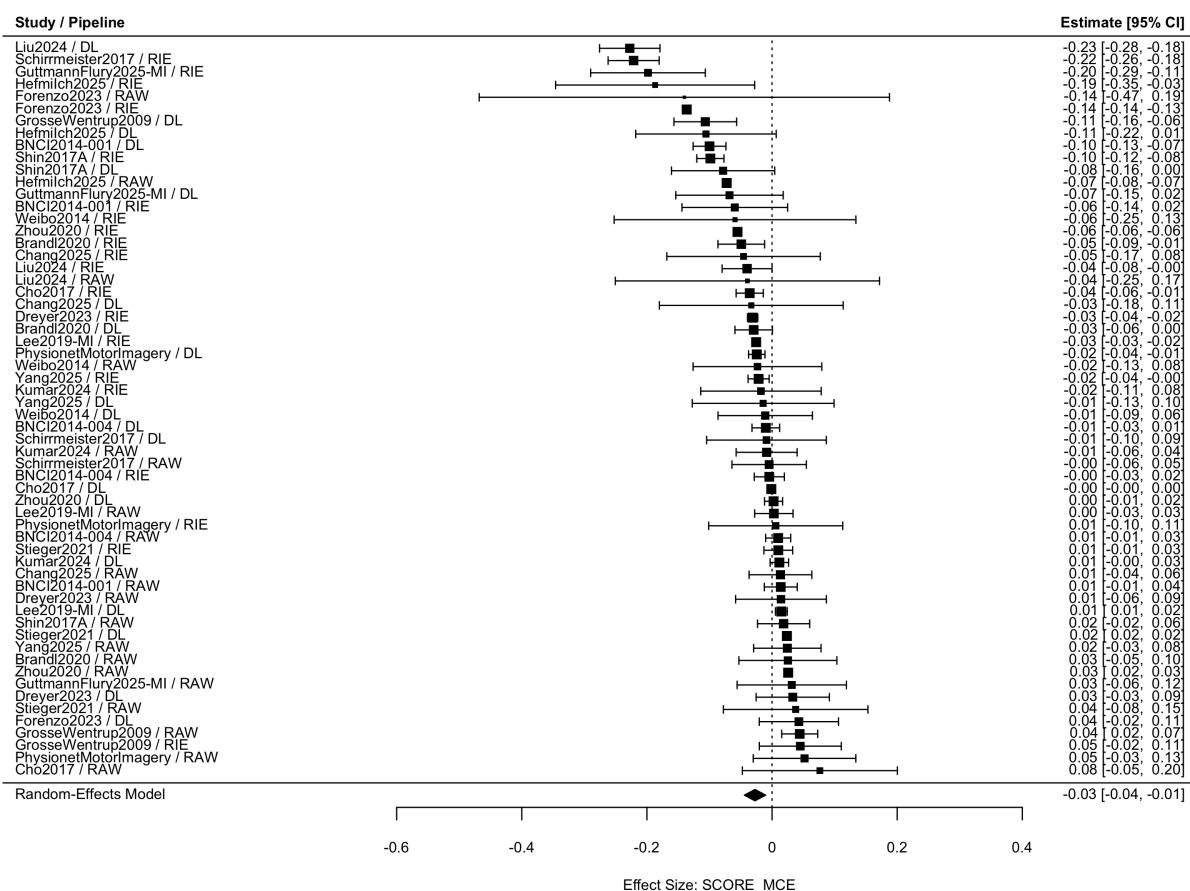


Figure 3.15: The MCE intercept-only meta-analysis forest plot.

Table 3.17: The intercept-only pooled effect size. Based on two-sided, zero-constrained t -tests with 59 degrees of freedom from 20 datasets by 3 pipeline families ($20 \times 3 = 60$). The acronyms are as follows: standard error (SE), confidence interval (CI), prediction interval (PI), lower bound (LB), and upper bound (UB).

Pooled Effect Size: Intercept-Only								
Metric	Estimate	SE	$t(59)$	p	95% CI LB	95% CI UB	95% PI LB	95% PI UB
AUROC	0.0028	0.0014	1.9616	0.0545	-0.0001	0.0056	-0.0154	0.0210
MCC	0.0035	0.0025	1.4178	0.1615	-0.0014	0.0084	-0.0222	0.0292
NLL	0.0024	0.0024	1.0159	0.3138	-0.0024	0.0073	-0.0231	0.0280
BS	-0.0007	0.0005	-1.5461	0.1274	-0.0016	0.0002	-0.0051	0.0037
ECE	-0.0058	0.0016	-3.6658	0.0005	-0.0090	-0.0027	-0.0227	0.0111
MCE	-0.0272	0.0085	-3.1992	0.0022	-0.0443	-0.0102	-0.1431	0.0886

The confidence intervals are given and represent the uncertainty around the pooled effect size. We also show prediction intervals capturing where we could expect the effect from a single new dataset to fall. Prediction intervals take into account the between study heterogeneity (τ_3^2) on top of the uncertainty in the mean. It is common for prediction intervals to be wider than confidence intervals, and was indeed what we saw in our experiment too. See section F.3 to visualize the heterogeneity of the effects across datasets through forest plots. All of Figure F.9 – Figure F.13 and Figure 3.15 show a mixture of single dataset effects that range from negative to positive Bayesian minus frequentist differences in classification performance.

Table 3.18: The intercept-only variance components. Based on Cochran’s Q test for heterogeneity with 59 degrees of freedom. Also shown are Higgins and Thompson’s I^2 statistic, and the heterogeneity variance τ^2 and standard deviation τ .

Variance Components: Intercept-Only								
Metric	$Q(59)$	p	I_3^2	I_2^2	τ_3^2	τ_3	τ_2^2	τ_2
AUROC	2848.6744	< 0.0001	0.0000	98.7439	0.0000	0.0000	0.0001	0.0090
MCC	2240.8959	< 0.0001	30.8506	68.5230	0.0000	0.0070	0.0001	0.0105
NLL	2866.4233	< 0.0001	0.0000	97.7473	0.0000	0.0000	0.0002	0.0125
BS	384.4948	< 0.0001	0.0000	80.4427	0.0000	0.0000	0.0000	0.0022
ECE	694.9828	< 0.0001	0.0000	95.8991	0.0000	0.0000	0.0001	0.0083
MCE	114816.5782	< 0.0001	0.9936	98.9607	0.0000	0.0057	0.0032	0.0570

Table 3.18 summarizes the variance components of our meta-analyses. The Cochran Q test for heterogeneity shows a significant probability that the combinations of level-3 datasets and level-2 pipeline families do not have the same effect. The sum of level-3 I_3^2 and level-2 I_2^2 statistics for the percentage of variance in observed effect sizes due to true heterogeneity

rather than sampling error supports this finding, with variance primarily driven by level-2. Sample variance τ^2 and standard deviation τ for both level-3 and level-2 is also shown, however, to counter low interpretability, Table 3.17 offers prediction intervals as described above.

When accounting for correlated and hierarchical effects within datasets using robust variance estimation, we found significant effects in calibration measurements. We also found significant effects on the differences in AUROC. The results of our intercept-only analysis with RVE are shown in Table 3.19 – Table 3.20. Sensitivity analysis with different values for our assumed correlation coefficient ρ was used to test the robustness of our results. These values spanned over $\rho \in \{0.2, 0.4, 0.6, 0.8\}$. RVE estimates the degrees of freedom across studies and therefore more closely resembled our number of datasets (20 total). Satterthwaite estimation based on a t -distribution was used to find the degrees of freedom and p -values.

The RVE can be heavily influenced by the correlation coefficient. Consistent outcomes from sensitivity analysis support rigorous results. All measurements for ECE and MCE across the iterations of ρ are relatively similar. However, although RVE tells us that the differences in AUROC are significant, we note that the p -values are monotonically decreasing over a wide range. For AUROC, this suggests that significance could be a function of correlation and that rejecting the null hypothesis risks false positive Type I error.

Our conclusion from three-level meta-analysis and robust variance estimation of the intercept is that Bayesian ML offers a significant difference in calibration for top performing left hand and right hand MI-EEG classifiers. At the same time, the metrics that account for discrimination, including the proper scoring rules, are not affected by Bayesian estimation. This summary is conservative with consideration of our RVE results for AUROC. Our assessment avoids the risk of Type I error. This is also consistent with the pattern we saw for the MCC, NLL, and Brier scores.

Table 3.19: The intercept-only pooled effect size with robust variance estimation (RVE) for correlated and hierarchical effects (CHE) models. The sensitivity analysis was conducted with a correlation coefficient $\rho \in \{0.2, 0.4, 0.6, 0.8\}$. Satterthwaite degrees of freedom (DF) and t -test estimation based on the number of clusters (20 datasets) was used.

Pooled Effect Size: Intercept-Only (RVE) I								
Metric	ρ	Estimate	SE	DF	t	p	95% CI LB	95% CI UB
AUROC	0.2	0.0029	0.0013	16.8	2.26	0.0377	0.0002	0.0057
AUROC	0.4	0.0031	0.0013	16.5	2.37	0.0303	0.0003	0.0058
AUROC	0.6	0.0032	0.0013	16.4	2.47	0.0247	0.0005	0.0060
AUROC	0.8	0.0034	0.0013	16.5	2.57	0.0201	0.0006	0.0061
MCC	0.2	0.0037	0.0025	17.5	1.51	0.1480	-0.0015	0.0089
MCC	0.4	0.0038	0.0024	17.0	1.57	0.1360	-0.0013	0.0090
MCC	0.6	0.0038	0.0024	16.5	1.58	0.1340	-0.0013	0.0090
MCC	0.8	0.0037	0.0025	16.5	1.50	0.1520	-0.0015	0.0089
NLL	0.2	0.0030	0.0019	15.0	1.58	0.1360	-0.0011	0.0072
NLL	0.4	0.0035	0.0019	15.1	1.81	0.0905	-0.0006	0.0077
NLL	0.6	0.0040	0.0020	15.5	1.96	0.0676	-0.0003	0.0082
NLL	0.8	0.0044	0.0023	16.1	1.95	0.0694	-0.0004	0.0091
BS	0.2	-0.0006	0.0004	15.6	-1.27	0.2210	-0.0015	0.0004
BS	0.4	-0.0004	0.0004	15.9	-1.03	0.3180	-0.0014	0.0005
BS	0.6	-0.0004	0.0004	16.3	-0.952	0.3550	-0.0013	0.0005
BS	0.8	-0.0005	0.0004	16.9	-1.21	0.2430	-0.0013	0.0004

Table 3.20: The intercept-only pooled effect size with robust variance estimation (RVE) continued.

Pooled Effect Size: Intercept-Only (RVE) II								
Metric	ρ	Estimate	SE	DF	t	p	95% CI LB	95% CI UB
ECE	0.2	-0.0054	0.0013	14.7	-4.30	< 0.001	-0.0081	-0.0027
ECE	0.4	-0.0052	0.0013	14.7	-4.09	0.0010	-0.0079	-0.0025
ECE	0.6	-0.0050	0.0013	15.0	-3.87	0.0015	-0.0077	-0.0022
ECE	0.8	-0.0049	0.0014	15.6	-3.50	0.0031	-0.0079	-0.0019
MCE	0.2	-0.0266	0.0084	18.2	-3.16	0.0054	-0.0443	-0.0089
MCE	0.4	-0.0262	0.0084	18.0	-3.12	0.0059	-0.0439	-0.0086
MCE	0.6	-0.0259	0.0084	17.8	-3.09	0.0064	-0.0435	-0.0082
MCE	0.8	-0.0256	0.0084	17.7	-3.05	0.0070	-0.0432	-0.0079

3.3.2 *Meta-Regression*

As a moderator analysis, we wanted to know whether the training sample size had an effect on the prediction performance. Using other dimensions such as training duration and carbon footprint is dependent on additional factors such as the underlying algorithm and silicon implemented at runtime. In section 3.4 we assess the results of the computational cost of fitting our classifiers. This allows us to consider both the benefits and costs of Bayesian ML for BCI research. However, the comparison is at best descriptive and must be interpreted alongside the specific software and hardware used in our experiment.

Frequentist ML systematically overfits when trained on small sample sizes. However, Bayesian ML avoids such overfitting altogether. Conceptually, this is because the Bayesian approach averages over the entire parameter space and therefore avoids overweighting highly sharp minima [9]. Therefore, Bayesian ML is known to be beneficial in small data scenarios [10]. Medical studies that include EEG datasets often suffer from small sample sizes. Indeed, this is how some BCI researchers interpret the relatively low performance of deep learning in EEG classification, as DL benefits from big data [21].

These past findings motivated our meta-regression on training sample size. However, we also knew that while cross subject evaluation maximizes the difficulty of pattern recognition between heterogeneous MI-EEG signals, it also maximizes the number of training samples, as described in chapter 2. Recall Table 2.13 showing the average number of training samples across k -fold cross-validation for each dataset. Table 3.21 gives summary statistics for these averages of training sample sizes.

Although our smallest datasets belonged to a small data regime, our largest datasets did not in terms of a majority of our classifiers: LDA, logistic regression, SVM, GP. Additionally, the interquartile range shows that our average number of training samples is right skewed towards larger datasets. This tells us that our experiment may not be appropriate to measure the improvement of prediction performance with respect to the training sample size. With that in mind, Table 3.22 presents the results of our meta-regression on sample size.

Table 3.21: The summary statistics for the average number of training samples from Table 2.13.

Summary Statistics:						
Avg. No. Training Samples						
Min.	Max.	Q1	Median	Q3	Mean	Std. Dev.
1264	39079.8	2016	4910	6446.4	7570.83	9374.86

Similarly to what we found in our intercept-only meta-analysis, we found that the effect size on the intercept estimates for ECE and MCE were significant. See Figure 3.16 for the forest plot of MCE from our meta-regression. We also saw that the effect on AUROC was significant. However, we were cautious to reject the null hypothesis for AUROC. As discussed with the intercept-only models with RVE, this result could be a false positive. This conservative assessment is supported by the findings of the other metrics that measure discrimination as well: MCC, NLL, and Brier score.

Our meta-regression was structured so that we evaluated the slope given a step size that was one standard deviation of the average number of training samples across all datasets. That is, the slope of our meta-regression considered the change in the effect when more than 9,000 trials were added to the training set. This number of trials is seen as the standard deviation in Table 3.21. This approach based on the standard deviation of average sample size was used because it is more interpretable than the effect of adding a trial to a study. Indeed, adding over 9000 trials to a study could be roughly thought of as adding a handful of subjects.

Although Table 3.22 shows that the sample size had a significant effect on MCE, it was more interesting to see that the majority metrics were unaffected by the training sample size. This finding gives us additional evidence that discrimination metrics and proper scoring rules

Table 3.22: The pooled effect size for meta-regression on the z -score normalized number of training samples. Based on the t -test with 58 degrees of freedom. Separate rows for the estimated intercept (β_0) and slope (β_1). Prediction intervals for the slope were not reported (NR).

Pooled Effect Size: Moderator									
Metric	Model	Estimate	SE	$t(58)$	p	95% CI LB	95% CI UB	95% PI LB	95% PI UB
AUROC	β_0	0.0029	0.0014	2.0949	0.0406	0.0001	0.0057	-0.0149	0.0207
AUROC	β_1	-0.0023	0.0014	-1.5664	0.1227	-0.0051	0.0006	NR	NR
MCC	β_0	0.0037	0.0024	1.5460	0.1275	-0.0011	0.0085	-0.0215	0.0289
MCC	β_1	-0.0030	0.0024	-1.2895	0.2024	-0.0078	0.0017	NR	NR
NLL	β_0	0.0017	0.0025	0.6815	0.4983	-0.0033	0.0067	-0.0241	0.0275
NLL	β_1	0.0024	0.0020	1.1546	0.2530	-0.0017	0.0064	NR	NR
BS	β_0	-0.0008	0.0005	-1.7524	0.0850	-0.0018	0.0001	-0.0055	0.0038
BS	β_1	0.0006	0.0004	1.2949	0.2005	-0.0003	0.0014	NR	NR
ECE	β_0	-0.0065	0.0017	-3.8480	0.0003	-0.0098	-0.0031	-0.0239	0.0110
ECE	β_1	0.0018	0.0014	1.3316	0.1882	-0.0009	0.0046	NR	NR
MCE	β_0	-0.0280	0.0082	-3.4297	0.0011	-0.0444	-0.0117	-0.1392	0.0831
MCE	β_1	0.0189	0.0082	2.3215	0.0238	0.0026	0.0353	NR	NR

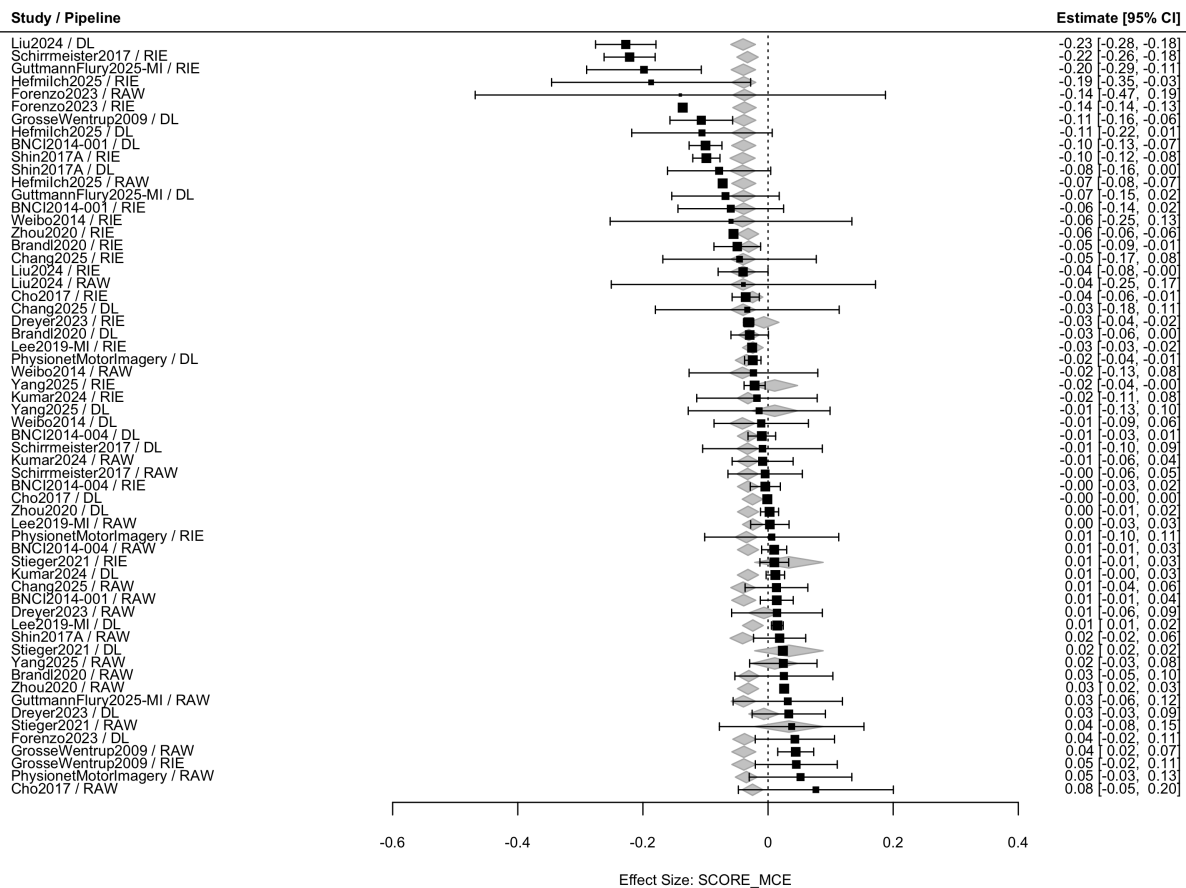


Figure 3.16: The MCE meta-regression forest plot.

are generally not effected by Bayesian ML for left hand and right hand MI-EEG classification. Furthermore, even for calibration measurements based on sample size, the effect size is still negligible. The average difference between ECE and MCE was -0.65% (95%CI: [-0.98%, -0.31%]) and -2.8% (95%CI: [-4.44%, -1.17%]), respectively.

Table 3.23: The variance components for meta-regression on the z -score normalized number of training samples. Based on Cochran’s Q test for residual heterogeneity with 58 degrees of freedom, and the Wald F -test for the moderator.

Variance Components: Moderator				
Metric	$Q_E(58)$	p_{Q_E}	$F(1, 58)$	p_F
AUROC	2092.0300	< 0.0001	2.4536	0.1227
MCC	2031.1478	< 0.0001	1.6627	0.2024
NLL	2866.2647	< 0.0001	1.3330	0.2530
BS	383.8096	< 0.0001	1.6768	0.2005
ECE	674.8489	< 0.0001	1.7731	0.1882
MCE	6609.2552	< 0.0001	5.3892	0.0238

Table 3.23 shows the results of the Cochran’s Q test for residual heterogeneity with 58 degrees of freedom. We see that the heterogeneity across the data set and pipeline family combinations was significant in all cases. Figure F.14 – Figure F.18 in section F.3 and Figure 3.16 visualize the heterogeneity from the intercept and slope estimates. The intercept models are shown in black, and the slope estimates are shown in gray. The prediction intervals in Table 3.22 also give us a perspective of the effect size that we could expect from a single new study being added to our experiment. The range of these PIs further supports that our estimate for the intercept was due to real heterogeneity in our data rather than sampling error alone.

Table 3.24: The pooled effect size with robust variance estimation for meta-regression on the z -score normalize number of training samples. Based on the t -test with 58 degrees of freedom.

Pooled Effect Size: Moderator (RVE) I								
Metric	ρ	Model	Estimate	SE	$t(58)$	p	95% CI LB	95% CI UB
AUROC	0.2	β_0	0.0031	0.0012	2.62	0.0188	0.0006	0.0056
AUROC	0.2	β_1	-0.0023	0.0009	-2.68	0.0945	-0.0055	0.0008
AUROC	0.4	β_0	0.0032	0.0012	2.76	0.0142	0.0007	0.0057
AUROC	0.4	β_1	-0.0023	0.0009	-2.61	0.0988	-0.0056	0.0009
AUROC	0.6	β_0	0.0034	0.0012	2.88	0.0111	0.0009	0.0059
AUROC	0.6	β_1	-0.0024	0.0009	-2.55	0.1033	-0.0058	0.0010
AUROC	0.8	β_0	0.0035	0.0012	2.99	0.0088	0.0010	0.0060
AUROC	0.8	β_1	-0.0024	0.0010	-2.52	0.1061	-0.0060	0.0011
MCC	0.2	β_0	0.0040	0.0024	1.68	0.1120	-0.0010	0.0090
MCC	0.2	β_1	-0.0032	0.0014	-2.29	0.1330	-0.0084	0.0021
MCC	0.4	β_0	0.0041	0.0023	1.76	0.0984	-0.0009	0.0091
MCC	0.4	β_1	-0.0032	0.0014	-2.29	0.1333	-0.0086	0.0022
MCC	0.6	β_0	0.0041	0.0023	1.75	0.1000	-0.0009	0.0091
MCC	0.6	β_1	-0.0032	0.0014	-2.24	0.1400	-0.0086	0.0023
MCC	0.8	β_0	0.0039	0.0024	1.64	0.1220	-0.0012	0.0089
MCC	0.8	β_1	-0.0030	0.0014	-2.13	0.1520	-0.0085	0.0025

Table 3.25: The pooled effect size with robust variance estimation for meta-regression continued.

Pooled Effect Size: Moderator (RVE) II								
Metric	ρ	Model	Estimate	SE	$t(58)$	p	95% CI LB	95% CI UB
NLL	0.2	β_0	0.0024	0.0018	1.35	0.1970	-0.0014	0.0062
NLL	0.2	β_1	0.0020	0.0014	1.40	0.2800	-0.0034	0.0074
NLL	0.4	β_0	0.0030	0.0018	1.61	0.1280	-0.0010	0.0069
NLL	0.4	β_1	0.0017	0.0015	1.13	0.3630	-0.0040	0.0073
NLL	0.6	β_0	0.0035	0.0020	1.76	0.0990	-0.0007	0.0077
NLL	0.6	β_1	0.0014	0.0015	0.92	0.4420	-0.0045	0.0073
NLL	0.8	β_0	0.0039	0.0024	1.68	0.1130	-0.0010	0.0089
NLL	0.8	β_1	0.0014	0.0016	0.88	0.4620	-0.0046	0.0074
BS	0.2	β_0	-0.0007	0.0004	-1.51	0.1530	-0.0016	0.0003
BS	0.2	β_1	0.0005	0.0003	1.30	0.3080	-0.0009	0.0018
BS	0.4	β_0	-0.0005	0.0004	-1.25	0.2320	-0.0015	0.0004
BS	0.4	β_1	0.0004	0.0004	1.07	0.3830	-0.0010	0.0018
BS	0.6	β_0	-0.0005	0.0004	-1.21	0.2430	-0.0014	0.0004
BS	0.6	β_1	0.0004	0.0004	1.12	0.3670	-0.0010	0.0019
BS	0.8	β_0	-0.0006	0.0004	-1.58	0.1320	-0.0014	0.0002
BS	0.8	β_1	0.0006	0.0004	1.51	0.2560	-0.0009	0.0020

Table 3.26: The pooled effect size with robust variance estimation for meta-regression continued.

Pooled Effect Size: Moderator (RVE) III								
Metric	ρ	Model	Estimate	SE	$t(58)$	p	95% CI LB	95% CI UB
ECE	0.2	β_0	-0.0060	0.0012	-5.01	< 0.001	-0.0086	-0.0034
ECE	0.2	β_1	0.0016	0.0011	1.49	0.2710	-0.0028	0.0060
ECE	0.4	β_0	-0.0057	0.0012	-4.66	< 0.001	-0.0083	-0.0031
ECE	0.4	β_1	0.0014	0.0012	1.19	0.3520	-0.0035	0.0062
ECE	0.6	β_0	-0.0054	0.0013	-4.27	< 0.001	-0.0081	-0.0027
ECE	0.6	β_1	0.0012	0.0012	1.00	0.4200	-0.0039	0.0064
ECE	0.8	β_0	-0.0053	0.0014	-3.69	0.0021	-0.0083	-0.0022
ECE	0.8	β_1	0.0012	0.0013	0.92	0.4537	-0.0041	0.0065
MCE	0.2	β_0	-0.0277	0.0076	-3.66	0.0019	-0.0437	-0.0118
MCE	0.2	β_1	0.0187	0.0052	3.61	0.0614	-0.0021	0.0395
MCE	0.4	β_0	-0.0275	0.0076	-3.63	0.0021	-0.0434	-0.0115
MCE	0.4	β_1	0.0185	0.0052	3.60	0.0622	-0.0022	0.0393
MCE	0.6	β_0	-0.0272	0.0076	-3.59	0.0023	-0.0432	-0.0112
MCE	0.6	β_1	0.0184	0.0051	3.59	0.0631	-0.0024	0.0391
MCE	0.8	β_0	-0.0270	0.0076	-3.55	0.0025	-0.0431	-0.0109
MCE	0.8	β_1	0.0182	0.0051	3.57	0.0640	-0.0025	0.0390

We also applied RVE to our meta-regression. See Table 3.24 – Table 3.26. We used the same correlation coefficient $\rho \in \{0.2, 0.4, 0.6, 0.8\}$ for the sensitivity analysis as in the previous section. Furthermore, our conclusions drawn from meta-regression with RVE were similar to our assessments for intercept-only meta-analysis with RVE. The effect on ECE and MCE on the moderator analysis on the training sample size was significant for the intercept. However, for AUROC we were cautious in accepting the significance of the effect. Here, we can again note a monotonically decreasing p -value over a broad range, making us reluctant to reject the null hypothesis.

Given that we had a relatively small number of datasets to perform RVE with a reliable estimate of the slope, we also applied cluster wild bootstrapping (CWB). See Table 3.27 – Table 3.28. Indeed, the results from the CWB change the significance of the result for the slopes of the AUROC and MCE. This suggests that our model of the effect on these metrics improves when we account for the training sample size. However, we note that for the majority metrics, the training sample size does not affect the pooled effect size of the prediction performance.

Our conclusions for the pooled effect sizes of meta-regression remain similar to those of our intercept-only meta-analysis. Bayesian ML improves calibration, while also not affecting discrimination. However, the effect size is negligible for static machine learning models. Considering that Bayesian ML requires integration over the parameter space and this is a computationally expensive operation, our meta-analyses suggest that frequentist estimators could be sufficient for MI-EEG classification. In the next section, we will look at the computational tradeoffs of our experiment more closely.

Table 3.27: The significance of the pooled effect size for the slope of meta-regression. Computed with cluster wild bootstrapping (CWB) and compared with RVE.

Pooled Effect Size: Moderator (CWB) I			
Metric	ρ	p_{RVE}	p_{CWB}
AUROC	0.2	0.0945	0.0325
AUROC	0.4	0.0988	0.0325
AUROC	0.6	0.1033	0.0330
AUROC	0.8	0.1061	0.0340
MCC	0.2	0.1330	0.1030
MCC	0.4	0.1333	0.1005
MCC	0.6	0.1400	0.1050
MCC	0.8	0.1520	0.1150
NLL	0.2	0.2800	0.4010
NLL	0.4	0.3630	0.4595
NLL	0.6	0.4420	0.5145
NLL	0.8	0.4620	0.4930
BS	0.2	0.3080	0.4060
BS	0.4	0.3830	0.4705
BS	0.6	0.3670	0.4835
BS	0.8	0.2560	0.3675

Table 3.28: The significance of the pooled effect size for the slope of meta-regression continued.

Pooled Effect Size: Moderator (CWB) II			
Metric	ρ	p_{RVE}	p_{CWB}
ECE	0.2	0.2710	0.5405
ECE	0.4	0.3520	0.6385
ECE	0.6	0.4200	0.6850
ECE	0.8	0.4537	0.6750
MCE	0.2	0.0614	0.0315
MCE	0.4	0.0622	0.0310
MCE	0.6	0.0631	0.0325
MCE	0.8	0.0640	0.0325

3.4 Computational Costs

In section F.4 we present Table F.41 – Table F.60 summarizing the duration in seconds and grams of CO₂ equivalent measured by fitting our pipelines on datasets. For each combination of dataset and pipeline, the measurements were averaged over a k -fold cross-validation. Subsequently, Table F.61 – Table F.80 show fine grained metrics of power and energy usage measured in watts and watt-hours, respectively, averaged across all folds of cross-validation. In these tables, CodeCarbon estimated a constant 70.0 W of RAM power for all rows. For all tables Table F.41 – Table F.80, the results of each cross-validation fold before averaged aggregation can be found in our Figshare [25].

3.4.1 Emissions Summary

The emissions measured in g CO₂eq is a direct conversion from the energy consumption in Wh multiplied by the intensity of the regional grid. As noted in section 2.4, the region of the server used for our experiment was WA, USA. CodeCarbon uses 2016 grid intensity reports to track emissions per state in the USA. For WA, the equivalent lbs of CO₂ equivalent per MWh is 186.844, one of the lowest measurements for all states at that time. That is, in the same year the USA was reported to have consumed 998.443 lbs CO₂eq/MWh. Additionally, the maximum intensity of the regional grid for any state was Wyoming (WY) with 2026.26 lbs of CO₂ eq/MWh.

Table 3.29 – Table 3.31 show the median, IQR, and outlier bounds for the average duration (s), emissions (g CO₂eq), and energy consumed (Wh) by our pipelines. Figure 3.17 – Figure 3.19 plot these tables, respectively. For context with the emissions of training our pipelines, consider that as part of its emissions index CodeCarbon uses reports from The French Agency of Ecological Transition that say the cost of charging a tablet or smartphone once per day is 19.178 Wh/day [51]. 19.178 Wh is close to the average energy consumed by our pairs of pipelines and datasets. As seen in Table 3.31, the Q3 energy consumed by frequentist pipelines falls below 19.178 Wh, and Q1 for most Bayesian pipelines falls above

this marker.

To gauge the cost of charging a smartphone, consider additional examples of common household appliances and their corresponding energy consumption. A combination refrigerator freezer with 24/7 usage on average consumes 863.01 Wh/day, and running a washing machine and tumble dryer together has an average combined energy consumption of 1338.38 Wh/cycle [51]. These findings tell us that the global impact of fitting our ML models is low compared to other everyday appliances. However, we keep in mind that between the averages of the medians for frequentist and Bayesian pipelines, there is a $5.78\times$ and $7.19\times$ growth in the duration and energy consumed, respectively, when training our Bayesian pipelines.

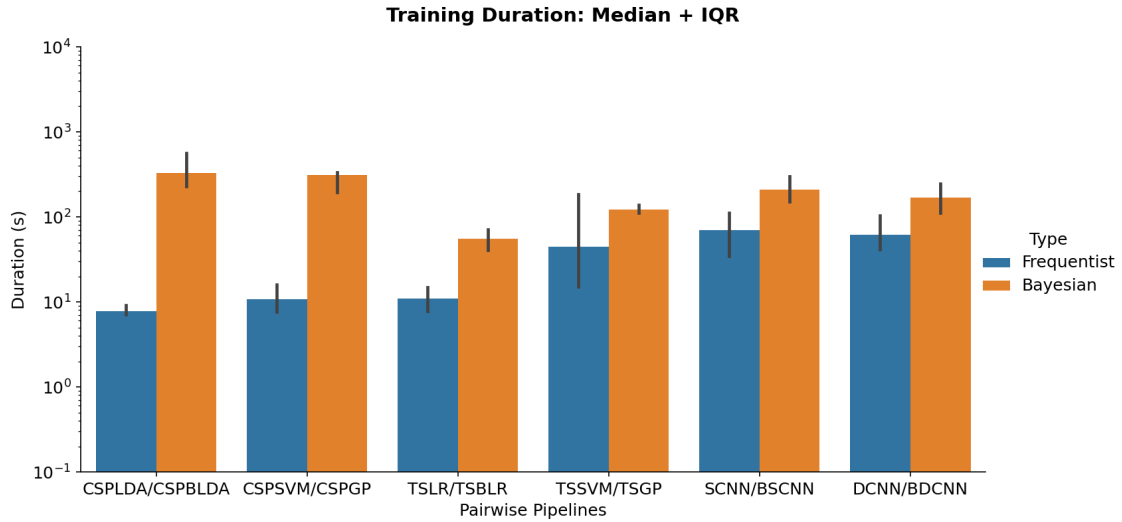


Figure 3.17: The median and IQR of average training duration plotted on logarithmic scale.

For another perspective, consider additional references used by CodeCarbon from The International Energy Agency (IEA) that say that the average US citizen emits 13.3 tons of CO₂ equivalent per year [44]. This converts to 0.384 g CO₂eq/s. Using the regional grid intensities for WA, USA, and WY mentioned above, we can convert the units from lbs CO₂eq/MWh to g CO₂eq/Wh, and calculate the emission rate of our study as if it were performed in these corresponding regions.

Table 3.29: The median, IQR, and outlier duration bounds were averaged over k -fold cross-validation and grouped by pipeline. The average of medians from training the frequentist pipelines is 34.37 s, while for Bayesian classifiers it is 198.55 s (3 min 18.55 sec). Furthermore, the average Q1 and Q3 for frequentist pipelines is 18.93 s and 73.21 s (1 min 13.21 sec), respectively, and for Bayesian estimators it is 139.45 s (2 min 19.45 sec) and 275.65 s (4 min and 35.65 sec).

Summary Statistics: Avg. Duration (s)								
Pipeline	Median	Q1	Q3	IQR	Lower fence	Upper fence	No. low outliers	No. high outliers
CSPLDA	7.753	7.064	9.129	2.065	3.967	12.227	0	3
CSPBLDA	329.301	228.296	563.299	335.003	-274.208	1065.804	0	3
CSPSVM	10.779	7.647	16.034	8.387	-4.932	28.614	0	3
CSPGP	308.212	193.5	334.453	140.954	-17.931	545.884	0	2
TSLR	10.945	7.807	14.939	7.133	-2.892	25.638	0	2
TSBLR	55.223	40.932	71.469	30.537	-4.874	117.274	0	2
TSSVM	44.645	15.037	183.5	168.463	-237.657	436.194	0	3
TSGP	121.267	112.267	137.281	25.014	74.746	174.803	0	4
SCNN	70.052	34.614	111.64	77.026	-80.926	227.179	0	3
BSCNN	208.774	150.83	300.183	149.353	-73.199	524.213	0	3
DCNN	62.06	41.389	104.022	62.633	-52.561	197.971	0	3
BDCNN	168.541	110.889	247.232	136.343	-93.625	451.746	0	4

Table 3.30: The median, IQR, and outlier limits of emissions average over k -fold cross-validation and are grouped by pipeline.

Summary Statistics: Avg. Emissions (g CO ₂ eq)								
Pipeline	Median	Q1	Q3	IQR	Lower fence	Upper fence	No. low outliers	No. high outliers
CSPLDA	0.065	0.059	0.077	0.018	0.033	0.103	0	3
CSPBLDA	3.151	2.125	4.894	2.768	-2.027	9.046	0	3
CSPSVM	0.09	0.064	0.133	0.069	-0.039	0.237	0	3
CSPGP	4.871	2.911	5.331	2.42	-0.719	8.961	0	2
TSLR	0.102	0.065	0.136	0.071	-0.04	0.242	0	1
TSBLR	0.794	0.539	1.089	0.55	-0.287	1.915	0	2
TSSVM	0.369	0.125	1.515	1.39	-1.961	3.601	0	3
TSGP	1.79	1.621	1.983	0.362	1.078	2.525	0	4
SCNN	0.912	0.43	1.604	1.174	-1.331	3.364	0	2
BSCNN	3.557	2.126	5.294	3.168	-2.625	10.046	0	3
DCNN	0.773	0.492	1.292	0.8	-0.707	2.491	0	3
BDCNN	2.44	1.481	3.989	2.508	-2.28	7.751	0	4

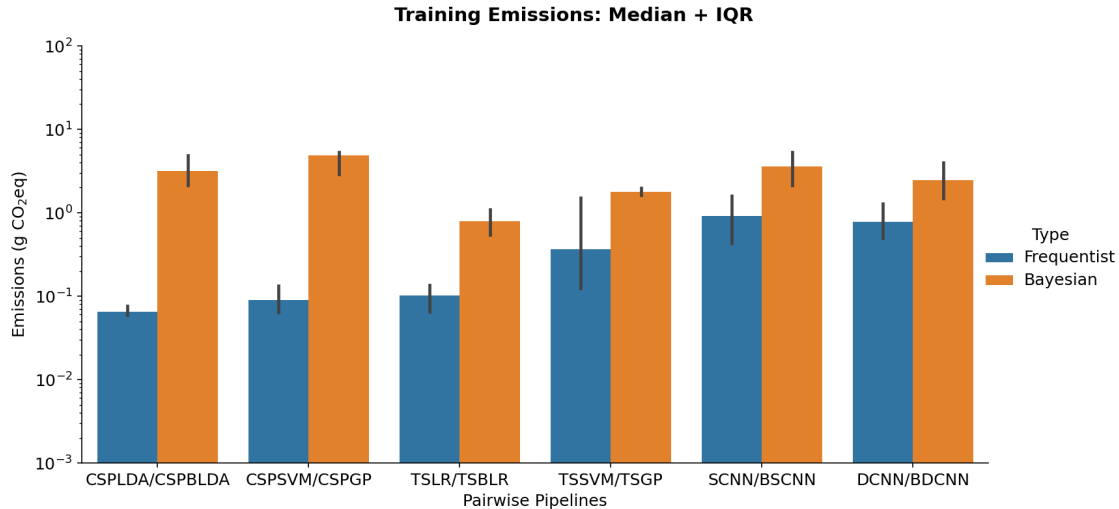


Figure 3.18: The median and IQR of average training emissions plotted on a logarithmic scale.

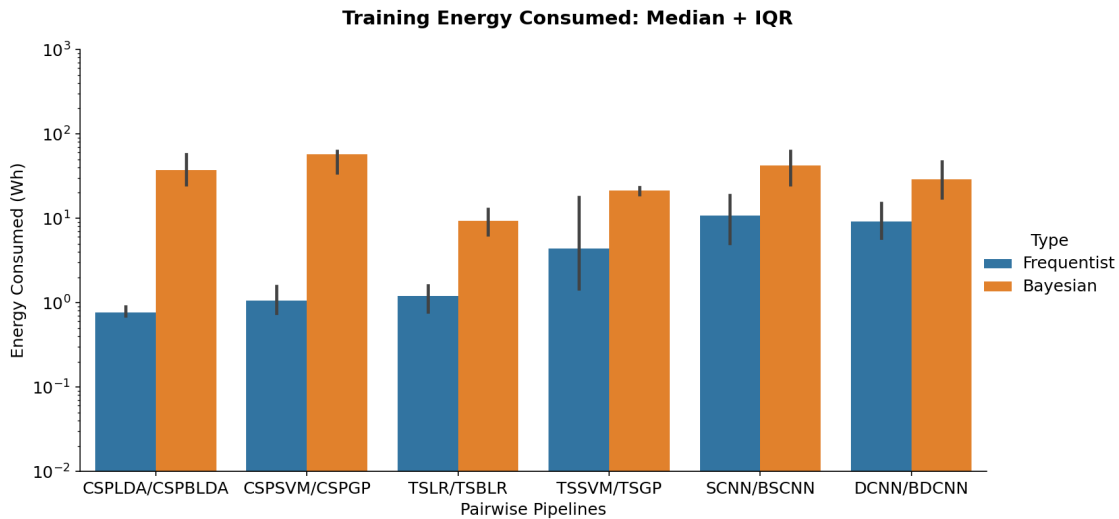


Figure 3.19: The median and IQR of average training energy consumed plotted on a logarithmic scale.

Table 3.31: The median, IQR, and outlier bounds of the energy consumed average over k -fold cross-validation and are grouped by pipeline.

Summary Statistics: Avg. Energy Consumed (Wh)								
Pipeline	Median	Q1	Q3	IQR	Lower fence	Upper fence	No. low outliers	No. high outliers
CSPLDA	0.767	0.696	0.903	0.207	0.385	1.214	0	3
CSPBLDA	37.176	25.076	57.74	32.665	-23.921	106.737	0	3
CSPSVM	1.061	0.757	1.571	0.814	-0.464	2.792	0	3
CSPGP	57.479	34.349	62.902	28.554	-8.482	105.733	0	2
TSLR	1.198	0.772	1.605	0.833	-0.478	2.854	0	1
TSBLR	9.37	6.356	12.852	6.495	-3.387	22.595	0	2
TSSVM	4.351	1.472	17.877	16.405	-23.136	42.484	0	3
TSGP	21.124	19.125	23.392	4.268	12.723	29.794	0	4
SCNN	10.767	5.07	18.921	13.851	-15.706	39.697	0	2
BSCNN	41.968	25.088	62.465	37.378	-30.979	118.531	0	3
DCNN	9.118	5.809	15.242	9.434	-8.341	29.393	0	3
BDCNN	28.789	17.48	47.07	29.59	-26.906	91.456	0	4

Table 3.32 offers the quartiles of the emission rates for WA, USA and WY. Given the average of all medians for WA, USA, and WY, there is $5.37\times$ growth from WA to USA, and $10.93\times$ growth from WA to WY. We find that if our evaluation were conducted in Wyoming, then the median of 2/6 of frequentist pipelines would be above 1/3 (8 hours/day) of the average US citizen emissions rate (0.128 g CO₂eq/s) and 5/6 of Bayesian pipeline medians would be above this point.

Table 3.32: The Q1/Q2/Q3 quartiles of the emission rate in g CO₂eq/s when the energy consumed by our experiment is converted to the estimates of grid intensity for USA and WY.

Comparison: Avg. Emissions Rate (g CO ₂ eq/s)									
Pipeline	Median (WA)	Median (USA)	Median (WY)	Q1 (WA)	Q1 (USA)	Q1 (WY)	Q3 (WA)	Q3 (USA)	Q3 (WY)
CSPLDA	0.008	0.043	0.087	0.008	0.043	0.087	0.008	0.043	0.087
CSPBLDA	0.009	0.048	0.098	0.009	0.048	0.098	0.010	0.053	0.108
CSPSVM	0.008	0.043	0.087	0.008	0.043	0.087	0.008	0.043	0.087
CSPGP	0.016	0.085	0.174	0.015	0.080	0.163	0.016	0.085	0.174
TSLR	0.008	0.045	0.092	0.008	0.043	0.087	0.009	0.048	0.098
TSBLR	0.014	0.077	0.157	0.013	0.069	0.141	0.015	0.080	0.163
TSSVM	0.008	0.043	0.087	0.008	0.043	0.087	0.008	0.043	0.087
TSGP	0.015	0.080	0.163	0.014	0.075	0.152	0.015	0.080	0.163
SCNN	0.013	0.069	0.141	0.012	0.064	0.130	0.014	0.075	0.152
BSCNN	0.017	0.091	0.184	0.015	0.080	0.163	0.018	0.096	0.195
DCNN	0.012	0.064	0.130	0.012	0.064	0.130	0.013	0.069	0.141
BDCNN	0.014	0.075	0.152	0.013	0.069	0.141	0.016	0.085	0.174

3.4.2 Detailed Profiling

Table 3.33 – Table 3.34 show the breakdown of CPU and GPU power usage in watts from our experiment. As mentioned above, CodeCarbon estimated a fixed 70.0 W of RAM power for every fold of cross-validation. Figure 3.20 – Figure 3.21 plot these tables with logarithmic scale. We see that the rate of energy consumption per unit time by CPUs was higher for our frequentist pipelines and for GPUs was higher for the Bayesian classifiers. This is understandable since on average our frequentist pipelines took less time to fit, and most were CPU based computation. Higher training duration and GPU bound execution also explains why fitting our Bayesian estimators had higher rates of GPU energy consumption.

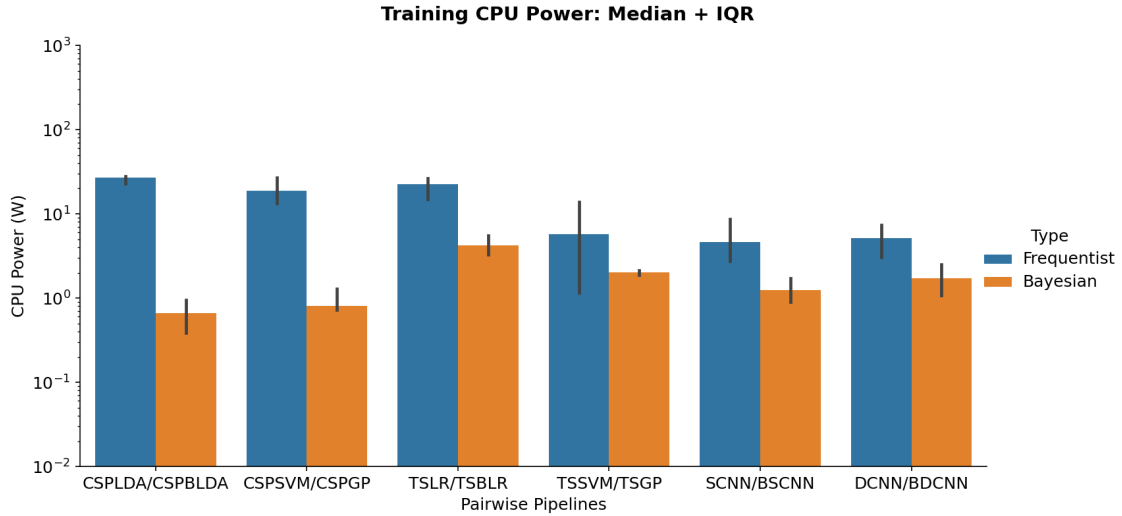


Figure 3.20: The median and IQR of the average CPU power from training plotted on logarithmic scale.

In terms of total energy consumed in watt-hours, the resources used to train our Bayesian classifiers dominated CPU, GPU, and RAM usage. This is seen in Table 3.35 – Table 3.37 and visualized in Figure 3.22 – Figure 3.24. Indeed, the average of median performance in energy consumed by the frequentist and Bayesian pipelines increases by $5.06\times$, $13.65\times$, and $5.78\times$ for CPU, GPU, and RAM Wh, respectively, when fitting our Bayesian pipelines.

Table 3.33: The median, IQR, and outlier bounds of CPU power average over k -fold cross-validation and grouped by pipeline.

Summary Statistics: Avg. CPU Power (W)								
Pipeline	Median	Q1	Q3	IQR	Lower fence	Upper fence	No. low outliers	No. high outliers
CSPLDA	26.974	22.736	28.118	5.381	14.665	36.189	2	0
CSPBLDA	0.663	0.386	0.957	0.571	-0.47	1.812	0	0
CSPSVM	18.698	13.288	26.914	13.625	-7.15	47.352	0	0
CSPGP	0.804	0.731	1.278	0.547	-0.09	2.098	0	0
TSLR	22.463	14.745	26.519	11.774	-2.917	44.181	0	0
TSBLR	4.218	3.275	5.566	2.291	-0.161	9.001	0	0
TSSVM	5.679	1.163	13.83	12.668	-17.839	32.832	0	0
TSGP	2.021	1.874	2.135	0.262	1.481	2.528	4	1
SCNN	4.59	2.723	8.644	5.921	-6.158	17.525	0	0
BSCNN	1.25	0.902	1.721	0.82	-0.328	2.951	0	0
DCNN	5.128	3.072	7.434	4.362	-3.471	13.977	0	0
BDCNN	1.706	1.082	2.517	1.435	-1.071	4.67	0	0

Table 3.34: The median, IQR, and outlier bounds of GPU power average over k -fold cross-validation and grouped by pipeline.

Summary Statistics: Avg. GPU Power (W)								
Pipeline	Median	Q1	Q3	IQR	Lower fence	Upper fence	No. low outliers	No. high outliers
CSPLDA	69.72	69.304	69.985	0.681	68.282	71.007	1	1
CSPBLDA	99.625	86.385	143.311	56.926	0.995	228.701	0	0
CSPSVM	69.795	69.505	69.868	0.362	68.962	70.412	1	0
CSPGP	358.115	327.167	364.928	37.76	270.527	421.568	0	1
TSLR	69.642	69.375	69.987	0.612	68.457	70.906	0	0
TSBLR	305.978	263.629	339.818	76.189	149.345	454.101	0	0
TSSVM	69.621	69.474	70.142	0.669	68.471	71.145	0	0
TSGP	303.789	292.289	313.552	21.262	260.396	345.446	3	0
SCNN	149.773	138.909	225.411	86.501	9.158	355.162	0	0
BSCNN	371.497	308.136	413.706	105.569	149.782	572.06	0	1
DCNN	133.828	126.825	141.644	14.818	104.598	163.872	0	1
BDCNN	262.9	218.094	321.828	103.734	62.492	477.429	0	2

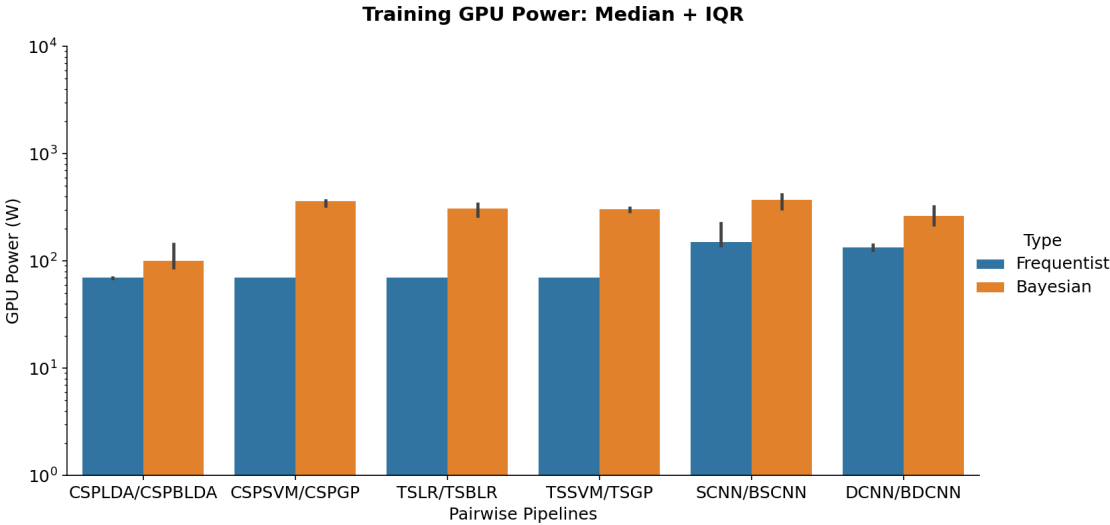


Figure 3.21: The median and IQR of average training GPU power plotted on logarithmic scale.

These findings reiterate that while fitting our Bayesian pipelines was more resource intensive relative to training our frequentist classifiers, the median cost of our Bayesian ML evaluation was low compared to the population of energy consumers.

Table 3.35: The median, IQR, and outlier bounds of CPU energy average over k -fold cross-validation and grouped by pipeline.

Summary Statistics: Avg. CPU Energy (Wh)								
Pipeline	Median	Q1	Q3	IQR	Lower fence	Upper fence	No. low outliers	No. high outliers
CSPLDA	0.466	0.424	0.549	0.125	0.237	0.736	0	3
CSPBLDA	19.738	13.701	27.93	14.229	-7.642	49.273	0	2
CSPSVM	0.642	0.46	0.948	0.488	-0.271	1.679	0	3
CSPGP	20.891	13.096	22.642	9.547	-1.224	36.962	0	2
TSLR	0.774	0.47	1.03	0.56	-0.371	1.871	0	1
TSBLR	3.589	2.614	4.675	2.061	-0.477	7.765	0	2
TSSVM	2.62	0.887	10.764	9.877	-13.928	25.58	0	3
TSGP	8.217	7.378	9.825	2.448	3.706	13.497	0	4
SCNN	5.931	2.939	10.215	7.276	-7.975	21.129	0	3
BSCNN	15.228	10.43	22.407	11.977	-7.537	40.373	0	3
DCNN	5.591	3.565	9.347	5.783	-5.109	18.021	0	3
BDCNN	13.432	8.303	19.403	11.1	-8.347	36.052	0	4

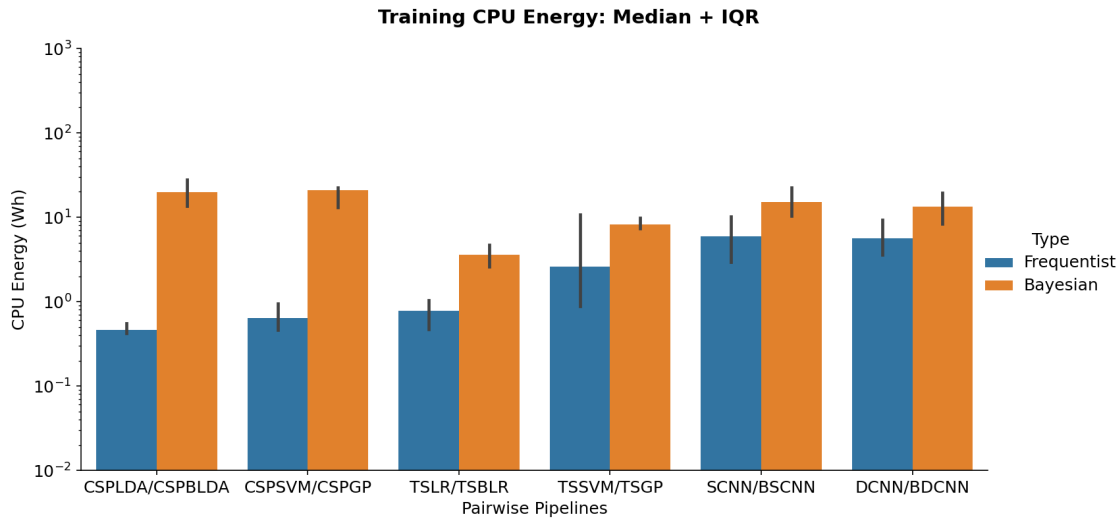


Figure 3.22: The median and IQR of the average CPU training energy plotted on a logarithmic scale.

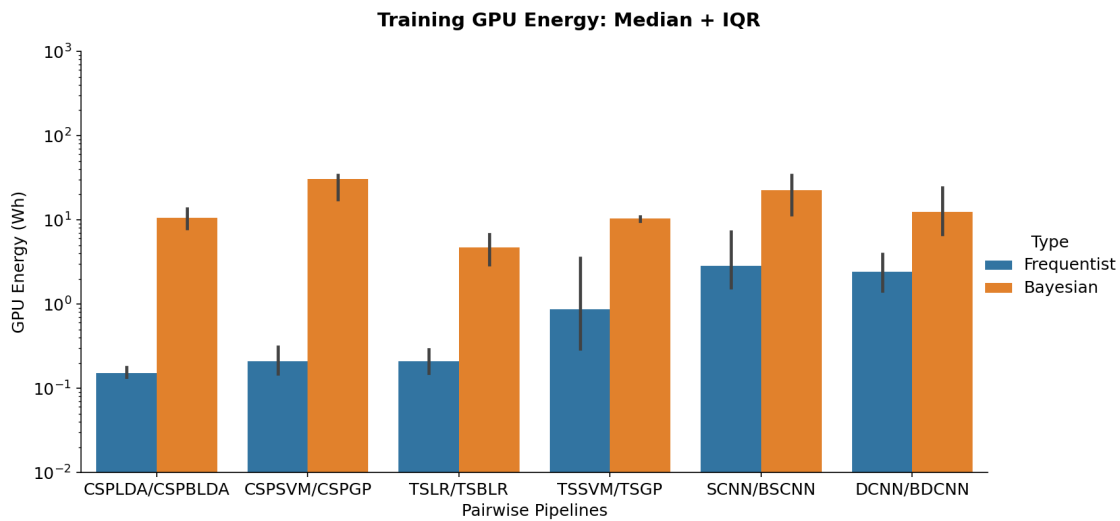


Figure 3.23: The median and IQR of average training GPU energy plotted on logarithmic scale.

Table 3.36: The median, IQR, and outlier bounds of GPU energy average over k -fold cross-validation and are grouped by pipeline.

Summary Statistics: Avg. GPU Energy (Wh)								
Pipeline	Median	Q1	Q3	IQR	Lower fence	Upper fence	No. low outliers	No. high outliers
CSPLDA	0.15	0.135	0.177	0.042	0.073	0.239	0	3
CSPBLDA	10.638	7.991	13.585	5.594	-0.4	21.975	0	3
CSPSVM	0.208	0.149	0.311	0.163	-0.096	0.556	0	3
CSPGP	30.596	17.468	33.84	16.372	-7.09	58.399	0	2
TSLR	0.211	0.151	0.288	0.137	-0.056	0.494	0	2
TSBLR	4.707	2.955	6.768	3.813	-2.764	12.487	0	2
TSSVM	0.863	0.292	3.544	3.252	-4.586	8.423	0	3
TSGP	10.319	9.625	10.864	1.239	7.766	12.723	1	4
SCNN	2.838	1.568	7.289	5.721	-7.014	15.871	0	2
BSCNN	22.453	11.563	34.221	22.658	-22.424	68.209	0	3
DCNN	2.406	1.439	3.897	2.458	-2.247	7.584	0	3
BDCNN	12.398	6.78	24.265	17.484	-19.446	50.491	0	2

Table 3.37: The median, IQR, and outlier bounds of RAM energy average over k -fold cross-validation and are grouped by pipeline.

Summary Statistics: Avg. RAM Energy (Wh)								
Pipeline	Median	Q1	Q3	IQR	Lower fence	Upper fence	No. low outliers	No. high outliers
CSPLDA	0.151	0.137	0.177	0.04	0.077	0.238	0	3
CSPBLDA	6.403	4.439	10.953	6.514	-5.332	20.724	0	3
CSPSVM	0.21	0.149	0.312	0.163	-0.096	0.556	0	3
CSPGP	5.993	3.762	6.503	2.741	-0.349	10.614	0	2
TSLR	0.213	0.152	0.29	0.139	-0.056	0.499	0	2
TSBLR	1.074	0.796	1.39	0.594	-0.095	2.28	0	2
TSSVM	0.868	0.292	3.568	3.276	-4.621	8.482	0	3
TSGP	2.358	2.183	2.669	0.486	1.454	3.398	0	4
SCNN	1.362	0.673	2.171	1.498	-1.574	4.417	0	3
BSCNN	4.059	2.933	5.837	2.904	-1.423	10.193	0	3
DCNN	1.207	0.805	2.023	1.218	-1.022	3.849	0	3
BDCNN	3.277	2.156	4.807	2.651	-1.82	8.784	0	4

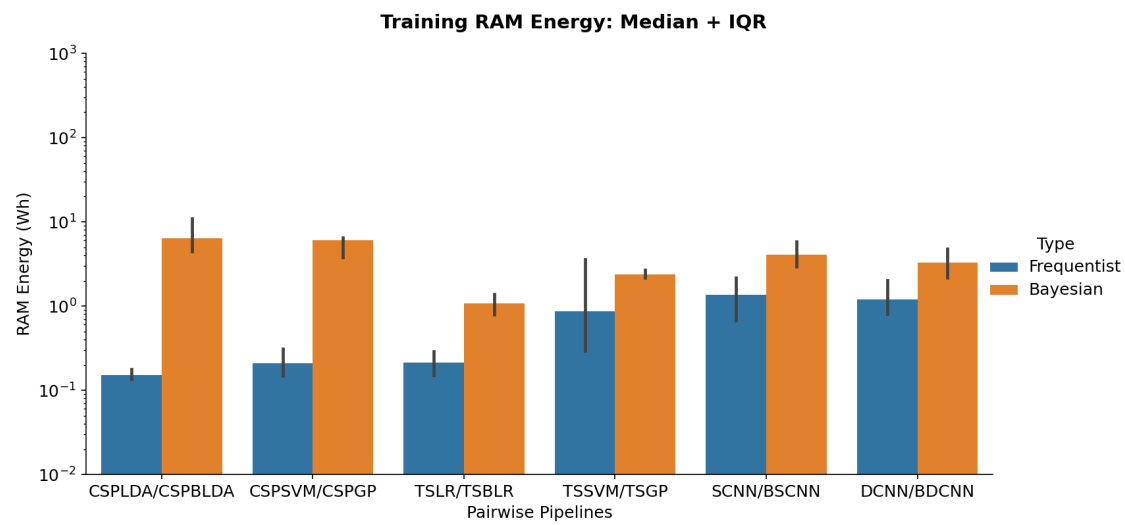
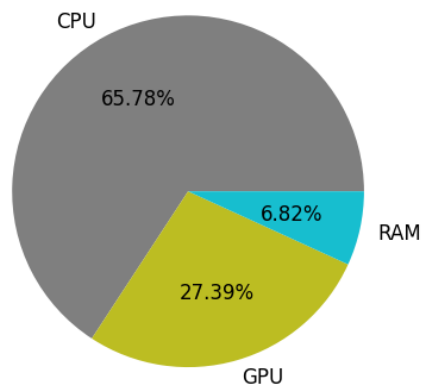


Figure 3.24: The median and IQR of average training RAM energy plotted on logarithmic scale.

Energy Consumed: Frequentist Pipelines (Wh)



Energy Consumed: Bayesian Pipelines (Wh)

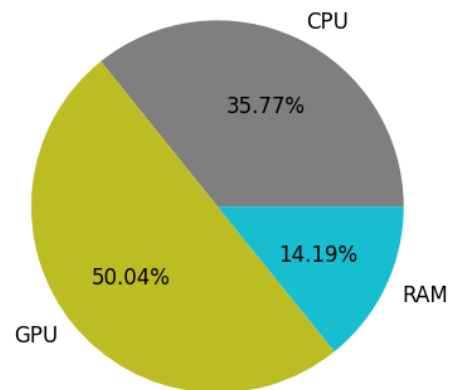


Figure 3.25: Pie charts for the percentage of energy consumed (Wh) by CPU, GPU, and RAM.

Chapter 4

DISCUSSION

4.1 *Validity of Methods*

We wanted to evaluate the benefits and costs of Bayesian ML for left hand and right hand MI-EEG classification. Our null hypothesis was that Bayesian ML would not significantly affect prediction performance compared to frequentist methods. For this purpose, we used 20 publicly available BCI datasets. For each mode, we took 6 measurements in total, from discrimination, proper scoring rules, and calibration. Our pipelines were frequent and Bayesian analogs of each other, where frequentist classifiers were selected from the top performers of recent MOABB benchmarks [21].

There were 2 pipelines each from 3 categories of classifiers well-established in BCI research. The evaluation conducted was conducted for all available subjects and sessions. The models were trained with k -fold cross-validation, collecting prediction performance measurements from each fold. With the results of measuring all our models, we accumulated scores from heterogeneous datasets with meta-analysis instruments. More specifically, we assumed a 3-level meta-analysis, with level-2 being pipeline categories within datasets, and level-3 clusters being modeled as datasets themselves.

4.1.1 *Datasets*

We selected all datasets available via MOABB PyPI version 1.5 that fit our criteria: Must be left hand and right hand MI-EEG, have at least 8 subjects, and not be a subset of another selected dataset. This approach was unbiased. It ensured that the beliefs we would draw from the results of our experiment would be based on the broadest available sample of the population. In other words, our selection criteria supported the generalizability of our

assessment. It strengthened the external validity of our experiment design. The alternative would be selective reporting of available data that would distort inference.

4.1.2 *Paradigm*

Our signal preprocessing was a funnel-based approach. In the first stage, we applied filters that reduced the size of our data. The second stage performed the transformation on the remaining relevant data. Our rules for first stage filtering were decided via raw data exploration and runtime trial and error. In the latter case, as is the nature when working with large datasets or models and limited compute resources, in certain cases our memory usage reached beyond exhaustion. Our response to reaching resource limits with our largest datasets, namely Stieger2021, was to reduce dimensionality of our data.

Our response to reaching resource limits with our largest datasets, namely Stieger2021, was to reduce dimensionality of our data. Since motor imagery and action signals occur along the sensorimotor cortex of the brain (see Figure 1.6), we removed all EEG channels outside of this region, considering them noise. This approach to dimensionality reduction is common in BCI research [60]. That is, it is a strategy that leverages the knowledge of the BCI domain. Furthermore, we downsampled all signal frequencies to 128 Hz. This strategy also leverages the knowledge of the BCI domain saying that sampling frequencies much higher than this anchor create redundant data points for EEG [60].

In the case of Stieger2021 we had to further reduce the size of data specifically to fit our Bayesian neural networks. For all other datasets and pipelines, we were able to evaluate cross subject signals with all sessions. To maintain consistent evaluations with Stieger2021 in all of our pipelines, we included only sessions 1–6 of Stieger2021 in our experiment. Together, both dimensionality reduction and slicing Stieger2021 summarize the filtering rules assumed by our study. These filtering rules allowed us to conduct our experiment and maintain a representative sample of all data that we could have observed.

For post-funneling transformation, we applied only 8 Hz and 32 Hz frequency high pass and low pass temporal filtering. This frequency range leverages the known bounds of the MI-

EEG signals. Any other preprocessing, such as spatial filtering or feature engineering, was left to pipelines to define. This minimal approach to data preprocessing effectively provides relevant, raw MI-EEG recordings for which varying and novel additional steps encapsulated within a pipeline defined by a BCI researcher may be applied. Indeed, our pipelines encoded supervised spatial filtering (CSP, TS) or feature extraction and selection corresponding to their category.

Our approach to signal preprocessing was in line with the domain of BCI research. It followed established practices and was valid. Meanwhile, our choice of scoring rules was motivated by the desire to assess the prediction performance of our models with a wide range of measurement classes. Discrimination (AUROC, MCC) measures the ability of a model to correctly rank one label above another. However, the probabilities may not be meaningful in absolute terms. This is because optimizing for discrimination can be gamed. For example, given a sample where some label is 70 percent of all labels, a model optimizing for discrimination may reasonably predict this label 100 percent of the time.

In the same scenario, though, optimizing for proper scoring rules (NLL, Brier score) would always penalize for reporting labels with the wrong probability. That is, the optimum is the honest reporting of true probabilities. However, at the same time, for finite samples, a model optimized for proper scoring may still be miscalibrated. Calibration (ECE, MCE) measures whether predicted probabilities match empirical frequencies. That is, on prediction bins, calibration asks whether the confidence in a bin matches the accuracy in that bin. With small datasets, as many in BCI research, optimizations for the sample cross-entropy versus population cross-entropy can diverge substantially.

Coming full circle, optimizing for calibration does not inherently optimize for discrimination. That is because always predicting labels according to their prior distribution results in perfect calibration but zero discrimination because the prediction never changes with the input. In other words, a model optimized purely for calibration learns the class frequencies but ignores the features entirely. It learns nothing about ranking individual trials. Thus, we see that the metrics and their classes of measurement that we chose to collect as part of our

study cover conceptually distinct dimensions.

Indeed, for aggregate scalar metrics that evaluate single probabilistic predictions, our selection of scoring rules is thorough. Discrimination assesses ordinal ranking quality, though is probability scale invariant. Calibration gives us the reliability of our probability estimates against empirical frequencies. Proper scoring rules have joint sensitivity to both and decompose into calibration and discrimination components. With these metrics, we are able to draw conclusions about the quality of probabilistic predictions in our experiment.

4.1.3 Pipelines

Our frequentist pipelines were selected according to the best performing pipelines from recent MOABB benchmarks [22]. Note that performance on these benchmarks was measured using AUROC alone. However, at the same time, BCI research often only measures performance in terms of AUROC or accuracy. In other words, these pipelines are likely familiar to many BCI researchers due to their relatively high discrimination scores. Given that discrimination is how MI-EEG classifiers are most commonly ranked, this was a reasonable approach to selecting frequentist pipelines for our experiment.

Most of the frequentist pipelines in our evaluation had simple Bayesian analogs found by placing probability distributions on the weights. However, unlike the other predictors that rely on MLE based optimization, SVMs are optimized geometrically. That is, there is no obvious Bayesian translation for SVMs. For this, we considered that SVMs are characterized as kernel methods. Furthermore, Gaussian processes are kernel methods and explicitly Bayesian. Indeed, there is a close correspondence between GP classifier MAP optimization and SVM optimization [68]. Therefore, benchmarking SVMs versus GPs was the most sensible comparison.

Implementation of the frequentist pipelines was straightforward, since major components are already offered through libraries like scikit-learn, PyRiemann, and BrainDecode. Furthermore, previously successful hyperparameters are also documented via MOABB. Implementation of the Bayesian pipelines was also relatively straightforward, since, as described

above, doing so largely involved placing probability distributions on the weights of models. Considering that we used the PyMC library to build our Bayesian pipelines, it was simple to describe these models as PGMs since PyMC offers declarative programming rather than imperative programming.

For the most part, selecting reasonable prior probability distributions for our Bayesian pipelines was also simple. That is because choosing a standard normal distribution was our default choice. Given that we centered and scaled our features before modeling, $\mathcal{N}(0, 1)$ is a principled weakly informative prior. In terms of entropy that measures uncertainty in a distribution, $\mathcal{N}(0, 1)$ is the maximum entropy prior over the reals constrained by mean=0 and variance=1. This makes it a natural default when the data have been centered and scaled.

In cases where we did not use a standard normal prior, our motivation was decided by the nature of the weights being modeled. For example, given nonnegative weights on a linear scale such as a Gaussian distribution variance, we selected a half standard normal prior distribution. Likewise, for nonnegative weights on a logarithmic scale, we chose a log normal prior. These assumptions we made gave us weakly informative priors that respected the constraints of the weights. As another example, to model the label priors, we used a distribution $\text{Beta}(2, 2)$, which is constrained to $[0, 1]$. For a prior on a probability, $\text{Beta}(2, 2)$ is a weakly informative prior more appropriate than $\mathcal{N}(0, 1)$, which covers all reals.

Generally, with the knowledge that our inputs were centered and scaled before modeling, our prior beliefs for the weight distribution were made such that the prior distributions remained weakly informative. In the case of our sparse approximations for Gaussian processes, our choice for the number of inducing points to be fixed at 100 for all datasets regardless of size was supported by the spatial dispersion of z -score normalized inputs. That is, all datasets had features that were tightly packed: centered around zero and had unit variance before being modeled. Therefore, it was justifiable that we assumed that 100 inducing points could reasonably summarize all datasets for sparse GPs.

Our choice of pipelines and configurations for them was grounded. The decision to use the

MCMC based approximation for our Bayesian pipelines was also sensible. Indeed, in terms of approximation accuracy, MCMC is asymptotically exact, whereas variational inference is often structurally inaccurate. Given comparable compute costs for the MCMC and VI approximation, the benefits of MCMC make it the sensible choice. In fact, using MCMC at runtime to approximate the posterior distribution of our Bayesian models fits well within our training budget.

4.1.4 *Evaluation*

The evaluation of our pipelines was carried out by the MOABB library. The motivational purpose of MOABB is to be the default evaluation loop for combined BCI and ML research. Indeed, its implementation of outer k -fold cross-validation for anchoring performance expectations, and inner k -fold cross-validation for hyperparameter optimization within folds is a sound design that BCI and ML researchers alike would otherwise want to implement for their own experiments. Given that MOABB also centralizes publicly available EEG datasets, all that remains is for researchers to bring their own novel pipelines for evaluation.

Using MOABB for our evaluation reduced the amount of development that we needed to do to conduct our experiment. The relatively wide and growing adoption of MOABB’s in the BCI literature also validates our experiment design. Its API provides a standardized and reproducible evaluation of the EEG prediction pipeline. Furthermore, as an open-source project, we were able to implement greater configurability and data collection for our experiment: configurable scoring rules and fine grained compute profiling. These features were shipped with MOABB PyPI version 1.5, adding to its expressiveness and reliability.

4.1.5 *Analysis*

We measured the differences in Bayesian treatments and frequentist baselines for subjects within each dataset. Indeed, across all datasets we measured the changes in effects with the same pairs of treatments and baselines. To pool the effects across heterogeneous datasets acquired under varying conditions, we used meta-analysis instruments. That is, within datasets

acting as primary studies, we had data points that measured differences in treatments and baselines. To account for various sources of correlation, we modeled our experiment as a 3-level meta-analysis.

At level-3 of our meta-analysis, we model across dataset heterogeneity. Within clusters at level-2, we grouped pipelines according to their category: CSP, TS, or DL. We expected effect sizes within categories to be correlated, based on the shared structure of the pipelines. Therefore, we grouped them as a precautionary measure. Furthermore, considering that all classifiers within a dataset were evaluated on the same data acquired under particular conditions, we used RVE with sensitivity analysis to model level-1 sampling error correlation. That is because we expected that all classifiers would share effects driven by characteristics specific to a data set.

Therefore, across 20 independent datasets, we measured the difference between pipeline variants. Within each dataset, we grouped differences by their corresponding category of pipelines, considering that the effects within groups would be correlated. This created 3 groups of 2 differences within each dataset. Then, additionally, we used RVE to model the sampling error correlation that appears from all treatments and the baselines that were applied to the same subjects within the datasets. With only 2 data points within subgroups, we expected level-2 variance estimations to be noisy. At level-3, 20 datasets gave us a large number of MI-EEG datasets to generate inferences.

For the size of our study, the Knapp-Hartung method based on t -distribution was principled. t -test based assessment was used to estimate the presence of an effect, its corresponding size, and our certainty in the estimation. We also report on the detailed variance components. For RVE, conducting sensitivity analysis asserted the robustness of our results. Meanwhile, moderator analysis lets us inspect whether sample size affected scoring for cross subject evaluation. Our statistical analysis methods provided robustly measured evidence supported by theory that supported the generation of new hypotheses.

4.2 Interpretation of Results

The purpose of subsection 4.2.1 is to consider the quality of the posterior sampling diagnostics from the fitting of our Bayesian pipelines. This gauges whether the results of the raw score and the statistical analysis were based on well approximated models. subsection 4.2.2 explores the relationship between discrimination, calibration, and proper scoring rules, and our measurements for these tell us about the MI-EEG classification. In subsection 4.2.3 we conclude the results of our hypothesis testing and what it could mean for future BCI and ML research. Then subsection 4.2.4 considers the tradeoffs of our experiment and what these could mean for testing additional hypotheses.

4.2.1 Posterior Sampling

The results of the posterior sampling diagnostics of section 3.1 generally tell us that our Bayesian models were well approximated within the pipelines. ESS measurements averaging over 400 in Table 3.1 and Figure 3.1 are a good sign. At the same time, worst case performance was less ideal for the Gaussian processes. This means that our sparse approximations were not always desirable. However, for the best (worst) case Gaussian process diagnostics over 1000 ESS, these models were adequate. Still, $\hat{R}$ was at most 1.01 for the Gaussian processes and also for other pipelines as well.

Overall, our Bayesian pipelines were well approximated with the MCMC. That is, the prior distributions, their hyperparameters, and non-centered parameterization helped to make the posterior distributions smooth so that they were simple to traverse. This makes our raw prediction performance scores trustworthy since they were generated from well-approximated Bayesian models. Indeed, a well-fit model is prerequisite for observing and inferring the potentials of Bayesian ML in BCI research. Detailed sampling diagnostics and convergence plots are reported in our Figshare [25].

4.2.2 *Raw Scores*

Table 3.12 from section 3.2 tells us that a much higher percentage of proper scoring rule measurements from our experiment were worse than random chance classification. Furthermore, a higher percentage of NLL measurements was worse than Brier scores. The latter can be explained by the NLL being less forgiving than the Brier score when penalizing incorrect probability estimates. Generally, NLL and BS scoring worse than random chance more than AUROC and MCC tells us that modeling high quality probabilities for MI-EEG classification is more difficult than ranking labels for discrimination.

This random classification imbalance suggests that although methods such as CSP can maximize the discrimination of labels (see section C.2), EEG remains a low signal to noise ratio source of data. This explains why subsection 3.2.1 suggests that discrimination for frequentist and Bayesian variants of models is essentially the same. That is, performing discrimination with a probabilistic classifier does not change the classification decision boundary. However, this probabilistic classifier reduces the performance of random prediction worse by around 10% according to Table 3.12.

Bayesian methods provided measurable benefits with probability quality. Furthermore, subsection 3.2.3 suggests that they improve uncertainty modeling as measured by calibration metrics ECE and MCE. In other words, Bayesian models improved the probability estimation around the same decision boundary. Indeed, the decomposition of proper scoring rules into resolution (discrimination) and reliability (calibration) tells us that our frequentist models were overconfident [18]. These effects on performance may be slight for static ML models. However, in environments where models must adapt to new settings, it is reasonable to consider whether these effects may be compounded.

4.2.3 *Meta-Analysis*

Our meta-analyses told us that calibration was significantly affected when using Bayesian pipelines. For proper scoring rules, composed of resolution and reliability, to not be affected,

the resolution must have changed counteractively. Resolution is richer than discrimination. The former is probabilistic, whereas discrimination is only based on ranking and is therefore more coarse. Resolution measures a model’s ability to distinguish different outcomes, and reliability tracks statistical consistency. Reliability is synonymous with calibration with measurements that include ECE and MCE [18].

Discrimination metrics are derived from the confusion matrix and require a decision threshold to classify predictions as TP, TN, FP or FN. For balanced classes, the threshold is often 0.5 for converting probability outputs into class assignments. This is a lossy compression because the confidence in the model, the variation in confidence between samples, and how well it tracked the true frequencies collapse into confusion matrix bins. Indeed, this explains how the effects of Bayesian models could be insignificant for discrimination and proper scoring rules, yet significant for calibration.

The fact that effects to calibration were significant for Bayesian models but insignificant for proper scoring rules means that changes in resolution balanced out reliability effects. At the same time, we did not measure a difference in discrimination, though, these metrics are coarse and insensitive to slight variations in resolution. However, we can infer from all the types of metrics we collected that resolution was affected by Bayesian models. Our inference tells us that frequentist models are overconfident for MI-EEG classification.

Therefore, based on the large scale evidence of prediction performance data that we acquired, our conclusion is that Bayesian models reduce overconfident MI-EEG classification. However, we consider that the effect sizes for the changes are small, as we saw with calibration metrics in ECE and MCE. That is, the effects of Bayesian models are small for single-line static prediction pipelines. Hypothetically, our measurements could point to meaningful differences for sequential learning scenarios where the effects of model reliability compound. Indeed, we already know from the BCI literature that Bayesian ML has been shown to improve the breadth of adaptability populations in transfer learning [47].

It would be interesting for future research to evaluate the quantitative effects and possible drift of resolution and reliability to BCI predictors in adaptive learning scenarios. Our guess,

generated by the results of our research, is that for sequential learning, we would see the effects of Bayesian models separate the difference between frequentist pipelines that we found in our study. Indeed, this newly generated hypothesis is supported by theory telling us that Bayesian statistics offer optimal updates to our beliefs. In addition, Bayesian ML is known to be beneficial in small data scenarios, which may fit adaptive learning, where sequences of steps produce relatively small data between model updates.

At the same time, we acknowledge that Bayesian ML is not immune to distribution shift [58]. In scenarios of long sequences of model updates or single large changes between steps, even uncertainty quantification around classification may be insufficient to hedge prediction. For these cases, rerunning feature engineering or refitting embedding spaces would be appropriate. As in our experiment, it may be enough to do this deterministically. In other words, we could still use frequentist methods to extract feature or embeddings, and then use Bayesian approaches for prediction to avoid overconfidence and adaptability in feature space.

The results of our large scale experiment in MI-EEG classification justifies research in this direction. To observe subtle effects on probabilistic prediction, resolution, reliability, and proper scoring rules can be measured to assess frequentist and Bayesian model performance at adaptability. Potentially, Stieger2021 as a longitudinal study with more than 60 subjects and 7–11 sessions per subject could be used as a data set to carry out this hypothetical experiment. However, researchers conducting such an experiment should be aware that sessions 1–6 of Stieger2021 were used in our meta-analyses, and so reusing Stieger2021 to test such a hypothesis based on our findings increases risks of Type I false positive errors.

4.2.4 *Computational Costs*

Our computational costs from fitting frequentist versus Bayesian pipelines followed a clear pattern: Training costs in terms of duration, energy consumed, and emissions scaled geometrically. However, in comparison to even common household appliances, our experiment was low cost. This finding supports future research on the benefits of Bayesian ML for adaptive

learning in the BCI domain. Of course, having access to a workstation with ample resources, such as the server to which we had access for our experiment, is helpful too for supporting larger datasets and pipelines.

4.3 Limitations

The between dataset heterogeneity of our experiment was measured with low sampling error. However, the level-2 precision of our multilevel meta-analysis generated by effects from pipeline categories (Raw, Riemannian, Deep Learning) within datasets was more noisy. This is because each category within datasets held just two pipelines. At the same time, though, level-2 precision was not our central concern. The purpose of our research was to assess the cumulative effects of Bayesian ML across a large sample of diverse datasets. As described in section 2.3, our choice of baseline classifiers and their categorization was an extension of MOABB’s research [21].

Hypothetically, a study specifically interested in reducing the percentage of level-2 heterogeneity due to sampling error as measured by the Higgins & Thompson’s I^2 statistic could include additional pipelines per category. For example, the EEGNet deep learning pipeline has been published with several variations of its architecture [49]. Like ShallowConvNet and DeepConvNet, EEGNet is based on convolutional neural networks. Perhaps more interesting, though, would be to research graph neural networks. Whereas CNNs operate over Euclidean domains, GNNs assume non-Euclidean domains [14, 38]. Indeed, BrainDecode implements at least one GNN [4].

Yet another type of model that could be added to rework of our experiment is hierarchical models. That is, rather than estimating a decision boundary only based on population-level effects, we could include estimates for the population, individual subjects, and within trials. Indeed, Bayesian hierarchical models would let us place priors on each of these levels and push inference through this model. Of course, hierarchical models are more computationally expensive than flat models such as those from our experiment. Based on our results, though, we could expect training costs for MI-EEG classification to still be low in absolute terms.

Another limitation of our experiment was that our evaluation did not explore hyperparameter space for our models. As described in section 2.3, this decision was made with the interest of limiting our training costs largely in terms of duration. MOABB natively offers grid search, though, this method is known to be inefficient and only offer sparse coverage of high dimensional feature space. However, MOABB integrates with Optuna which offers a much broader selection of hyperparameter search including genetic algorithms and Bayesian optimization [2]. Including hyperparameter optimization in our experiment would increase the robustness of our report.

In subsection 4.2.3 we interpreted the resolution of both our frequentist and Bayesian classifiers through the change in effects on discrimination metrics and proper scoring rules. We discussed how discrimination metrics are coarse grained measurements that track ranking ability without consideration of probability magnitudes. Indeed, measurements of resolution or sharpness that are sensitive to the magnitude of probabilities include the variance of predictions, Shannon entropy, or the Kullback-Leibler divergence computing how far predictions are from maximum entropy [9].

In a GitHub fork of MOABB we implemented a method to collect the raw predictions generated by our models when being tested against holdout sets. We built this after completing our meta-analysis and interpretation of our results. Our motivation for building this feature was to be able to explicitly measure probabilistic sharpness from the predictions of our pipelines rather than inferring these effects through other metrics. To make this feature available to BCI researchers using MOABB and interested in probabilistic machine learning, we would need to create a feature request through GitHub.

Our meta-analyses were limited to hypothesis tests of paired differences for one metric. However, meta-analytic structural equation models (SEMs) allow us to perform multivariate meta-analyses which include more than one outcome of interest [39]. It would be useful to analyze the prediction performance of our models from the perspective of the correlation between resolution, reliability, and proper scoring.

Furthermore, conventional meta-analyses rely on frequentist methods and are often based

on normal distributions. Bayesian meta-analysis allows us to perform richer experiments. We can directly model the uncertainty in our estimate of τ^2 heterogeneity, calculate the exact probability that μ or τ^2 is smaller or larger than some specified value, and integrate prior knowledge and assumptions when calculating meta-analyses [39].

We also acknowledge that throughout our literature review, we set a boundary on our research goals to not be responsible for descriptions of our ML models from the perspective of measure theory. However, for rigorous studies of probability theory and statistical models, measure theory is foundational. Indeed, there are deep connections relating the inductive biases of the classifiers we researched. Measure theory provides frameworks for proving the correctness and discovering the utility of methods we used. For example, a core mechanism of statistical learning theory (SLT) is to use PAC-Bayes to leverage and explain the generalization abilities of machine learning models [71].

4.4 Future Work

The hypothesis that motivated our research was that Bayesian learning offers improved prediction performance in EEG classification. Our methods to test this hypothesis were to design a large-scale meta-analysis that would aggregate the results of many primary studies, each of which evaluated multiple pipelines on a multifaceted MI-EEG dataset. The results of our experiment were fruitful. It yielded interesting interpretations into the decomposition of probabilistic sharpness and calibration in EEG classification. Our work uncovered new avenues for future novel research.

The hypothesis we generated from the results of our experiment is a defensible direction of future research. From the BCI literature, we know that Bayesian ML benefits the transferability of EEG classifiers across broad populations [47]. Furthermore, the ML literature tells us that modeling posterior uncertainty can be critical when we make predictions in a decision-oriented system where bad decisions can have significant consequences such as in reinforcement learning or robotics [27].

These perspectives from our research and the wider literature suggest that Bayesian ML

may be useful for building high performing BCI with robust sequential updates. As previously suggested, reinforcement learning is a potential avenue for researching BCI classifiers with online learning components. Dynamical systems focused on modeling differential equations such as spatiotemporal dynamics are another possible direction of research [9, 17]. Furthermore, the BCI literature documents hidden Markov models (HMMs) being applied to EEG classifiers [59]. Indeed, HMMs are another method that could be used for testing the effectiveness of Bayesian ML in EEG classification with sequential learning.

Additional directions of future work include exploiting any of the limitations in our experiment. As described above, these limitations include novel research related to the paradigms, pipelines, evaluations, and analysis of our methods. Indeed, another direction of research is to compare frequentist methods and Bayesian models for datasets besides left hand and right hand motor imagery EEG. For example, ensemble models combining multiple pipelines or types of data is active research [9]. Any of these topics is capable of generating original results based on our research.

Chapter 5

CONCLUSION

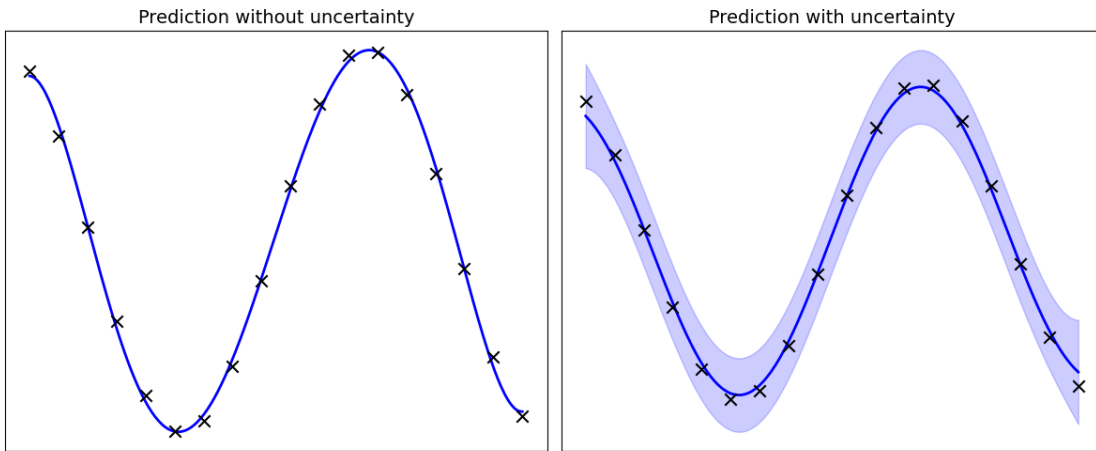


Figure 5.1: Sample frequentist and Bayesian estimates of a dataset.

This thesis presented a large-scale meta-analytic evaluation of Bayesian versus frequentist machine learning for motor imagery EEG classification. Across 20 publicly available BCI datasets, six frequentist pipelines spanning spatial filtering, Riemannian geometry, and deep learning were benchmarked against pairwise Bayesian variants. Prediction performance was assessed across conceptually distinct measurement classes: discrimination, proper scoring rules, and calibration.

Posterior sampling diagnostics confirmed that our Bayesian models were well approximated, establishing a reliable basis for inference. Raw score analysis revealed that modeling high quality probabilities for MI-EEG classification is more difficult than ranking labels for discrimination, consistent with the low signal-to-noise ratio of EEG. Meta-analysis yielded statistically significant improvements in calibration – specifically ECE and MCE –

for Bayesian pipelines, while discrimination and proper scoring rules did not show significant difference.

This pattern is coherent: Bayesian models reduced overconfident probability estimates around the same decision boundary, with resolution changes counteracting reliability improvements such that proper scoring rules remained unaffected in aggregate. Computational cost analysis further showed that Bayesian training overhead, although substantially larger than frequentist baselines, remained economically modest in absolute terms.

Taken together, the evidence supports the conclusion that Bayesian ML reduces overconfident MI-EEG classification, with effect sizes that are small but theoretically meaningful. We hypothesize that these effects would compound in adaptive learning scenarios that involve sequential model updates, where the reliability advantages of Bayesian predictors may accumulate over time. Longitudinal datasets such as Stieger2021 offer a promising avenue to testing this hypothesis. Probabilistic resolution and reliability are proposed as richer indicators than discrimination alone to evaluate and maintain high-performing BCI systems in such settings.

BIBLIOGRAPHY

- [1] Oriol Abril-Pla, Virgile Andreani, Colin Carroll, Larry Dong, Christopher J. Fonnesbeck, Maxim Kochurov, Ravin Kumar, Junpeng Lao, Christian C. Luhmann, Osvaldo A. Martin, Michael Osthege, Ricardo Vieira, Thomas Wiecki, and Robert Zinkov. PyMC: A modern and comprehensive probabilistic programming framework in Python. *PeerJ Computer Science*, 9(e1516), 2023.
- [2] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 2623–2631, New York, NY, USA, 2019. Association for Computing Machinery.
- [3] Aphex34. Max pooling. https://commons.wikimedia.org/wiki/File:Max_pooling.png, 2015. Licensed under CC BY-SA 4.0.
- [4] B. Aristimunha, P. Guetshel, M. Wimpff, L. Gemein, C. Rommel, H. Banville, M. Sliwowski, D. Wilson, S. Brandt, T. Gnassounou, J. Paillard, B. Junqueira Lopes, S. Sedlar, T. Moreau, S. Chevallier, A. Gramfort, and R. T. Schirrmeister. Braindecode: toolbox for decoding raw electrophysiological brain data with deep learning models, 2025.
- [5] Bruno Aristimunha, Igor Carrara, Pierre Guetschel, Sara Sedlar, Pedro Rodrigues, Jan Sosulski, Divyesh Narayanan, Erik Bjareholt, Quentin Barthelemy, Robin Tibor Schirrmeister, Reinmar Kobler, Emmanuel Kalunga, Ludovic Darnet, Cattán Gregoire, Ali Abdul Hussain, Ramiro Gatti, Vladislav Goncharenko, Anton Andreev, Jordy Thie-

- len, Thomas Moreau, Yannick Roy, Vinay Jayaram, Alexandre Barachant, and Sylvain Chevallier. Mother of all bci benchmarks, 2025.
- [6] Alexandre Barachant, Quentin Barthélemy, Jean-Rémi King, Alexandre Gramfort, Sylvain Chevallier, Pedro L. C. Rodrigues, Emanuele Olivetti, Vladislav Goncharenko, Gabriel Wagner vom Berg, Ghiles Reguig, Arthur Lebeurrier, Erik Bjäreholt, Maria Sayu Yamamoto, Pierre Clisson, Marie-Constance Corsi, Igor Carrara, Apolline Mellot, Bruna Junqueira Lopes, Brent Gaisford, Ammar Mian, Anton Andreev, Gregoire Cattan, and Arthur Lebeurrier. pyriemann, 7 2025.
- [7] Alexandre Barachant, Stéphane Bonnet, Marco Congedo, and Christian Jutten. Multiclass brain–computer interface classification by riemannian geometry. *IEEE Transactions on Biomedical Engineering*, 59(4):920–928, 2012.
- [8] Camille Benaroch, Maria Sayu Yamamoto, Aline Roc, Pauline Dreyer, Camille Jeunet, and Fabien Lotte. When should MI-BCI feature optimization include prior knowledge, and which one? *Brain-Computer Interfaces*, 9(2):115–128, 4 2022.
- [9] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer New York, New York, NY, 1 edition, 2006.
- [10] Christopher M. Bishop and Hugh Bishop. *Deep Learning: Foundations and Concepts*. Springer Cham, 1 edition, 2023.
- [11] Benjamin Blankertz. Gentle introduction to signal processing and classification for single-trial eeg analysis. In Chang S. Nam, Anton Nijholt, and Fabien Lotte, editors, *Brain–Computer Interfaces Handbook: Technological and Theoretical Advances*, chapter 18, pages 343–370. CRC Press, 2018.
- [12] Michael Borenstein, Larry V. Hedges, Julian P. T. Higgins, and Hannah R. Rothstein. *Meta-Analysis Methods Based on Direction and p-Values*, chapter 36, pages 325–330. John Wiley & Sons, Ltd, 2009.

- [13] Stephanie Brandl and Benjamin Blankertz. Motor imagery under distraction— an open access bci dataset. *Frontiers in Neuroscience*, Volume 14 - 2020, 2020.
- [14] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.
- [15] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng, editors. *Handbook of Markov Chain Monte Carlo*. Chapman and Hall/CRC, 1 edition, 2011.
- [16] BrunelloN. Example of a deep neural network. https://commons.wikimedia.org/wiki/File:Example_of_a_deep_neural_network.png, 2021. Licensed under CC BY-SA 4.0.
- [17] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, USA, 1st edition, 2019.
- [18] Jochen Bröcker. Reliability, sufficiency, and the decomposition of proper scores. *Quarterly Journal of the Royal Meteorological Society*, 135(643):1512–1519, 2009.
- [19] Igor Carrara and Théodore Papadopoulo. Classification of BCI-EEG based on augmented covariance matrix. working paper or preprint, 2023.
- [20] Wenwen Chang, Weixuan Kong, Guanghui Yan, Renjie Lv, Kaiyue Du, Muhammad Tariq Sadiq, Bin Guo, Rong Yin, and Xuan Liu. A multi-paradigm EEG dataset for studying upper limb rehabilitation exercises. *Scientific Data*, 12(1):1877, 2025.
- [21] Sylvain Chevallier, Igor Carrara, Bruno Aristimunha, Pierre Guetschel, Sara Sedlar, Bruna Junqueira Lopes, Sébastien Velut, Salim Khazem, and Thomas Moreau. The largest eeg-based bci reproducibility study for open science: the moabb benchmark. working paper or preprint, 4 2024.

- [22] Sylvain Chevallier, Emmanuel K. Kalunga, Quentin Barthélemy, and Florian Yger. Riemannian classification for ssvep-based bci: Offline versus online implementations. In Chang S. Nam, Anton Nijholt, and Fabien Lotte, editors, *Brain–Computer Interfaces Handbook: Technological and Theoretical Advances*, chapter 19, pages 371–396. CRC Press, 2018.
- [23] Hohyun Cho, Minkyu Ahn, Sangtae Ahn, Moonyoung Kwon, and Sung Chan Jun. Eeg datasets for motor imagery brain–computer interface. *GigaScience*, 6(7):gix034, 5 2017.
- [24] Marie-Constance Corsi. Electroencephalography and magnetoencephalography. In Olivier Colliot, editor, *Machine Learning for Brain Disorders*, pages 285–312. Springer US, New York, NY, 2023.
- [25] Ethan Davis. Bayesian and frequentist motor imagery eeg classification: Model scores, compute profiling, and mcmc convergence diagnostics, 5 2026.
- [26] Ethan Davis. SPANERlab/bmi: Thesis release, 2026.
- [27] Marc Peter Deisenroth, Aldo Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020.
- [28] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *ArXiv e-prints*, 03 2016.
- [29] Bill Engels, Oriol Abril Pla, and Juan Orduz. Mean and covariance functions. In PyMC Team, editor, *PyMC examples*. Zenodo, 2023.
- [30] Tom Fletcher. Statistics on manifolds. In Xavier Pennec, Stefan Sommer, and Tom Fletcher, editors, *Riemannian Geometric Statistics in Medical Image Analysis*, chapter 2, pages 39–74. Academic Press, 2020.
- [31] Chris Fonnesbeck, Ana Rita Santos, and Sandra Meneses. Gaussian processes using numpy kernel. In PyMC Team, editor, *PyMC examples*. Zenodo, 2022.

- [32] Dylan Forenzo, Yixuan Liu, Jeehyun Kim, Yidan Ding, Taehyung Yoon, and Bin He. Integrating simultaneous motor imagery and spatial attention for eeg-bci control. *IEEE Transactions on Biomedical Engineering*, 71(1):282–294, 2024.
- [33] A. L. Goldberger, L. A. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C. K. Peng, and H. E. Stanley. PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals. *Circulation*, 101(23):e215–e220, 2000.
- [34] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [35] Moritz Grosse-Wentrup, Christian Liefhold, Klaus Gramann, and Martin Buss. Beam-forming in noninvasive brain–computer interfaces. *IEEE Transactions on Biomedical Engineering*, 56(4):1209–1219, 2009.
- [36] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1321–1330. PMLR, 08 2017.
- [37] Eva Guttmann-Flury, Xinjun Sheng, and Xiangyang Zhu. Dataset combining EEG, eye-tracking, and high-speed video for ocular activity analysis across BCI paradigms. *Scientific Data*, 12(1):587, 2025.
- [38] William L. Hamilton. *Graph Representation Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Springer Cham, 2020.
- [39] Mathias Harrer, Pim Cuijpers, Furukawa Toshi A, and David D Ebert. *Doing Meta-Analysis With R: A Hands-On Guide*. Chapman & Hall/CRC Press, Boca Raton, FL and London, 1st edition, 2021.

- [40] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York, New York, NY, 2 edition, 02 2009.
- [41] HgDeviassse. 3112189_pone.0020674.g001. https://commons.wikimedia.org/wiki/File:3112189_pone.0020674.g001.png, 2014. Licensed under public domain.
- [42] Bo Hjorth. Eeg analysis based on time domain properties. *Electroencephalography and Clinical Neurophysiology*, 29(3):306–310, 1970.
- [43] Iamozy. Human motor cortex.jpg. https://commons.wikimedia.org/wiki/File:Human_motor_cortex.jpg, 2014. Licensed under CC BY-SA 3.0.
- [44] IEA. Co2 emissions in 2023. Technical report, IEA, Paris, 2024. Licence: CC BY 4.0.
- [45] Jahobr. Coloured voronoi 3d slice. https://commons.wikimedia.org/wiki/File:Coloured_Voronoi_3D_slice.svg, 2017. Licensed under CC BY-SA 3.0.
- [46] Vinay Jayaram and Alexandre Barachant. Moabb: trustworthy algorithm benchmarking for bcis. *Journal of Neural Engineering*, 15(6):066011, 9 2018.
- [47] Vinay Jayaram, Karl-Heinz Fiebig, Jan Peters, and Moritz Grosse-Wentrup. Transfer learning for bcis. In Chang S. Nam, Anton Nijholt, and Fabien Lotte, editors, *Brain–Computer Interfaces Handbook: Technological and Theoretical Advances*, chapter 22, pages 425–441. CRC Press, 2018.
- [48] Satyam Kumar, Hussein Alawieh, Frigyes Samuel Racz, Rawan Fakhreddine, and José del R Millán. Transfer learning promotes acquisition of individual bci skills. *PNAS Nexus*, 3(2):pgae076, 02 2024.
- [49] Vernon J Lawhern, Amelia J Solon, Nicholas R Waytowich, Stephen M Gordon, Chou P Hung, and Brent J Lance. Eegnet: a compact convolutional neural network for eeg-based brain–computer interfaces. *Journal of Neural Engineering*, 15(5), 07 2018.

- [50] Min-Ho Lee, O-Yeon Kwon, Yong-Jeong Kim, Hong-Kyung Kim, Young-Eun Lee, John Williamson, Siamac Fazli, and Seong-Whan Lee. Eeg dataset and openbmi toolbox for three bci paradigms: an investigation into bci illiteracy. *GigaScience*, 8(5):giz002, 1 2019.
- [51] La librairie ADEME. Panel usages électrodomestiques année 6, 3 2026.
- [52] Haijie Liu and Xiaodong Lv. Eeg datasets of stroke patients, 12 2022.
- [53] Haijie Liu, Penghu Wei, Haochong Wang, Xiaodong Lv, Wei Duan, Meijie Li, Yan Zhao, Qingmei Wang, Xinyuan Chen, Gaige Shi, Bo Han, and Junwei Hao. An EEG Motor Imagery Dataset for Brain Computer Interface in Acute Stroke Patients. *Scientific Data*, 11(1):131, 2024.
- [54] Osvaldo A. Martin, Ravin Kumar, and Junpeng Lao. *Bayesian Modeling and Computation in Python*. Chapman and Hall/CRC, 1 edition, 2021.
- [55] Richard McElreath. *Statistical Rethinking: A Bayesian Course with Examples in R and STAN*. Chapman and Hall/CRC, 2 edition, 2020.
- [56] Jelena Mladenović, Jérémie Mattout, and Fabien Lotte. A generic framework for adaptive eeg-based bci training and operation. In Chang S. Nam, Anton Nijholt, and Fabien Lotte, editors, *Brain–Computer Interfaces Handbook: Technological and Theoretical Advances*, chapter 31, pages 595–611. CRC Press, 2018.
- [57] Kevin P. Murphy. *Probabilistic Machine Learning: An Introduction*. MIT Press, 2022.
- [58] Kevin P. Murphy. *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023.
- [59] Chang S. Nam, Inchul Choi, Amy Wadeson, and Mincheol Whang. Brain–computer interface: An emerging interaction technology. In Chang S. Nam, Anton Nijholt, and Fabien Lotte, editors, *Brain–Computer Interfaces Handbook: Technological and Theoretical Advances*, chapter 1, pages 11–52. CRC Press, 2018.

- [60] Chang S. Nam, Anton Nijholt, and Fabien Lotte, editors. *Brain–Computer Interfaces Handbook: Technological and Theoretical Advances*. CRC Press, 1 edition, 2018.
- [61] Theodore Papamarkou, Maria Skoularidou, Konstantina Palla, Laurence Aitchison, Julyan Arbel, David Dunson, Maurizio Filippone, Vincent Fortuin, Philipp Hennig, José Miguel Hernández-Lobato, Aliaksandr Hubin, Alexander Immer, Theofanis Karaletsos, Mohammad Emtiyaz Khan, Agustinus Kristiadi, Yingzhen Li, Stephan Mandt, Christopher Nemeth, Michael A. Osborne, Tim G. J. Rudner, David Rügamer, Yee Whye Teh, Max Welling, Andrew Gordon Wilson, and Ruqi Zhang. Position: Bayesian deep learning is needed in the age of large-scale ai, 2024.
- [62] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. *PyTorch: an imperative style, high-performance deep learning library*. Curran Associates Inc., Red Hook, NY, USA, 2019.
- [63] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [64] Xavier Pennec. Manifold-valued image processing with spd matrices. In Xavier Pennec, Stefan Sommer, and Tom Fletcher, editors, *Riemannian Geometric Statistics in Medical Image Analysis*, chapter 3, pages 75–134. Academic Press, 2020.
- [65] G. Pfurtscheller and F.H. Lopes da Silva. Event-related eeg/meg synchronization and desynchronization: basic principles. *Clinical Neurophysiology*, 110(11):1842–1857, 1999.
- [66] Léa Pillette, Aline Roc, Bernard N’Kaoua, and Fabien Lotte. Experimenters’ influence

- on mental-imagery based brain-computer interface user training. *International Journal of Human-Computer Studies*, 149:102603, 2021.
- [67] Ralf@ark.in-berlin.de. Motor homunculus.svg. https://commons.wikimedia.org/wiki/File:Motor_homunculus.svg, 2016. Licensed under CC BY-SA 4.0.
- [68] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 11 2005.
- [69] G. Schalk, D.J. McFarland, T. Hinterberger, N. Birbaumer, and J.R. Wolpaw. Bci2000: a general-purpose brain-computer interface (bci) system. *IEEE Transactions on Biomedical Engineering*, 51(6):1034–1043, 2004.
- [70] Robin Tibor Schirrmeister, Jost Tobias Springenberg, Lukas Dominique Josef Fiederer, Martin Glasstetter, Katharina Eggensperger, Michael Tangermann, Frank Hutter, Wolfram Burgard, and Tonio Ball. Deep learning with convolutional neural networks for eeg decoding and visualization. *Human Brain Mapping*, 38(11):5391–5420, 2017.
- [71] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, USA, 2014.
- [72] Jian Shi, Danyang Chen, Xingwei Zhao, Zhixian Zhao, Shengjie Li, Yeguang Xu, Tao Ding, Zheng Zhu, Peng Zhang, Qing Ye, Yingxin Tang, Ping Zhang, Bo Tao, and Zhouping Tang. HEFMI-ICH: a hybrid EEG-fNIRS motor imagery dataset for brain-computer interface in intracerebral hemorrhage. *Scientific Data*, 12(1):1816, 2025.
- [73] Jaeyoung Shin, Alexander von Lühmann, Benjamin Blankertz, Do-Won Kim, Jichai Jeong, Han-Jeong Hwang, and Klaus-Robert Müller. Open access dataset for eeg+nirs single-trial classification. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 25(10):1735–1745, 2017.
- [74] Stefan Sommer, Tom Fletcher, and Xavier Pennec. Introduction to differential and riemannian geometry. In Xavier Pennec, Stefan Sommer, and Tom Fletcher, editors,

- Riemannian Geometric Statistics in Medical Image Analysis*, chapter 1, pages 3–37. Academic Press, 2020.
- [75] Dustin Stansbury and Richard McElreath. Correlated features. In PyMC Team, editor, *PyMC examples*. Zenodo, 2024.
 - [76] Star. Why test error and variance has different curve in bias variance trade off graph? Cross Validated. Stack Exchange.
 - [77] James R. Stieger, Stephen A. Engel, and Bin He. Continuous Sensorimotor Rhythm Based Brain Computer Interface Learning in a Large Population. *Scientific Data*, 8(1):98, 2021.
 - [78] Michael Tangermann, Klaus-Robert Müller, Ad Aertsen, Niels Birbaumer, Christoph Braun, Clemens Brunner, Robert Leeb, Carsten Mehring, Kai J. Miller, Gernot Mueller-Putz, Guido Nolte, Gert Pfurtscheller, Hubert Preissl, Gerwin Schalk, Alois Schlögl, Carmen Vidaurre, Stephan Waldert, and Benjamin Blankertz. Review of the bci competition iv. *Frontiers in Neuroscience*, Volume 6 - 2012, 2012.
 - [79] Marian Tietz, Thomas J. Fan, Daniel Nouri, Benjamin Bossan, and skorch Developers. *skorch: A scikit-learn compatible neural network library that wraps PyTorch*, 7 2017.
 - [80] Gael Varoquaux and Olivier Colliot. Evaluating machine learning models and their diagnostic value. In Olivier Colliot, editor, *Machine Learning for Brain Disorders*, pages 601–630. Springer US, New York, NY, 2023.
 - [81] Jonathan Wolpaw and Elizabeth Winter Wolpaw, editors. *Brain–Computer Interfaces: Principles and Practice*. Oxford University Press, 01 2012.
 - [82] Banghua Yang, Fenqi Rong, Yunlong Xie, Du Li, Jiayang Zhang, Fu Li, Guangming Shi, and Xiaorong Gao. A multi-day and high-quality EEG dataset for motor imagery brain-computer interface. *Scientific Data*, 12(1):488, 2025.

- [83] Weibo Yi, Shuang Qiu, Kun Wang, Hongzhi Qi, Lixin Zhang, Peng Zhou, Feng He, and Dong Ming. Evaluation of eeg oscillatory patterns and cognitive process during simple and compound limb motor imagery. *PLOS ONE*, 9(12):1–19, 12 2014.
- [84] Yu Zhang. Bayesian learning for eeg analysis. In Chang S. Nam, Anton Nijholt, and Fabien Lotte, editors, *Brain–Computer Interfaces Handbook: Technological and Theoretical Advances*, chapter 21, pages 407–424. CRC Press, 2018.
- [85] Qing Zhou, Jiafan Lin, Lin Yao, Yueming Wang, Yan Han, and Kedi Xu. Relative power correlates with the decoding performance of motor imagery both across time and subjects. *Frontiers in Human Neuroscience*, 15, 2021.

Appendix A

METHODS: DATASETS

A.1 *Electroencephalogram*

A.1.1 Basic Principles

The origin of electroencephalogram (EEG) signals comes from neurons creating electrical signals that are transmitted to other cells through synapses. An action potential (AP) arrives at a synaptic cleft (step 1 in Figure A.1) where it transmits chemical information through neurotransmitters (step 2) that generate postsynaptic potentials (PSPs) and local current (step 3). A PSP will create a current sink and will propagate to the cell body to generate a current source (step 4) [24].

As a result, the postsynaptic potential creates an electrical dipole consisting of a negative pole (the sink) and a positive pole (the source). The dipole generates primary (intracellular) currents and secondary (extracellular) currents. EEG signals result from postsynaptic potentials. More specifically, EEG signals result from the spatial and temporal summation of activity from a large population of synchronous neurons [24].

EEG signals correspond to a difference between electrical potentials, mainly due to extracellular currents. It is sensitive to both radial currents (actively located at the gyrus level) and to tangential currents (generated within sulci) even though it has strong sensitivity to radial currents. EEG, though, is also strongly attenuated and deformed by crossing through the skull [24].

There are two main types of electrophysiological activity of interest that are exploited for EEG: the evoked and oscillatory activity. Evoked responses are weak variations of electromagnetic activity resulting from a stimulation, such as in response to a task performed by the participant. There is a specific way to identify and describe evoked responses according

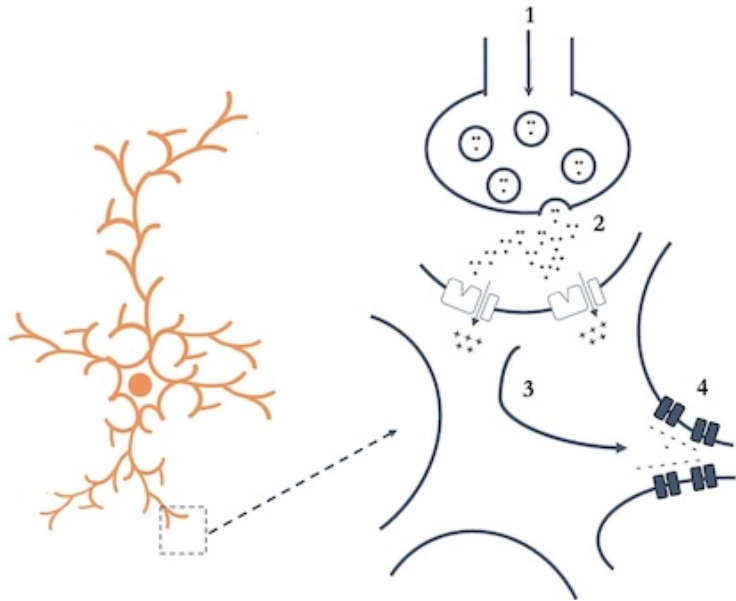


Figure A.1: Example of neural activity [24].

Table A.1: Characteristics of EEG [24].

EEG Characteristics	
Property	Value
Measurement	Difference of potentials, extracellular currents
Spatial resolution	2-3 cm
Temporal resolution	1 ms or less
Amplitudes	$\approx 100 \mu\text{Volts}$
Advantages	Portable, low cost
Drawbacks	Need of a reference, affected by bone, diffuse

to their latency, amplitude, shape, and polarity [24].

Figure A.2 first shows a positive deflection occurring 300 ms after the presentation of a stimulation. These waves of evoked activity can reflect different mechanisms: the early components are mostly exogenous and are related to stimulus characteristics, and late components are endogenous and are related to the performed task and to the subject's state. Alternatively, oscillatory activity, or induced activity, results from the summation of activity in a given brain region. These rhythms are mainly defined by their frequency, amplitude, shape, location, and duration [24].

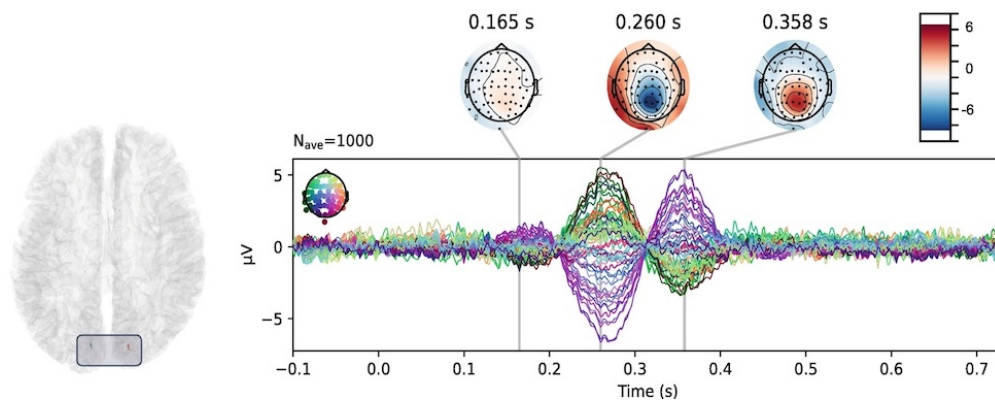


Figure A.2: P300 waves visualization [24].

Each frequency band is referred to by a Greek letter and corresponds to a subject's state. Delta (0.5–3 Hz) and Theta (3–7 Hz) rhythms are detected in deep and slight sleeps respectively. Alpha (8–12 Hz in posterior areas) and Mu (7–13 Hz in central areas) rhythms are observed in quiet watch and resting state (with eyes closed for Alpha). Beta (13–30 Hz) rhythm is detected during active watch and cognitive tasks such as motor imagery. Gamma rhythm (divided into two sub-rhythms: slow in 30–70 Hz and fast beyond 70 Hz) is observed during specific cognitive processing [24].

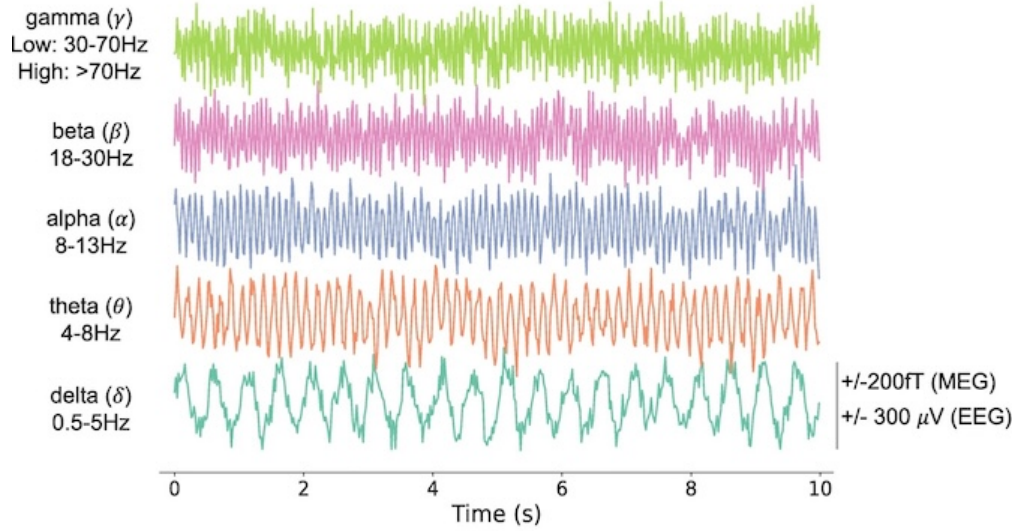


Figure A.3: Frequency bands characteristics [24].

A.1.2 Experiment Protocol

EEG signals are recorded through the use of electrodes placed over the scalp. The EEG relies on different types of electrodes: wet/dry and active/passive electrodes. Wet electrodes need an electrolytic gel to enable conduction between the skin and electrode. Dry electrodes behave as a conductor between the skin and electrode. Active electrodes contain an electronic module that performs a pre-amplification of the signal to ensure stability of the system from changes in impedance and noise. Passive electrodes do not use a pre-amplification module [24].

Even though some differences can be found from one EEG device to another, there are standardized ways to name and localize EEG sensors (also called channels). Each channel is referred to by a letter and number. Odd channels are located on the left hemisphere and even ones on the right hemisphere. The letters correspond to the area: frontal, temporal, parietal, central, and occipital. In addition to the sensors, one can also find landmarks: nasion, inion, and pre-auricular points [24].

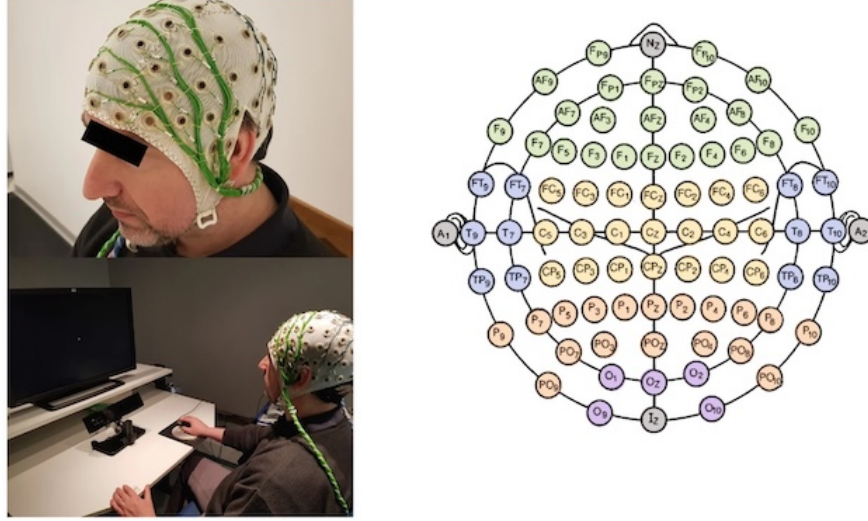


Figure A.4: Experiment protocol for EEG data acquisition [24].

Depending on the scientific question to be addressed and, therefore, the brain areas of interest, different montages can be found. One can build an EEG montage from less than 5 electrodes and up to 256 electrodes. EEG measurements rely on a difference of electrical potentials. For this purpose, two montages can be considered: the referential and bipolar montage [24].

In the referential montage each difference of electrical potentials considers an electrode placed over the scalp and a reference. The choice of the reference is crucial. The most commonly chosen locations for the reference electrodes are the mastoids (temporal bone behind the ears), though several studies prefer placing the reference at the vertex (Cz, midline of the scalp). The bipolar montage consists of performing the difference between two electrodes after the experiment. Another electrode referred to as the ground electrode is used. Among the chosen locations is the scapula (shoulder blade) [24].

There has been increased interest in developing wearable EEG, to remove wires and reduce its dimension but also to enable long lasting recordings in a less constrained environment. Three bottlenecks need to be overcome: the EEG electrodes, hard to put on and keep in

place on the head; the EEG hardware, to make it less power consuming and miniaturized; and the EEG software, to propose the most intelligible and reliable information regarding the captured brain activity [24].

For data acquisition, it will consist of cleaning the locations where electrodes will be in contact with the skin, like forehead and mastoids. The electrodes and EEG cap are then placed. Several key distances can be measured to verify the cap is well placed. Then, the experimenter needs to ensure that the communication between electrodes and scalp is established. For that purpose, an assessment of the impedance is made for each electrode. The lower it is, the better it is. Once impedance are lower than a certain threshold, typically a few kOhms, then the experiment can start [24].

Once the subject is correctly in place, the experimenter can start some pre-recordings to check quality of the signal and give specific instructions to the subjects accordingly, like loosening the jaw to avoid muscular artifacts. Finally, the experimenter can give further instructions regarding the task to perform before starting the recordings. After the session ends, the data is stored in specific servers to be processed [24].

A.1.3 Data Analysis

Artifacts are signals that make recording more difficult and may hamper the analysis of the brain activity recorded with EEG. Such artifacts can be divided into two categories: neurophysiological artifacts and environmental noise [24].

Neurophysiological artifacts correspond to noise generated by the subjects themselves, whether it is voluntary or not. The brain is far from being the only organ that generates electromagnetic activity. In particular, the eyes and heart produce electromagnetic activity. As a result, the main neurophysiological artifacts are related to cardiac activity and ocular activity (blinks and saccades), and can be visually seen during an EEG recording. A possible way to reduce ocular artifacts is to instruct the subject to avoid moving their eyes and, for short recordings only, to avoid eye blinking [24].

Another neurophysiological artifact, can be induced by the subject's voluntary motion.

Motion engenders muscular activity that can distort the recorded brain signals. Typical examples are jaw clenching and swallowing. A possible way to reduce the muscular artifacts is to instruct the subjects to remain as quiet as possible and to avoid moving their jaws [24].

Environmental noise refers to artifacts generated by the environment that surrounds the experimental setup. They can be linked with mechanical vibrations like the presence of a tramway nearby, or associated with power lines occurring at 50 or 60 Hz. Use of a Faraday cage (shielded room) may help to achieve the absolute best signal-to-noise ratio. Modern EEG equipment, especially systems with active electrodes, can also reduce noise. Additionally, system noise refers to artifacts generated by the sensors themselves that could be broken [24].

Preprocessing for artifact removal is probably the most crucial part when analyzing EEG data. The point here is to remove noise without eliminating information of interest of distorting the signal. The first thing to do is to extensively study the dataset, in particular, inspecting the EEG signals but also the associated broadband power spectra. This preliminary step enables identification of most artifacts and, most importantly, if they have a specific temporal and frequency signature (presence of periodic artifacts) [24].

From this point, it is possible to choose a specific strategy to remove the observed noise. In the case of cardiac and ocular artifacts, given their clear pattern, an efficient way to isolate and reduce them consists of applying independent component analysis (ICA). In the case of artifacts at a specific frequency, like power line noise at 50 or 60 Hz, one can consider applying notch filters. In the case of muscular activity, applying a low pass filter with a cutoff frequency at 40 Hz can be of interest [24].

It is possible to directly analyze the signals recorded by the sensors. In such a case, one will say that the analysis is performed in the space of the sensors. However, it is possible to go one step further and estimate the activity within the brain. This processing step is called source reconstruction and consists of estimating neural correlations of signals locations. It can be performed when one wants to have access to a higher spatial resolution and to provide a more accurate description, and interpretation, of the neurophysiological activity occurring.

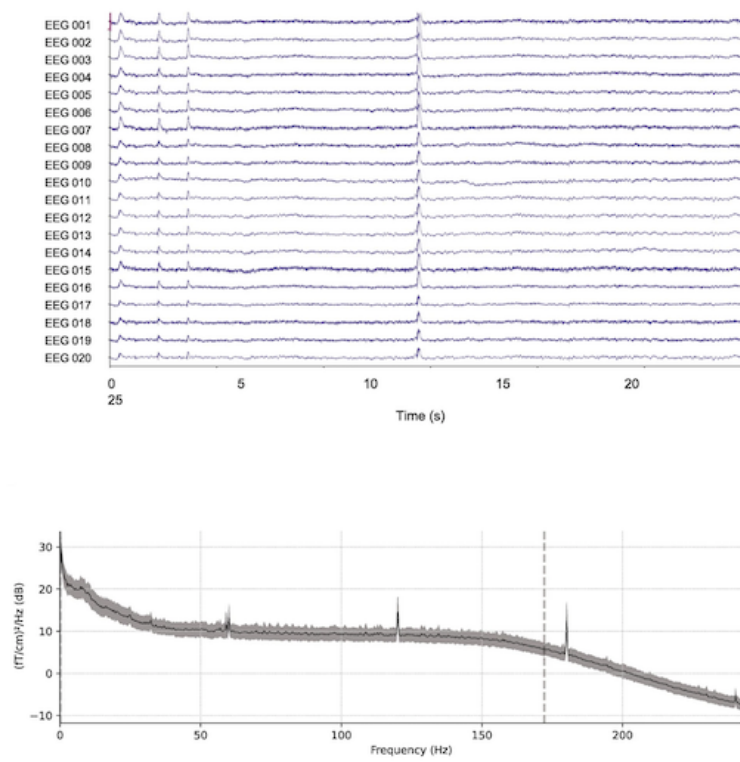


Figure A.5: Example of EEG signal artifacts [24].

For that purpose, both direct and inverse problems need to be solved [24].

For the direct problem, we aim at modeling the electromagnetic field produced. For that purpose, it is necessary to consider both a physical model and a model that will predict how the electromagnetic field will be generated at the scalp. The simplest model is the spherical model, which considers the head as an ensemble of spheres (brain, cerebrospinal fluid, skull, scalp) each characterized by a given conductivity. More realistic models rely on geometrical reconstruction of the different layers that form the head tissues, directly extracted from magnetic resonance imaging (MRI). They consist of building meshes of the interfaces between different tissues [24].

The purpose of the inverse problem is to reconstruct brain activity in the physical model. One of the main challenges is the nonuniqueness of its solution. In other words, a large number of brain activity patterns could generate the same signature detected at the sensor level. Therefore, some constraints or assumptions are essential to lead to a unique solution that best reflects the acquired data. Dipole modeling methods rely on modeling with a reduced number of equivalent dipoles and, therefore, are based on an a priori hypothesis of the required number of sources. Scanning methods consist of estimating the probability of presence of a current dipole inside each voxel [24].

A.1.4 Feature Engineering & Selection

The event related features consist of chunks of time series concatenated from all the channels, resulting from a low pass or band pass filtering and a down sampling step. This category of features is relevant when considering evoked activity after the presentation of a stimulus (visual, auditory, or sensory). They are therefore of interest when one is expecting significant changes in several amplitudes occurring at a given moment [24].

The spectral features are used in the case of detection for an oscillatory activity, when changes in EEG rhythms amplitudes are expected. The features are associated with the power spectra estimated in a given channel and in a given frequency band for a specific time window. Power spectra can be computed through many methods like, most notably,

spectrogram, Morlet wavelet scalogram, and autoregressive models [24].

Spatial filtering can be a valuable tool both for event related and spectral features. It relies on the combination of signals, recorded from different sensors, to obtain a new one, associated with an improved signal-to-noise ratio. Spatial filtering methods can be divided into three categories [24].

First, and not data driven, this category relies on physical considerations regarding the way the signals propagate through different brain tissues. The most famous illustration of this category is the Laplacian filter. In its simplest version, the small Laplacian consists of a derivation of the EEG waveform from the average signal computed with the four nearest neighbors for each electrode location [24].

Second, and data driven and unsupervised, this spatial filtering category can rely on a principal component analysis (PCA) approach. Third, and data driven and supervised, has its most famous examples in common spatial patterns (CSP) for spectral patterns and xDawn for event related features. CSP consists of a linear combination of EEG signals to maximize the difference between two classes in terms of variance. The xDawn approach aims at improving the signal-to-noise ratio through a projection of the raw EEG signals onto an estimated subspace of the evoked potentials [24].

Even though spectral and event related features are the most used in EEG literature, alternative features have been considered in past years. Firstly, features relying on covariance matrices have recently been used extensively, in particular for Riemannian geometry based classification. Despite an unclear neurophysiological interpretation, they have reached state of the art performance and won a large number of competitions. Secondly, new features, which take into account the interconnected nature of brain functioning and intensity of interactions, have emerged [24].

Estimating brain interactions in real time is not trivial: it consists of finding a compromise between ensuring the quasi-stationary nature of the signals and the statistical reliability of the functional connectivity estimation. Recent studies have considered the use of brain network metrics as potential features. These include local and global network metrics [24].

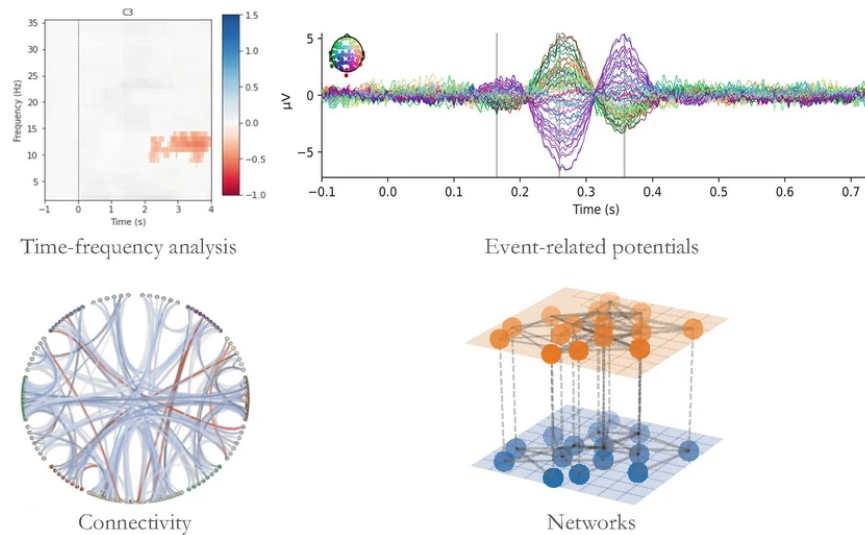


Figure A.6: Various contemporary forms of EEG data analysis [24].

At the local scale, the node degree counts the number of connections linking one node to another. In weighted networks, it is referred to as node strength and consists of summing the weights of the connections of the considered node. Another local scale property of interest is the betweenness centrality defined as the extent to which a node lies between other pairs of nodes. This metric enables the identification of the nodes that are crucial for information transfer between distant regions [24].

At the global scale, there is the characteristic path length and clustering coefficient. The characteristic path length indicates the global tendency of the nodes in the network to integrate and exchange information. The clustering coefficient measures the tendency of having nodes whose neighbors are mutually interconnected [24].

The feature selection is a crucial step as it prevents redundancy, ensures the reliability of the features, reduces dimensionality, and helps to provide interpretable results. It can be divided into three categories: embedded, filtered, and wrapper methods [24].

In filter methods, the feature selection is performed independently and before evaluation. For example, the R^2 score criterion can be used to assess to which extent a given feature is

influenced by a task performed by the subject. In wrapper methods, the feature selection utilizes the classification. In other words, in an iterative process, the relevance of each subset of features is assessed through classification performance until a given criterion is met. The embedded method consists of integrating both the feature selection and classification in the same process, for example, with a decision tree [24].

Appendix B

METHODS: PARADIGM

B.1 Metrics

B.1.1 Proper Scoring Rules

It is common to measure performance of a predictive model using a proper scoring rule, defined as follows. Let $S(p_{\boldsymbol{\theta}}, (y, \mathbf{x}))$ be the score for predictive distribution $p_{\boldsymbol{\theta}}(y|\mathbf{x})$ when given an even $y|\mathbf{x} \sim p^*(y|\mathbf{x})$, where p^* is the true conditional distribution. If we want to evaluate a Bayesian model, where we marginalize out $\boldsymbol{\theta}$ rather than condition on it, we just replace $p_{\boldsymbol{\theta}}(y|\mathbf{x})$ with $p(y|\mathbf{x}) = \int p_{\boldsymbol{\theta}}(y|\mathbf{x})p(\boldsymbol{\theta}|\mathcal{D})d\boldsymbol{\theta}$. The expected score is defined by the following. A proper scoring rule is one where $S(p_{\boldsymbol{\theta}}, p^*) \leq S(p^*, p^*)$, with equality if $p_{\boldsymbol{\theta}}(y|\mathbf{x}) = p^*(y|\mathbf{x})$. Thus, maximizing such a proper scoring rule will force the model to match the true probabilities [58].

Negative Log-Likelihood

The log-likelihood, $S(p_{\boldsymbol{\theta}}, (y, \mathbf{x})) = \log p_{\boldsymbol{\theta}}(y|\mathbf{x})$, is a proper scoring rule. Therefore, minimizing the negative log-likelihood (NLL) should result in well-calibrated probabilities. However, in practice, the NLL can over emphasize tail probabilities. The NLL is defined as follows where N is the number of samples [58].

$$\text{NLL} = -\frac{1}{N} \sum_{n=1}^N \log p_{\boldsymbol{\theta}}(y_n|\mathbf{x}_n).$$

Brier Score

A common alternative to NLL is the use of the Brier score. This is just the squared error of the predictive distribution $\mathbf{p} = p(1 : C|\mathbf{x})$ compared to the one-hot label distribution $\mathbf{y}$. Since it is based on squared error, the Brier score is less sensitive to extremely rare or extremely common classes. The Brier score is defined as follows for binary classification [58].

$$\text{BS} = \frac{1}{N} \sum_{n=1}^N (p_{\boldsymbol{\theta}}(y_n|\mathbf{x}_n) - \mathbf{1}(y_n))^2.$$

B.1.2 Discrimination

A machine learning model is validated by evaluating its prediction performance. For validation to provide an unbiased estimate of the model performance, data similar to that used for training is used for evaluation. The results of classification tasks can be summarized in the confusion matrix. For binary classification, the confusion matrix divides the test samples into four categories: true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) [80].

		Predicted	
		0	1
Actual	0	TN	FP
	1	FN	TP

		Predicted	
		0	1
Actual	0	TN/N = Spec	FP/N = Type I
	1	FN/P = Type II	TP/P = Sens

Figure B.1: (Left) The confusion matrix for binary classification. (Right) The confusion matrix normalized per row. Abbreviations: Spec = specificity, Sens = sensitivity.

Area Under the Receiver Operating Characteristic

Many classification algorithms output a continuous value that is thresholded to get a binary label. When the output is a probability the most common threshold used is 0.5. However, there are cases where it is interesting to study the performance for varying thresholds and the receiver operating characteristic (ROC) curve lets us do this [80]. The AUROC is a standard choice of discriminative summary metric for binary classification with balanced classes [57]. It has a probabilistic interpretation: The AUROC is the probability that a positive sample has a higher classification score than a negative sample [80].

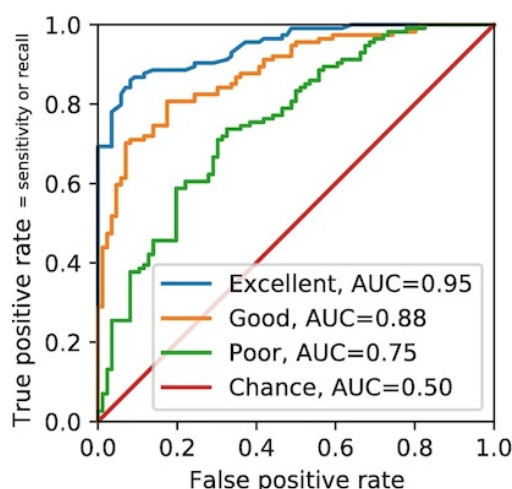


Figure B.2: The ROC curve plots the Sensitivity as a function of $1 - \text{Specificity}$ [80].

Matthews Correlation Coefficient

The Matthews correlation coefficient (MCC) makes full use of the confusion matrix. However, its interpretation may be less intuitive than that of other metrics [80]. It is generally regarded as a balanced measure which can be used even if the classes are of very different sizes. The MCC is in essence a correlation coefficient value between -1 and 1. A coefficient of 1 represents perfect prediction, 0 an average random prediction, and -1 an inverse prediction [63].

$$\text{MCC} = \frac{\text{TP} \times \text{TN} - \text{FP} \times \text{FN}}{\sqrt{(\text{TP} + \text{FP}) \times (\text{TP} + \text{FN}) \times (\text{TN} + \text{FP}) \times (\text{TN} + \text{FN})}}.$$

B.1.3 Calibration

A model whose predicted probabilities match the empirical frequencies is said to be calibrated. For example, if a classifier predicts $p(y = c \mid x) = 0.9$, then we expect this to be the true label 90% of the time. A well-calibrated model is useful to avoid making the wrong decision when the outcome is too uncertain. Bayesian approaches to fitting classifiers often result in more calibrated predictions, since they represent uncertainty in the parameters [58].

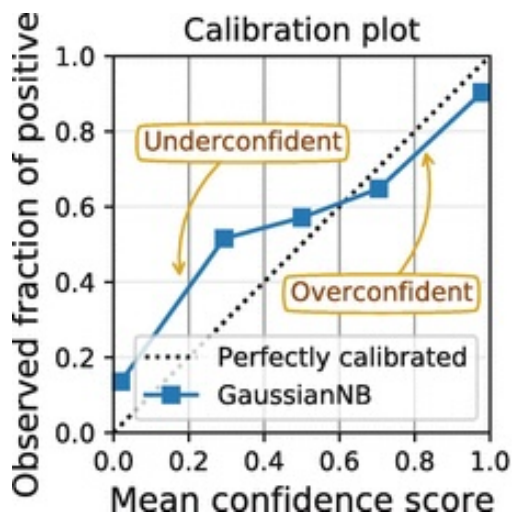


Figure B.3: Example calibration plot or reliability diagram for a Gaussian Naive Bayes classifier [80].

Expected Calibration Error

Reliability diagrams are useful visual tools, but it is more convenient to have a scalar summary statistic of calibration. One notion of miscalibration is the difference in expectation between confidence and accuracy. The expected calibration error (ECE) measures this by

partitioning predictions into M equally spaced bins (similar to reliability diagrams) and taking a weighted average of the accuracy and confidence difference of bins as follows, where n is the number samples. The difference between accuracy and confidence for a given bin represents the calibration gap between the solid and dotted lines in Figure B.3 [36].

$$\text{ECE} = \sum_{m=1}^M \frac{|B_m|}{n} \left| \text{acc}(B_m) - \text{conf}(B_m) \right|.$$

Maximum Calibration Error

In high-risk applications where reliable confidence measures are absolutely necessary, we may wish to minimize the worse case deviation between confidence and accuracy. The maximum calibration error (MCE) is the upper bound of this deviation. Similarly to ECE, this metric involves binning. In reliability diagrams, MCE measures the largest calibration gap across all bins, whereas ECE measures a weighted average of all gaps [36].

$$\text{MCE} = \max_{m \in \{1, \dots, M\}} \left| \text{acc}(B_m) - \text{conf}(B_m) \right|.$$

Appendix C

METHODS: PIPELINES

C.1 Markov Chain Monte Carlo

Almost as soon as computers were invented, they were used for simulation. Markov chain Monte Carlo (MCMC) was invented as Los Alamos, Metropolis et al (1953) simulated a liquid in equilibrium with its gas phase. Their tour de force was the realization that they did not need to simulate the exact dynamics, they only needed to simulate some Markov chain with the same equilibrium distribution. The Metropolis algorithm was widely used by chemists and physicists, but was not widely known among statisticians until after 1990. Hastings (1970) generalized the Metropolis algorithm, and simulations following his scheme are said to use the Metropolis-Hastings (MH) algorithm [15].

A special case of the MH algorithm was introduced by Geman et al (1984) discussing optimization to find the posterior mode rather than simulation. Algorithms following their scheme are said to use the Gibbs sampler. It took some time for the spatial statistics community to understand that the Gibbs sampler simulated the posterior distribution, thus enabling full Bayesian inference of all kinds. Gelfand et al (1990) made the wider Bayesian community aware of the Gibbs sampler, and then it was rapidly realized that most Bayesian inference could be done using MCMC, whereas very little could be done without MCMC. Green (1995) generalized the MH algorithm as much as it could be generalized [15].

C.1.1 Theoretical Foundations

A sequence $X_1, X_2, \dots$ of random elements of some set is a Markov chain if the conditional distribution of X_{n+1} given $X_1, \dots, X_n$ depends on X_n only. The set in which the X_i take values is called the state space of the Markov chain. A Markov chain has stationary transition

probabilities if the conditional distribution of X_{n+1} given X_n does not depend on n . This is the main kind of Markov chain of interest in MCMC. The joint distribution of a Markov chain is determined by the following [15].

- The marginal distribution of X_1 called the initial distribution, and
- The conditional distribution of X_{n+1} given X_n called the transition probability distribution.

If the state space is finite or countable, written $\{x_1, \dots, x_n\}$, then the initial distribution can be associated with a vector $\lambda = (\lambda_1, \dots, \lambda_n)$ defined by $P(X_1 = x_i) = \lambda_i$ for $i = 1, \dots, n$, and the transition probabilities can be associated with a matrix P having elements p_{ij} defined by $P(X_{n+1} = x_j \mid X_n = x_i)$ where $i = 1, \dots, n$ and $j = 1, \dots, n$. When the state space is uncountable, we must think of the initial distribution and transition probability distribution as unconditional and conditional probability distributions [15].

A stochastic process is stationary if for every positive integer k the distribution of the k -tuple $(X_{n+1}, \dots, X_{n+k})$ does not depend on n . An initial distribution is said to be stationary, invariant, or equilibrium for some transition probability distribution if the Markov chain specified by this initial distribution and transition probability distribution is stationary. Stationarity implies stationary transition probabilities, but not vice versa. The Metropolis-Hastings-Green (MHG) algorithm constructs a transition probability distribution that preserves a specified equilibrium distribution [15].

A transition probability distribution is reversible with respect to an initial distribution if for its Markov chain $X_1, X_2, \dots$, the distribution of pairs (X_i, X_{i+1}) is exchangeable. Reversibility implies stationarity, but not vice versa. A reversible Markov chain has the same laws running forward and backward. That is, for any i and k , the distributions $(X_{i+1}, \dots, X_{i+k})$ and $(X_{i+k}, \dots, X_{i+1})$ are the same. All known methods for constructing transition probabilities that preserve a specified equilibrium are special cases of the MHG algorithm, and all elementary updates from the MHG algorithm are reversible [15].

A bit of compute code that makes a pseudorandom change to its state is an update mechanism. An update mechanism is elementary if it is not made up of parts that are themselves update mechanisms preserving the specified distribution. Suppose the specified distribution (the desired stationary distribution of the MCMC sampler) has unnormalized density h . The Metropolis-Hastings update does the following [15]:

- When the current state is x , propose a move to y , having conditional probability density given x denoted $q(x, y)$.
- Calculate the Hastings ratio

$$r(x, y) = \frac{h(y)q(y, x)}{h(x)q(x, y)}. \quad (\text{C.1})$$

- Accept the proposed move y with probability

$$a(x, y) = \min(1, r(x, y)), \quad (\text{C.2})$$

that is, the state after the update is y with probability $a(x, y)$, and the state after the update is x with probability $1 - a(x, y)$.

For example, consider the probability density function $h(x)$ is given by $h(x) = g(x)/\int_{-\infty}^{\infty} g(u)du$ where we cannot or do not desire to solve the integral in the denominator analytically. Therefore, the distribution is only known up to some unknown constant: $h(x) \propto g(x)$. Notice that the ratio $h(y)/h(x)$ from $r(x, y)$ does not depend on the normalizing constant. The latter term $q(y, x)/q(x, y)$ corrects for biases from the proposal distribution. The special case of the Metropolis-Hastings algorithm where the proposal distribution is symmetric meaning $q(y, x) = q(x, y)$ is referred to as the Metropolis algorithm [15].

The Metropolis-Hastings update is reversible with respect to h , meaning that the transition probability that describes the update is an exact sampler of the specified distribution. If X_n is the current state and Y_n is the proposal, we have $X_n = X_{n+1}$ whenever the proposal is rejected. The distribution of (X_n, X_{n+1}) given rejection is exchangeable, and we must show

that (X_n, Y_n) is exchangeable given acceptance. That is, we must show that for any function f that has expectation, we can interchange the arguments of f [15].

$$\mathbb{E}[f(X_n, Y_n)a(X_n, Y_n)] = \iint f(x, y)h(x)a(x, y)q(x, y)dx dy = \mathbb{E}[f(Y_n, X_n)a(X_n, Y_n)].$$

This follows if we can interchange x and y in $h(x)a(x, y)q(x, y)$. Only the set of x and y such that $h(x) > 0$, $q(x, y) > 0$, and $a(x, y) > 0$ contribute to the integral or sum in the discrete case. These inequalities further imply that $h(y) > 0$ and $q(y, x) > 0$. Thus $r(y, x) = 1/r(x, y)$ for all x and y . Suppose that $r(x, y) \leq 1$, so $r(x, y) = a(x, y)$ and $a(y, x) = 1$. Then we have the following [15].

$$\begin{aligned} h(x)a(x, y)q(x, y) &= h(x)r(x, y)q(x, y) \\ &= h(y)q(y, x) \\ &= h(y)a(y, x)q(y, x). \end{aligned}$$

Conversely, suppose that $r(x, y) > 1$, so $a(x, y) = 1$ and $a(y, x) = r(y, x)$, and we have the following [15].

$$\begin{aligned} h(x)a(x, y)q(x, y) &= h(x)q(x, y) \\ &= h(y)r(y, x)q(y, x) \\ &= h(y)a(y, x)q(y, x). \end{aligned}$$

C.1.2 Hamiltonian Monte Carlo

Hamiltonian Monte Carlo (HMC) is a type of MCMC method that makes use of gradients to generate new proposed states. HMC attempts to avoid the random walk behavior typical of Metropolis-Hastings by using the gradient to propose new positions far from the current

one with high acceptance probability. This allows HMC to better scale to higher dimensions and in principle more complex geometries [54].

In physical systems, the Hamiltonian is a description of the total energy. The total energy can be decomposed into two terms, the kinetic and the potential energy. We can write the Hamiltonian of such a system as the following where $K(\mathbf{p}, \mathbf{q})$ is the kinetic energy and $V(\mathbf{q})$ is the potential energy [54].

$$H(\mathbf{q}, \mathbf{p}) = K(\mathbf{p}, \mathbf{q}) + V(\mathbf{q}). \quad (\text{C.3})$$

The probability of being at a particular position with a particular momentum is given as follows [54]. The exponential is a non-negative quantity and can therefore be viewed as an unnormalized probability distribution. The introduction of the minus sign in the exponential is simply a convention, meaning that higher values of energy correspond to lower values of probability [10].

$$p(\mathbf{q}, \mathbf{p}) = \exp(-H(\mathbf{q}, \mathbf{p})). \quad (\text{C.4})$$

To simulate such systems the Hamiltonian equations must be solve [54].

$$\frac{d\mathbf{q}}{dt} = \frac{\partial H}{\partial \mathbf{p}} = \frac{\partial K}{\partial \mathbf{p}} + \frac{\partial V}{\partial \mathbf{p}} \quad (\text{C.5})$$

$$\frac{d\mathbf{p}}{dt} = -\frac{\partial H}{\partial \mathbf{q}} = -\frac{\partial K}{\partial \mathbf{q}} - \frac{\partial V}{\partial \mathbf{q}}. \quad (\text{C.6})$$

The potential energy $V(\mathbf{q})$ is given by the probability we wish to sample from $p(\mathbf{q})$. For the momentum $K(\mathbf{p}, \mathbf{q})$ we can invoke an auxiliary variable, and by choosing $p(\mathbf{p}|\mathbf{q})$ we can write $p(\mathbf{q}, \mathbf{p}) = p(\mathbf{p}|\mathbf{q})p(\mathbf{q})$. This ensures we can recover the target distribution by marginalizing out the momentum. By introducing the auxiliary variable we can rewrite Equation C.3 [54].

$$H(\mathbf{q}, \mathbf{p}) = -\log p(\mathbf{p}|\mathbf{q}) - \log p(\mathbf{q}). \quad (\text{C.7})$$

We are free to choose the kinetic energy, and if we choose it to be Gaussian and drop the normalization constant, then we have the following where $\mathbf{M}$ is the precision matrix [54].

$$K(\mathbf{p}, \mathbf{q}) = \frac{1}{2} \mathbf{p}^\top \mathbf{M}^{-1} \mathbf{p} + \log |\mathbf{M}|. \quad (\text{C.8})$$

If we choose $\mathbf{M}$ to be the identity matrix we have $K(\mathbf{p}, \mathbf{q}) = 1/2 \mathbf{p}^\top \mathbf{p}$, making $\partial K / \partial \mathbf{p} = \mathbf{p}$ and $\partial K / \partial \mathbf{q} = 0$, and then we can simplify the Hamiltonian equations [54].

$$\frac{d\mathbf{q}}{dt} = \mathbf{p} \quad (\text{C.9})$$

$$\frac{d\mathbf{p}}{dt} = -\frac{\partial V}{\partial \mathbf{q}}. \quad (\text{C.10})$$

Therefore, the HMC algorithm can be summarized as follows [54].

1. Sample $\mathbf{p} \sim \mathcal{N}(0, \mathbf{I})$
2. Simulate $\mathbf{q}_t$ and $\mathbf{p}_t$ for some amount of time T
3. $\mathbf{q}_T$ is the new proposed state
4. Use the Metropolis acceptance criterion to accept or reject $\mathbf{q}_T$.

In an idealized model, energy is perfectly conserved. However, we use the Metropolis acceptance criterion to correct for errors introduced by the numerical simulation of the Hamiltonian equations. To compute the Hamiltonian equations we have to compute a trajectory. That is, all the intermediate points between one state and the next. In practice this involves computing a series of small integration steps. The most popular method is the leapfrog integrator. Leapfrog integration updates positions $\mathbf{q}_t$ and momentums $\mathbf{p}_t$ at interleaved time points, staggered in such that they leapfrog over each other [54].

An efficient HMC run requires proper tuning of its hyperparameters [54]:

- The time discretization (step size of the leapfrog)

- The integration time (number of leapfrog steps)
- The precision matrix $\mathbf{M}$ that parametrizes the kinetic energy.

For example, if the step size is too large then the leapfrog integrator will be inaccurate and too many proposals will be rejected. However, if it is too small we waste computation resources. If the number of steps is too small, the simulated trajectory at each iteration will be too short and sampling will fall back to random walk. But if too large the trajectory might run in circles and we again waste computation resources. If the estimated covariance (inverse of the precision matrix) is too different from the posterior covariance, the proposal momentum will be suboptimal and movement in the position space will be too large or too small in some dimension [54].

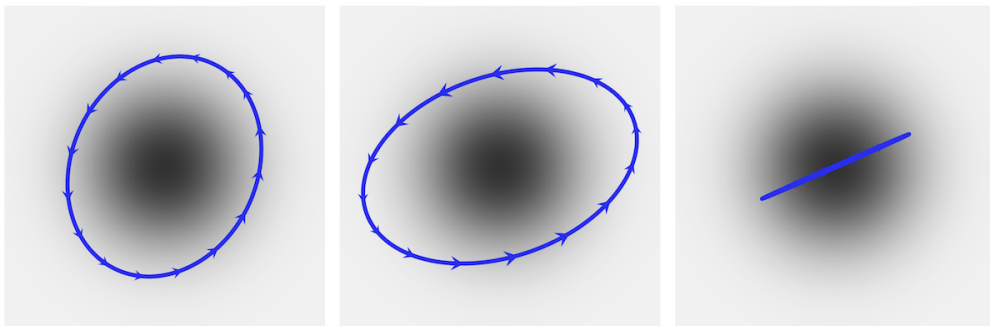


Figure C.1: Example of the circular leapfrog. Three different trajectories around the same 2D normal distribution. For practical sampling we want to move as far as possible from the starting point while avoiding U-turns in the trajectory. Hence, the name of one of the most popular dynamic HMC methods No U-Turn Sampling (NUTS) [54].

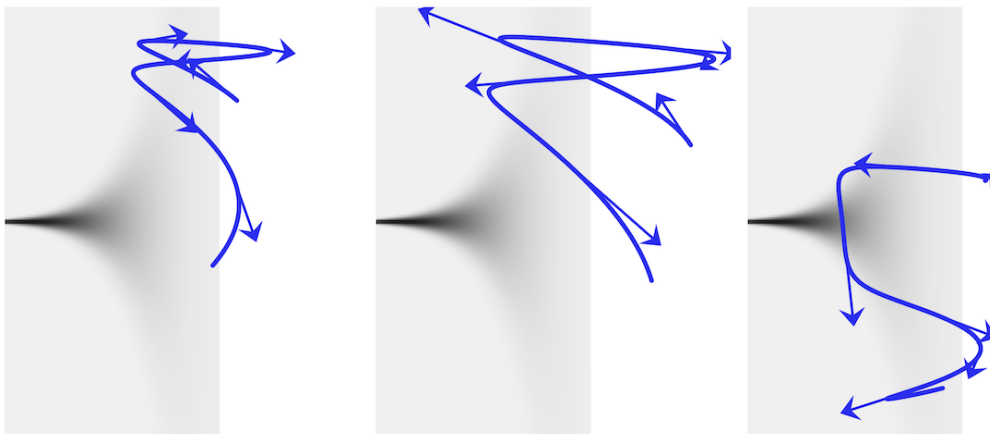


Figure C.2: Example of the funnel leapfrog. Three different trajectories around the same Neal’s funnel, a common geometry arising in (centered) hierarchical models. These trajectories fail to properly simulate following the correct distribution resulting in divergences. When the exact trajectories lie on regions of high curvature, the numerical trajectories generated can rapidly get off towards the boundaries of the distribution we are trying to explore [54].

C.1.3 Runtime Diagnostics

The following numerical and visual diagnostic tools are illustrated with three synthetic posteriors. The first named `good_chains` was generated from independent and identically distributed (IID) draws from a $\text{Beta}(2, 5)$. Samples that are IID are ideal for approximating the posterior [54].

The second is `bad_chains0` generated by sorting `good_chains` and then adding Gaussian noise. These values are highly autocorrelated (not independent) and not identically distributed since the flattened and sorted array is being reshaped into a 2D array. The third synthetic posterior called `bad_chains1` is generated from `good_chains` by randomly introducing portions where consecutive samples are highly correlated to each other [54].

Effective Sample Size

When using MCMC sampling methods we may wonder if a particular sample is large enough to confidently compute quantities of interest like a mean or variance. This cannot be answered directly because samples from MCMC methods have some degree of autocorrelation. The amount of information contained in autocorrelated samples will be less than what we would get from an IID sample of the same size. The effective sample size (ESS) is an estimator that takes autocorrelation into account and provides the number of draws we would have if our sample was IID [54].

Table C.1: The ESS when the count of actual samples in our synthetic posteriors is 4000 [54].

ESS		
good_chains	bad_chains0	bad_chains1
4.389e+03	2.436	111.1

Convergence of Markov chains is not uniform across parameter space. It is easier to get a good approximation from the bulk of a distribution than from the tails simply because the tails are dominated by rare events. As a general rule of thumb, an ESS greater than 400 is recommended, otherwise the estimation is unreliable [54].

Potential Scale Reduction Factor $\hat{R}$

We need ways to estimate convergence for finite samples. One pervasive idea is to run more than one chain starting from very different points and then check the resulting chains to see if they look similar to each other. This intuitive notion can be formalized into a numerical diagnostic known as $\hat{R}$ [54].

The overall idea is that the $\hat{R}$ for the parameter θ is the standard deviation of all the

samples of θ , that is including all chains together, divided by the root mean square of the separated within chain standard deviations. Ideally we should get a value of 1, as the variance between chains should be the same as the variance within chains. From a practical point of view, values of $\hat{R} \lesssim 1.01$ are considered safe [54].

Table C.2: The $\hat{R}$ from our synthetic posteriors [54].

$\hat{R}$		
good_chains	bad_chains0	bad_chains1
1.000	2.408	1.033

Monte Carlo Standard Error

Using MCMC methods introduces an additional layer of uncertainty as we are approximating the posterior with a finite number of samples. We can estimate the amount of error introduced using the Monte Carlo Standard Error (MCSE), which is based on Markov chain central limit theorem. We should check the MCSE only once we are sure ESS is high enough and $\hat{R}$ is low enough, otherwise MCSE is of no use. As with the ESS the MCSE varies across the parameter space and we may want to evaluate it for different regions like specific quantiles [54].

Table C.3: The MCSE of our example posteriors [54].

MCSE		
good_chains	bad_chains0	bad_chains1
0.002381	0.1077	0.01781

Trace Plots

Trace plots are the most common plots in Bayesian literature. They are the first plot after inference for visual inspection. A trace plot is made by drawing the sampled values at each iteration step. From these plots we are able to see if different chains converged to the same distribution, getting a sense of the degree of autocorrelation [54].

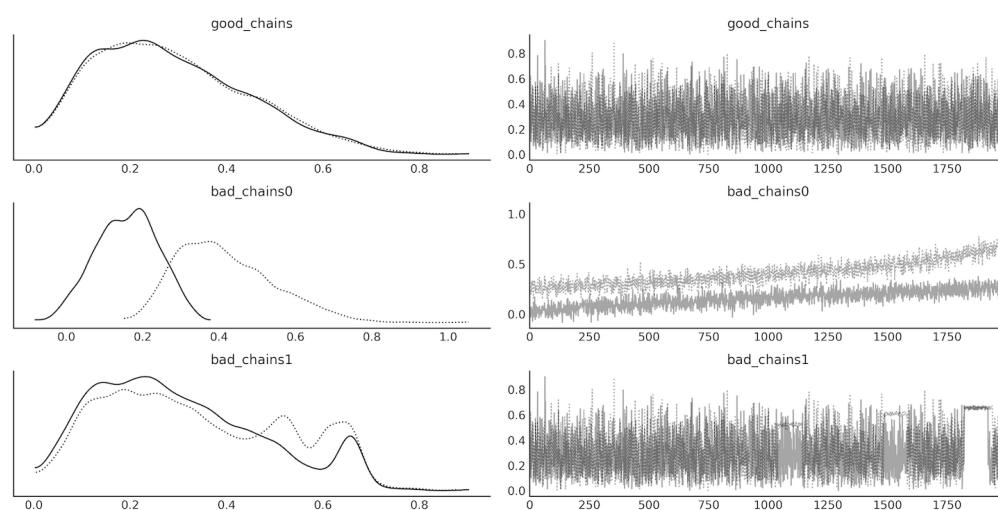


Figure C.3: (Left) One kernel density estimation (KDE) per chain. (Right) The sampled values per chain per step. Notice the fuzzy caterpillar appearance in `good_chains` versus the irregularities in the other two [54].

Autocorrelation Plots

As seen when discussing effective sample size, autocorrelation decreases the amount of information contained in a sample and thus is something we want to keep to a minimum. An autocorrelation plot lets us qualitatively inspect the results from ESS. Figure C.4 is the bar plot of the autocorrelation function, where the gray band represents the 95% confidence interval [54].

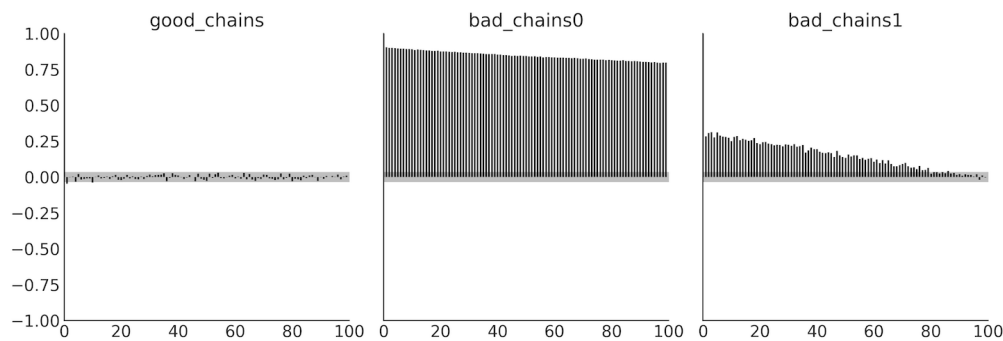


Figure C.4: The autocorrelation plots of our posterior samples [54].

Rank Plots

Rank plots are histograms of the ranked samples. The ranks are computed by first combining all chains but then plotting the results separately for each chain. If all the chains are targeting the same distribution, we expect the ranks to have a uniform distribution. Additionally, if rank plots of all chains look similar, this indicates good mixing of the chains. Rank plots can be more sensitive than trace plots, are thus recommended after refinement of sampling and modeling definitions. Trace plots are especially useful during early stages when we need to explore many different alternatives [54].

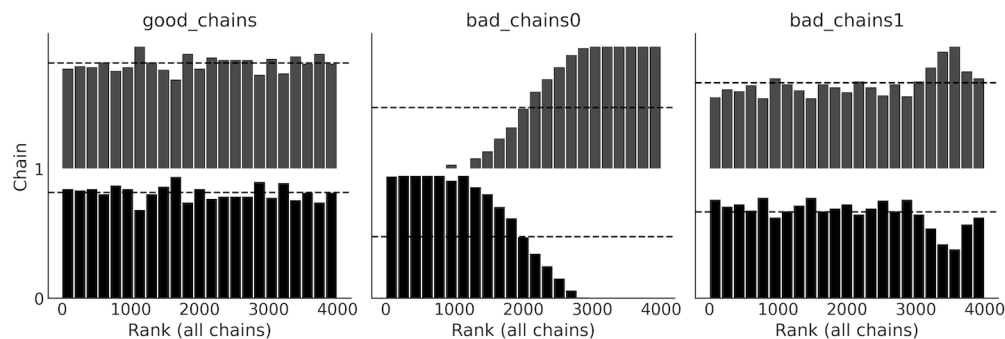


Figure C.5: Rank plots compare the height of the bar to the dashed line representing a uniform distribution. Ideally, the bars should follow a uniform distribution [54].

Divergences

The above diagnostics are summaries of the generated samples. Another way to perform a diagnostic is through monitoring the sampling method. One prominent example are the divergences present in Hamiltonian Monte Carlo (HMC) methods. Divergences can be visualized in trace plots as vertical bars at the bottom of the KDEs indicating where something went wrong during sampling [54].

C.1.4 Non-centered Parameterization

For relatively simple models, samplers mostly just work, providing us with an accurate estimate of the posterior. However, certain posterior geometries are challenging for samplers. A common example is Neal’s funnel. The shape at the top of the funnel is quite wide before narrowing into a small neck. Samplers function by taking steps from one set of parameter values to another, and a key setting is how large of a step to take when exploring the posterior surface. In complex geometries, such as with Neal’s funnel, a step size that works well in one area fails in another [54].

In hierarchical models the posterior geometry is largely defined by the correlation of hyperpriors to other parameters, which can result in funnel geometry that are difficult to sample. These inconvenient posteriors can create inefficient MCMC caused by steep curvature in the negative log-likelihood and lead to divergent transitions. Transforming the prior to be smoother helps. There is a relatively simple tweak to models, referred to as a non-centered parameterization, that helps alleviate the issue [54].

Example: Devil’s Funnel prior

$$\begin{aligned}v &\sim \text{Normal}(0, \sigma_v) \\x &\sim \text{Normal}(0, \exp(v)).\end{aligned}$$

As σ_v increases, a trough forms in the prior probability surface known as centered parameterization that is difficult for Hamiltonian dynamics to sample [55, 75].

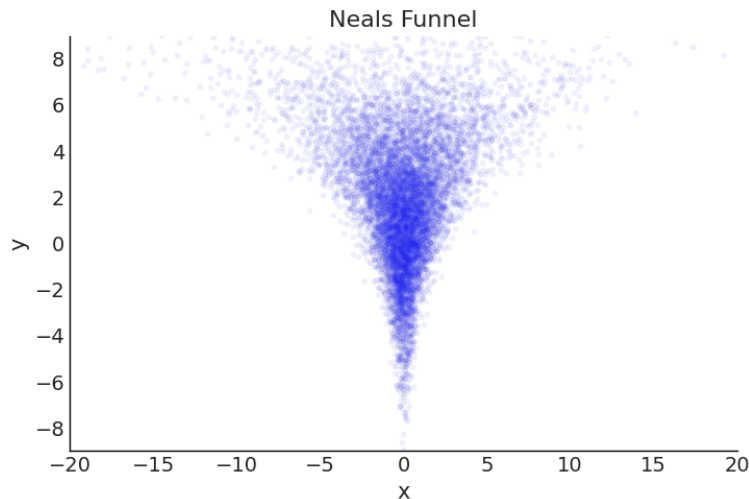


Figure C.6: An example of Neal’s Funnel created from correlated samples. At the top of the funnel, where y is between 6 to 8, a step size of say 1 unit is likely to stay in a dense region of the posterior. However, when sampling at the bottom of the posterior, around y between -6 to -8, a step size of 1 unit in almost any direction will likely step into a low-density region. For HMC samplers, the occurrence of divergences can diagnose these sampling issues [54].

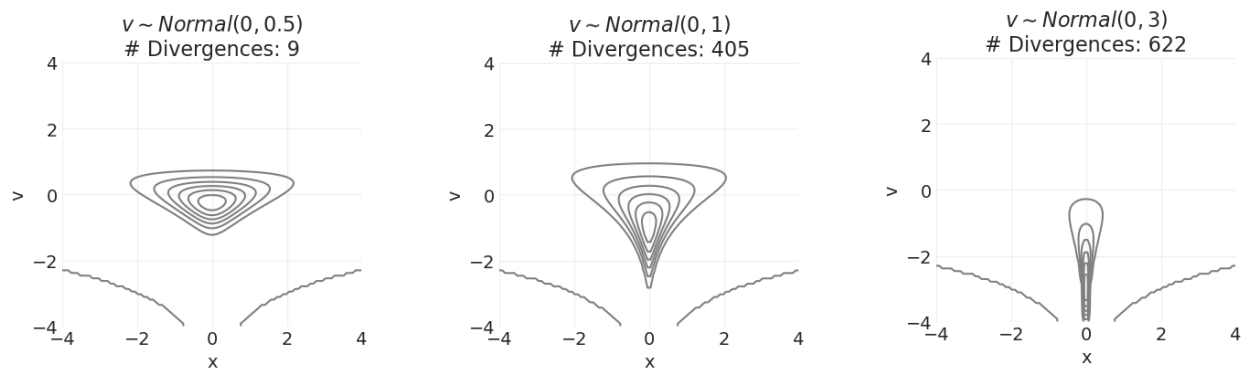


Figure C.7: Centered parameterization [55, 75].

To handle this, we could configure a smaller steps size to manage the steepness better, but then sampling would take longer to explore the posterior. Alternatively, we can re-parameterize (transform) to make the surface smoother known non-centered parameterization. For this purpose, we add an auxiliary variable z that has a smooth probability surface. We then sample that auxiliary variable, and transform it to obtain the target variable distribution. For the case of the devil’s funnel prior we do the following [55, 75].

$$v \sim \text{Normal}(0, \sigma_v)$$

$$z \sim \text{Normal}(0, 1)$$

$$x = z \exp(v).$$

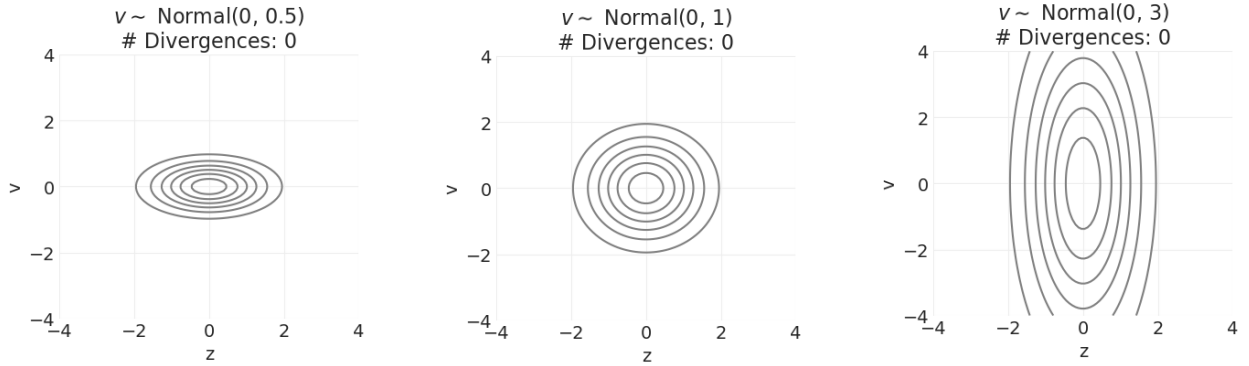


Figure C.8: Non-centered parameterization [55, 75].

We can also transform sampling using Cholesky factors. The Cholesky factor $\mathbf{L}$ provides an efficient means for encoding a correlation matrix $\mathbf{\Omega}$ since it requires fewer floating point numbers than the full correlation matrix. It is defined by $\mathbf{\Omega} = \mathbf{L}\mathbf{L}^\top$ where $\mathbf{L}$ is a lower-triangular matrix. We can sample data with correlation $\mathbf{\Omega}$, and standard deviation σ_i , using the Cholesky factorization $\mathbf{L}$ of $\mathbf{\Omega}$ as follows where $\mathbf{Z}$ is a matrix of z -scores sampled from a standard normal distribution [55, 75].

$$\mathbf{X}_\Omega \sim \text{diag}(\sigma_i) \mathbf{L} \mathbf{Z}.$$

C.2 Common Spatial Pattern

C.2.1 Correctness Proof

The common spatial pattern (CSP) allows one to maximize the variance of signals from one condition and at the same time minimize the variance of signals from another condition [11]. For example, consider two three-dimensional tensors (trials $\times$ channels $\times$ time) of electroencephalogram (EEG) from separate classes $\mathbf{X}_1, \mathbf{X}_2 \in \mathbb{R}^{p \times q \times r}$. For each trial let us assume zero mean and compute between channel across time sample covariance matrices (SCMs) $\Sigma_i^{(k)} = \frac{1}{r-1} \mathbf{X}_i^{(k)} \mathbf{X}_i^{(k)\top} \in \mathbb{R}^{q \times q}$ for $i = 1, 2$ and $k = 1, \dots, p$. Furthermore, let us take $\bar{\Sigma}_i \in \mathbb{R}^{q \times q}$ as the arithmetic mean SCM for each class across all trials k .

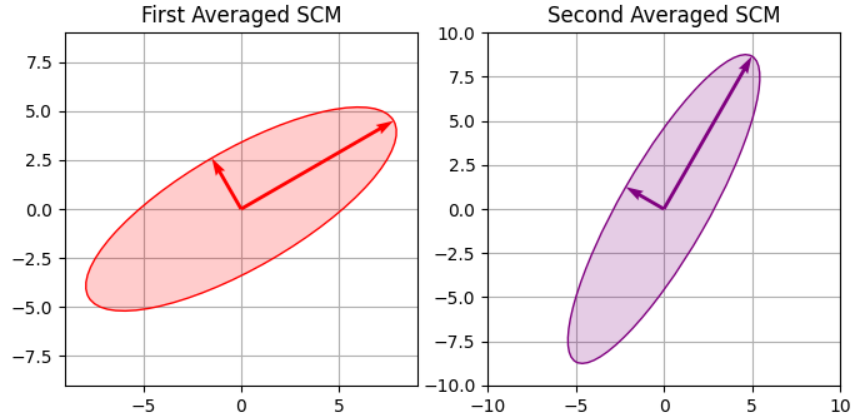


Figure C.9: Example of averaged sample covariance matrices (SCMs).

Our example is a standard data and preprocessing pipeline for CSP used by software libraries like PyRiemann [6]. We assume the SCMs are conditioned so that they are symmetric positive definite (SPD). By definition, for a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and any non-zero

vector $\mathbf{x} \in \mathbb{R}^n$, if $\mathbf{A} = \mathbf{A}^\top$ and $\mathbf{x}^\top \mathbf{A} \mathbf{x} > 0$, then $\mathbf{A}$ is SPD. As a result, all eigenvalues are positive $\lambda_i > 0$ from the eigendecomposition $\mathbf{A} = \mathbf{Q} \mathbf{D} \mathbf{Q}^\top$ where $\mathbf{D} = \text{diag}(\lambda_1, \dots, \lambda_n)$ [57]. Furthermore, the sum of two SPD matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times n}$ is also SPD since $\mathbf{x}^\top (\mathbf{A} + \mathbf{B}) \mathbf{x} = \mathbf{x}^\top \mathbf{A} \mathbf{x} + \mathbf{x}^\top \mathbf{B} \mathbf{x} > 0$. Let us diagonalize the mean SCMs from our example.

$$\bar{\Sigma}_1 + \bar{\Sigma}_2 = \mathbf{V} \mathbf{\Lambda} \mathbf{V}^\top. \quad (\text{C.11})$$

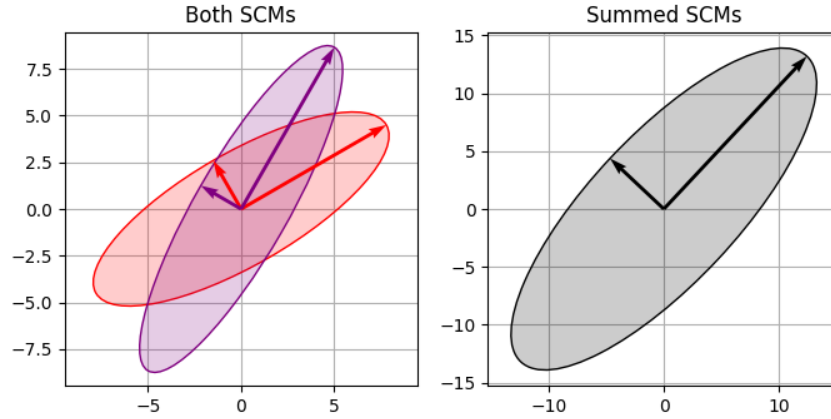


Figure C.10: The averaged SCMs summed.

The whitening transformation of an SCM results in zero mean, unit variance, and zero covariance [11]. In other words, the whitening transformation maps an SCM to the identity matrix $\Sigma \mapsto \mathbf{I}$. We define the whitening matrix as $\mathbf{W} = \mathbf{V} \mathbf{\Lambda}^{-1/2} \mathbf{V}^\top \in \mathbb{R}^{n \times n}$. The result is no longer an ellipse when plotted but rather the unit circle.

$$\mathbf{W}(\bar{\Sigma}_1 + \bar{\Sigma}_2) \mathbf{W}^\top = \mathbf{V} \mathbf{\Lambda}^{-1/2} \mathbf{V}^\top (\mathbf{V} \mathbf{\Lambda} \mathbf{V}^\top) \mathbf{V} \mathbf{\Lambda}^{-1/2} \mathbf{V}^\top = \mathbf{I}. \quad (\text{C.12})$$

We can rewrite our whitening transformation to show that the variance of the signals in our first class $\bar{\Sigma}_1$ is maximized, while the variance of the signals in our second class $\bar{\Sigma}_2$ is also minimized. We do this by distributing the whitening matrix throughout the sum of our

SCMs. The result is two transformed SCMs we denote by $\bar{\Sigma}'_1, \bar{\Sigma}'_2 \in \mathbb{R}^{n \times n}$ whose sum is the identity matrix.

$$\mathbf{W}(\bar{\Sigma}_1 + \bar{\Sigma}_2)\mathbf{W}^\top = \mathbf{W}\bar{\Sigma}_1\mathbf{W}^\top + \mathbf{W}\bar{\Sigma}_2\mathbf{W}^\top = \bar{\Sigma}'_1 + \bar{\Sigma}'_2 = \mathbf{I}. \quad (\text{C.13})$$

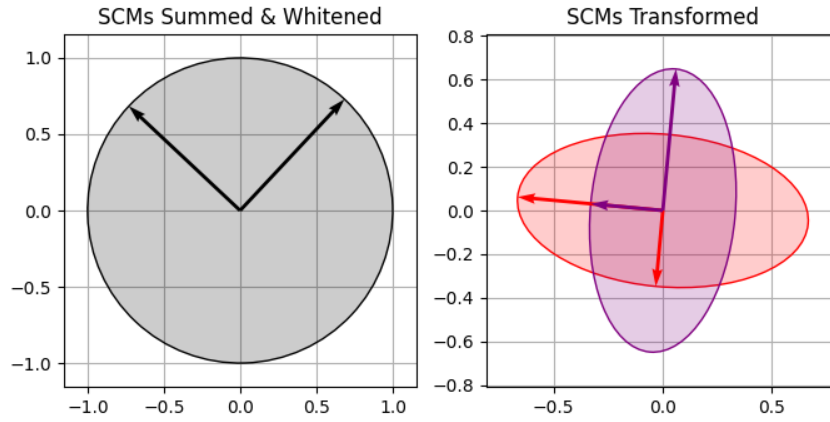


Figure C.11: Both the whitened sum of the SCMs and the sum of each SCM transformed by the whitening matrix equals the identity matrix.

Let us use $\mathbf{V}'_i \mathbf{\Lambda}'_i \mathbf{V}'_i{}^\top$ where $i = 1, 2$ to denote the diagonalization of our transformed SCMs. Since $\bar{\Sigma}'_1 + \bar{\Sigma}'_2 = \mathbf{I}$ the sum of the diagonals $d_{ij} = \text{diag}(\mathbf{\Lambda}'_i)_j$ is $d_{1j} + d_{2j} = 1$. Furthermore, $d_{ij} > 0$ since $\bar{\Sigma}'_i$ is SPD. Therefore, the diagonals are bounded between 0 and 1. When the variance $d_{aj} \gg d_{bj}$ for $a \neq b$, the classification problem is more or less solved for unseen EEG trials. However, when d_{aj} is close to d_{bj} the discrimination between classes is more ambiguous [11].

C.3 Tangent Space

Current state-of-the-art (SOTA) machine learning (ML) for electroencephalogram (EEG) uses across channel covariance matrices and Riemannian geometric statistics on the sym-

metric positive definite (SPD) manifold for classification [7, 22]. These methods either discriminate directly on the SPD manifold or in the vector space that is tangent to the manifold. In this section we analyze the latter method of discrimination that performs classification in the tangent space (TS) of the SPD manifold.

C.3.1 Definitions

A manifold is a collection of points that locally, but not globally, resembles Euclidean space. A Riemannian metric is defined by a smoothly varying collection of scalar products $\langle \cdot, \cdot \rangle_x$ in each tangent space $T_x\mathcal{M}$ at points x on a manifold $\mathcal{M}$. For example, the inner product gives a norm $\| \cdot \|_x: T_x\mathcal{M} \rightarrow \mathbb{R}$ by $\|v\|_x^2 = \langle v, v \rangle_x$. The shortest distance between two points on a manifold is a geodesic $\gamma(t)$, and it is computed by integrating the norm along its curve [74].

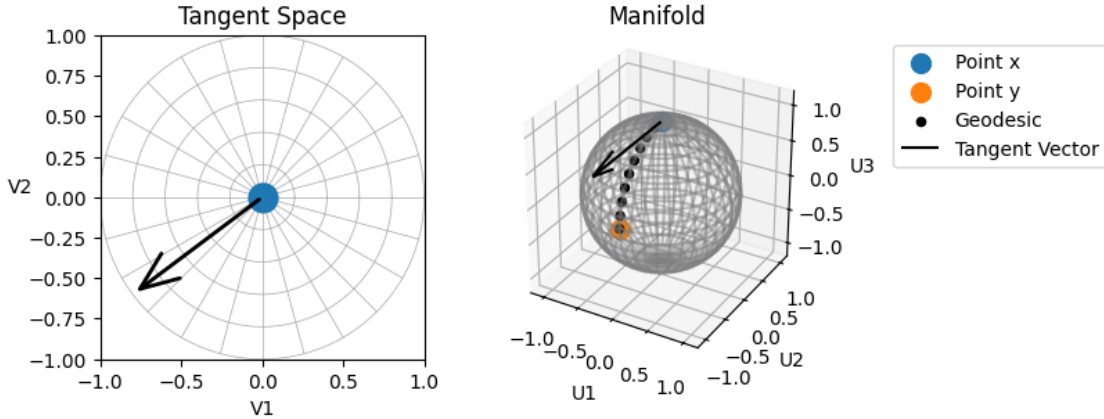


Figure C.12: Example of a tangent space, manifold, and geodesic. (Left) The tangent space $T_x\mathcal{M}$ that is a vector space at some point x on a manifold $\mathcal{M}$. (Right) The curved manifold $\mathcal{M}$ with geodesic $\gamma(t)$ from the point x to some other point y on the manifold.

We can map vectors in the tangent space to the manifold using geodesics. The vector $v \in T_x\mathcal{M}$ can be mapped to the point of the manifold that is reached after a unit time $t = 1$ by the geodesic $\gamma(t)$ starting at x with initial velocity $\gamma'(0) = v$. This mapping $\text{Exp}_x: T_x\mathcal{M} \rightarrow \mathcal{M}$ is called the exponential map where $\text{Exp}_x(v) = \gamma(1)$. The inverse of the

exponential map is the logarithmic map $\text{Log}_x(y)$ and is the smallest vector v as measured by the Riemannian metric so that $\text{Exp}_x(v) = y$ [74].

In general, finding geodesics involves solving second-order ordinary differential equations (ODEs) [74]. However, SPD matrices have additional Lie group structure that can be used to simplify algorithms and speed up computations of Riemannian metrics and geodesics [64]. A group is a set G with an associative binary operation satisfying $(xy)z = x(yz)$ for all $x, y, z \in G$, contains an identity element $e \in G$ so that $ex = xe = x$ for all $x \in G$, and contains an inverse element $x^{-1} \in G$ for each $x \in G$ where $x^{-1}x = xx^{-1} = e$. A Lie group is a smooth manifold and also a group [74].

For each x, y in a Lie group G , left-translations by y are denoted $L_y: x \mapsto yx$. The differential $(L_y)_*$ of left-translation maps the tangent space $T_x G$ to the tangent space $T_{yx} G$. In particular, $(L_y)_*$ maps any vector $u \in T_e G$ to the vector $(L_y)_* u \in T_y G$. The vector field $\tilde{u}(y) = (L_y)_* u$ is called left-invariant since it is invariant under left-translations $\tilde{u} \circ L_y = (L_y)_* \tilde{u}$ for all $y \in G$ [74]. In other words, pullback to the identity and pushforward to another object are commutative under left-translation.

$$\begin{aligned}
 (L_y)_* \tilde{u}(x) &= (L_y)_* (L_x)_* u \\
 &= (L_{yx})_* u \\
 &= \tilde{u}(yx) \\
 &= \tilde{u}(L_y x) \\
 &= (\tilde{u} \circ L_y) x.
 \end{aligned}
 \qquad
 \begin{array}{ccc}
 u & \xrightarrow{(L_y)_*} & (L_y)_* u \\
 \tilde{u} \uparrow & & \uparrow \tilde{u} \\
 e & \xrightarrow{L_y} & y
 \end{array}$$

The left-translation maps give a useful way of defining Riemannian metrics on Lie groups. By definition the tangent space $T_e G$ of a Lie group G at the identity element, typically denoted $\mathfrak{g}$, is called a Lie algebra. Given an inner product $\langle \cdot, \cdot \rangle_{\mathfrak{g}}$ on the Lie algebra, we can extend it to an inner product on tangent spaces at all elements of the group by setting $\langle u, v \rangle_g = \langle (L_{g^{-1}})_* u, (L_{g^{-1}})_* v \rangle_{\mathfrak{g}}$. This defines a left-invariant Riemannian metric on G since $\langle (L_h)_* u, (L_h)_* v \rangle_{hg} = \langle u, v \rangle_g$ for any $u, v \in T_g G$ [74].

$$\begin{aligned}
\langle (L_h)_* u, (L_h)_* v \rangle_{hg} &= \langle (L_{(hg)^{-1}})_* (L_h)_* u, (L_{(hg)^{-1}})_* (L_h)_* v \rangle_{\mathfrak{g}} \\
&= \langle (L_{(hg)^{-1}h})_* u, (L_{(hg)^{-1}h})_* v \rangle_{\mathfrak{g}} \\
&= \langle (L_{g^{-1}h^{-1}h})_* u, (L_{g^{-1}h^{-1}h})_* v \rangle_{\mathfrak{g}} \\
&= \langle (L_{g^{-1}})_* u, (L_{g^{-1}})_* v \rangle_{\mathfrak{g}} \\
&= \langle u, v \rangle_g.
\end{aligned}$$

$$\begin{array}{ccc}
& \langle \cdot, \cdot \rangle_g & \\
(L_g)_* \uparrow & \searrow (L_h)_* & \\
& \langle \cdot, \cdot \rangle_{\mathfrak{g}} & \xrightarrow{(L_{hg})_*} \langle \cdot, \cdot \rangle_{hg}
\end{array}$$

For a matrix $P \in \mathbb{R}^{n \times n}$ and any non-zero vector $x \in \mathbb{R}^n$, if $P = P^\top$ and $x^\top P x > 0$, then P is an SPD matrix [57]. The space of SPD matrices is a smooth manifold, and not a vector space since it lacks an additive identity, additive inverses, and zero and negative real scalar multiplication [64]. The SPD manifold Sym_n^+ of all SPD matrices is a Lie group with matrix multiplication as its group operation [74]. Therefore, we can derive invariant metrics and geodesics on the SPD manifold.

Let A be a matrix from the general linear group $\text{GL}(n)$ of non-singular matrices. Then $APA^\top \in \text{Sym}_n^+$ since $(APA^\top)^\top = (A^\top)^\top P A^\top = APA^\top$ and for any non-zero vector $x \in \mathbb{R}^n$ then $x^\top (APA^\top) x = (x^\top A) P (A^\top x) = (A^\top x)^\top P (A^\top x) > 0$. Given two matrices $P, Q \in \text{Sym}_n^+$ we can derive the distance between them as a norm from the identity by choosing $A = P^{-1/2}$ [64]. That is, with left-invariance we can pullback to the identity before pushforward.

$$\text{dist}(P, Q) = \text{dist}(\text{Id}, P^{-1/2} Q P^{-1/2}) = N(P^{-1/2} Q P^{-1/2}). \quad (\text{C.14})$$

By left-invariance at the identity of a Lie group we have the vector field $\tilde{u}(x) = (L_x)_* u$. The geodesic starting at the identity e with initial velocity u satisfies $x(0) = e$ and $x'(t) = \tilde{u}(x(t))$. For matrix groups, this ODE becomes $x'(t) = x(t)u$ and is uniquely solved by the matrix exponential $x(t) = \exp(tu)$. In other words, the exponential map is the matrix exponential and its inverse the logarithmic map is the matrix logarithm [74].

From Equation C.14 take $R = P^{-1/2} Q P^{-1/2}$. The matrix logarithm $\log(R)$ produces the vector for the norm $N(R)$. This norm is given by the Frobenius norm $\|\cdot\|_F$. Furthermore, since R is SPD, it has eigendecomposition $R = V \Lambda V^\top$ where $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$ and

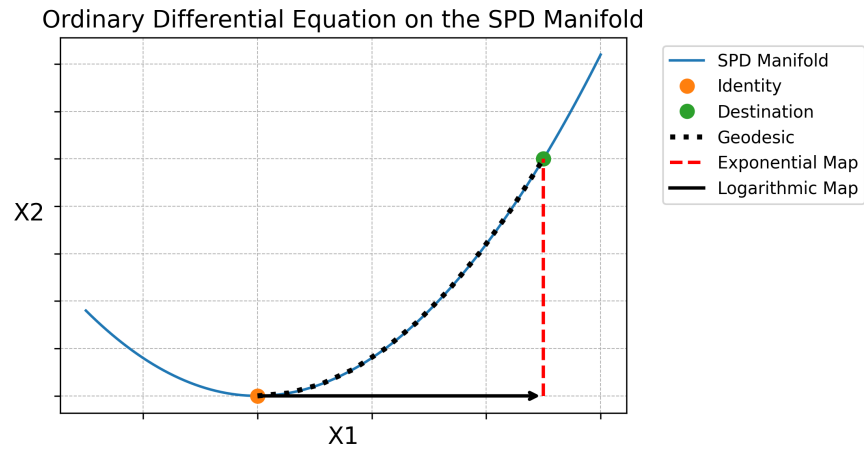


Figure C.13: The exponential map $\text{Exp}_x(v) = \gamma(1) = y$ produces the point on the manifold $\mathcal{M}$ reached after a unit time $t = 1$ along the geodesic $\gamma(t)$ starting at point x with initial velocity $\gamma'(0) = v$. The logarithmic map $\text{Log}_x(y) = v$ is its inverse and produces the initial velocity needed to reach y from x after the unit time $t = 1$. On Sym_n^+ the exponential map is the matrix exponential and the logarithmic map is the matrix logarithm.

$\lambda_i > 0$. By the matrix logarithm $\log(R) = V \text{diag}(\log \lambda_1, \dots, \log \lambda_n) V^\top$. Therefore, by the orthogonal invariance of the Frobenius norm, the norm $N(R)$ can be written as the square root of the sum of squared eigenvalue logarithms [64].

$$\text{dist}(P, Q) = N(P^{-1/2} Q P^{-1/2}) = \|\log(R)\|_F = \left(\sum_{i=1}^n \log^2 \lambda_i \right)^{1/2}. \quad (\text{C.15})$$

Equation C.15 gives a closed form solution for the distance between two SPD matrices on Sym_n^+ . Not only is it exact and so does not require optimization, but it can be computed in the vector space of the tangent space and so does not require integration on the curve. Furthermore, by the nature of logarithms we see that matrices with zero or negative eigenvalue are in fact infinite distance from SPD matrices on Sym_n^+ , contrary to Euclidean space and metrics [64].

We can also find closed form solutions for the logarithmic and exponential maps on Sym_n^+ . Rather than solve for the norm of the logarithmic map, we can solve for the vector itself. This is done by pushing forward to P after we pulled back to the identity from P . Furthermore, as the inverse of the logarithmic map, the closed form solution for the exponential map takes a vector as input and outputs the point on Sym_n^+ reached after unit time elapsed along the geodesic.

$$\text{Log}_P(Q) = P^{1/2} \log(P^{-1/2} Q P^{-1/2}) P^{1/2}. \quad (\text{C.16})$$

$$\text{Exp}_P(W) = P^{1/2} \exp(P^{-1/2} W P^{-1/2}) P^{1/2}. \quad (\text{C.17})$$

At the foundation of statistics is the notion of a distance. The closed form solution of the mean in Euclidean space $\frac{1}{N} \sum_{i=1}^N X_i$ relies on the additive structure of vector space and does not generalize to Riemannian space. However, there are defining properties of the mean that do generalize: The geometric mean is a least-squares centroid that minimizes the sum-of-squared distances. A natural strategy for computing the geometric mean is gradient descent optimization [30].

$$\bar{y} = \arg \min_{y \in \mathcal{M}} \sum_{i=1}^N \text{dist}(y, y_i)^2. \quad (\text{C.18})$$

The exponential and logarithmic map, distance, and mean on Sym_n^+ are common Riemannian geometric statistics used by SOTA ML for EEG that takes across channel sample covariance matrices for classification [7, 22]. The minimum distance to the mean (MDM) classifier is trained by memorizing the mean of each class and is tested by predicting classes based on the minimum distance to the memorized means [7]. All operations occur on the SPD manifold. In the next section we describe in more detail an alternative algorithm that performs classification in the tangent space of Sym_n^+ .

C.3.2 Example

Consider two three-dimensional tensors of EEG recordings $X_1, X_2 \in \mathbb{R}^{p \times q \times r}$ from separate classes where the dimensions p, q, r are the trials, channels, and time, respectively. For each trial, let sample covariance matrices (SCMs) be computed between channels across time so that $\Sigma_1^{(k)}, \Sigma_2^{(k)} \in \mathbb{R}^{q \times q}$ for $k = 1, \dots, p$. Assume each SCM is conditioned so that it is SPD. Furthermore, let us define $\bar{\Sigma}_1, \bar{\Sigma}_2 \in \mathbb{R}^{q \times q}$ as geometric mean SCMs averaged across all trials k of each class.

With the geometric mean of each class $\bar{\Sigma}_1, \bar{\Sigma}_2 \in \text{Sym}_n^+$ we define $\bar{\Sigma} \in \text{Sym}_n^+$ as the geometric mean of these means. Then, using the logarithmic map we find the vectors between all points in all clusters to $\bar{\Sigma}$. We represent these vectors in the tangent space $T_{\bar{\Sigma}}\text{Sym}_n^+$ of the mean point $\bar{\Sigma}$. This linearization of SPD matrices is done in a way that respects the metric space of Sym_n^+ . Once in the tangent space $T_{\bar{\Sigma}}\text{Sym}_n^+$ we can train standard ML algorithms that are optimized for vector space.

Once in the tangent space $T_{\bar{\Sigma}}\text{Sym}_n^+$ we can train standard ML algorithms that are optimized for vector space. This method of linearization from the space of SPD matrices to vector space is the standard used by software libraries like PyRiemann [6]. As of now, SOTA EEG classifiers are those that learn and predict in the tangent space of the SPD manifold

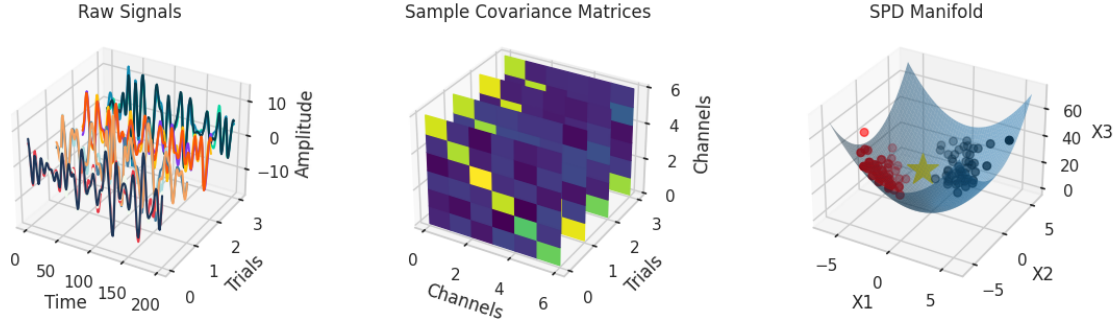


Figure C.14: Example of our data preprocessing pipeline. (Left) We receive a three-dimensional tensor ($\text{trials} \times \text{channels} \times \text{time}$) of raw EEG signals recorded from a brain-computer interface session. (Center) We transform each trial of signals into between channel sample covariance matrices (SCMs). (Right) We represent SCMs on the SPD manifold where they are naturally clustered by their label and the golden star is the geometric mean whose tangent space we map all SCMs to for ML in vector space.

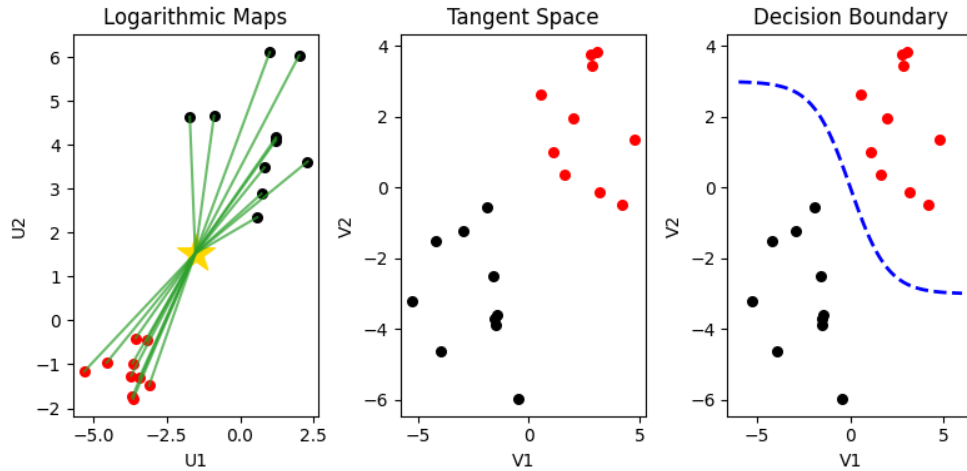


Figure C.15: Example of our data preprocessing and classification pipeline. (Left) All SPD matrices are vectorized by the logarithmic map. (Center) The vectorized SPD matrices in tangent space. (Right) A learned decision boundary from a standard ML algorithm.

rather than on the manifold itself [7, 22]. An explanation for this is that machine learning is traditionally optimized for vector space.

C.4 Linear Discriminant Analysis

C.4.1 Decision Boundary Derivation

Decision theory for classification tells us that we need to know the class posteriors $P(G|X)$ for optimal classification. Suppose $f_k(x)$ is the conditional density of X for class $G = k$, and let π_k be the prior probability of class k , with $\sum_{\ell=1}^K \pi_\ell = 1$. Bayes theorem gives us the following [40].

$$P(G = k|X = x) = \frac{f_k(x)\pi_k}{\sum_{\ell=1}^K f_\ell(x)\pi_\ell}. \quad (\text{C.19})$$

Many machine learning techniques are based on models for the class densities. For example, linear and quadratic discriminant analysis use Gaussian densities. More flexible Gaussian mixture models (GMMs) allow for nonlinear decision boundaries. General non-parametric density estimates for each class density allow the most flexibility. Naive Bayes models are a variant of the previous case, and assume that the inputs are conditionally independent in each class [40].

Suppose that each class density is modeled as a multivariate Gaussian [40].

$$f_k(x) = (2\pi)^{-p/2} |\Sigma_k|^{-1/2} \exp \left(-\frac{1}{2} (x - \mu_k)^\top \Sigma_k^{-1} (x - \mu_k) \right). \quad (\text{C.20})$$

Linear discriminant analysis (LDA) is the special case when we assume that the classes have a common covariance matrix $\Sigma_k = \Sigma \forall k$. When we compare the log-ratio of two classes k and ℓ we get an equation linear in x . The equal covariance matrices cause the normalization factors to cancel, as well as the quadratic part in the exponents. This linear log-odds function implies that the decision boundary between classes k and ℓ is linear in x [40].

$$\begin{aligned}
\log \frac{P(G = k|X = x)}{P(G = \ell|X = x)} &= \log \frac{f_k(x)}{f_\ell(x)} + \log \frac{\pi_k}{\pi_\ell} \\
&= \log \frac{\pi_k}{\pi_\ell} - \frac{1}{2}(\mu_k + \mu_\ell)^\top \Sigma^{-1}(\mu_k - \mu_\ell) \\
&\quad + x^\top \Sigma^{-1}(\mu_k - \mu_\ell).
\end{aligned} \tag{C.21}$$

The linear discriminant functions are an equivalent description of the decision rule, with $G(x) = \operatorname{argmax}_k \delta_k(x)$ [40].

$$\delta_k(x) = x^\top \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^\top \Sigma^{-1} \mu_k + \log \pi_k. \tag{C.22}$$

In practice we do not know the parameters of the Gaussian distributions, and need to estimate them using training data [40].

- $\hat{\pi}_k = N_k/N$, where N_k is the number of observations from class k ,
- $\hat{\mu}_k = \sum_{g_i=k} x_i / N_k$,
- $\hat{\Sigma} = \sum_{k=1}^K \sum_{g_i=k} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^\top / (N - K)$.

In the general discriminant problem, if the Σ_k are not assumed to be equal, then the convenient cancellations in Equation C.21 do not take place. In particular the pieces quadratic in x remain. We then get quadratic discriminant analysis (QDA). The decision boundary between each pair of classes k and ℓ is described by a quadratic equation $\{x : \delta_k(x) = \delta_\ell(x)\}$ [40].

$$\delta_k(x) = -\frac{1}{2} \log |\Sigma_k| - \frac{1}{2} (x - \mu_k)^\top \Sigma_k^{-1} (x - \mu_k) + \log \pi_k. \tag{C.23}$$

Both LDA and QDA perform well on an amazingly large and diverse set of classification tasks. The question is why LDA and QDA have such a good track record. The reason is likely not that the data are approximately Gaussian, and in addition for LDA that the covariances

are approximately equal. More likely the reason is that the data can only support simple decision boundaries such as linear or quadratic, and the estimates provided by the Gaussian models are stable. This is a bias variance tradeoff where we put up with the bias of a linear decision boundary because it can be estimated with much lower variance than more exotic alternatives [40].

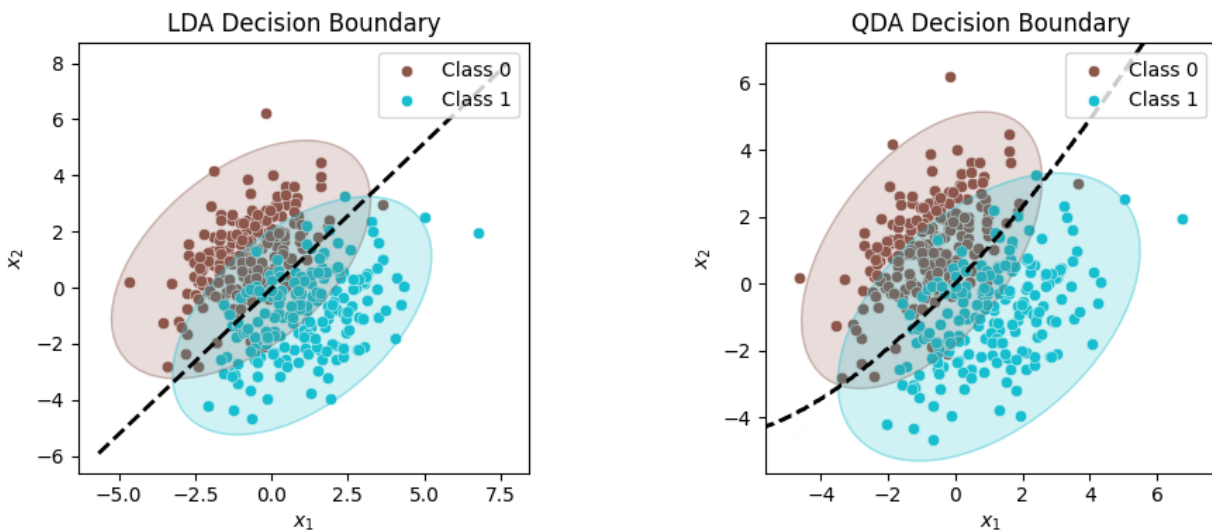


Figure C.16: Example of LDA and QDA decision boundaries from a toy dataset.

C.5 Logistic Regression

C.5.1 Expressiveness Delineation

The logistic regression model arises from the desire to model the posterior probabilities of K classes via linear function in x , while at the same time ensuring that they sum to one and remain in $[0, 1]$. The model has the following form. It is specified in terms of $K - 1$ log-odds or logit transformations, reflecting the constraint that the probabilities sum to one. Although the model uses the last class as the denominator in the odds-ratios, the choice of denominator is arbitrary in that the estimates are equivalent under this choice [40].

$$\begin{aligned}
\log \frac{P(G=1|X=x)}{P(G=K|X=x)} &= \beta_{10} + \beta_1^\top x \\
\log \frac{P(G=2|X=x)}{P(G=K|X=x)} &= \beta_{20} + \beta_1^\top x \\
&\vdots \\
\log \frac{P(G=K-1|X=x)}{P(G=K|X=x)} &= \beta_{(K-1)0} + \beta_{K-1}^\top x.
\end{aligned} \tag{C.24}$$

A reformulation shows the following [40].

$$\begin{aligned}
P(G=k|X=x) &= \frac{\exp(\beta_{k0} + \beta_k^\top x)}{1 + \sum_{\ell=1}^{K-1} \exp(\beta_{\ell 0} + \beta_\ell^\top x)}, \quad k=1, \dots, K-1, \\
P(G=K|X=x) &= \frac{1}{1 + \sum_{\ell=1}^{K-1} \exp(\beta_{\ell 0} + \beta_\ell^\top x)}.
\end{aligned} \tag{C.25}$$

To emphasize the dependence on the entire parameter set $\theta = \{\beta_{10}, \beta_1^\top, \dots, \beta_{(K-1)0}, \beta_{K-1}^\top\}$, we can denote the probabilities $P(G=k|X=x) = p_k(x; \theta)$. When $K=2$ the model is especially simple since there is only a single linear function. Logistic regression models are usually fit by maximum likelihood, using the conditional likelihood of G given X . Since $P(G|X)$ completely specifies the conditional distribution, the multinomial distribution is appropriate. The log-likelihood for N observations is as follows where $p_k(x_i; \theta) = P(G=k|X=x_i; \theta)$ [40].

$$\ell(\theta) = \sum_{i=1}^N \log p_{g_i}(x_i; \theta). \tag{C.26}$$

It is convenient to code the two-class g_i via a 0/1 response y_i , where $y_i = 1$ when $g_i = 1$, and $y_i = 0$ when $g_i = 2$. Let $p_1(x; \theta) = p(x; \theta)$, and $p_2(x; \theta) = 1 - p(x; \theta)$. The log-likelihood can be written as the following where $\beta = \beta_{10}, \beta_1$, and we assume that the vector of inputs x_i includes the constant term 1 to accommodate the intercept [40].

$$\begin{aligned}
\ell(\beta) &= \sum_{i=1}^N \left\{ y_i \log p(x_i; \beta) + (1 - y_i) \log(1 - p(x_i; \beta)) \right\} \\
&= \sum_{i=1}^N \left\{ y_i \beta^\top x_i - \log(1 + e^{\beta^\top x_i}) \right\}.
\end{aligned} \tag{C.27}$$

To maximize the log-likelihood, we set its derivatives to zero, where these score equations are nonlinear in β [40].

$$\frac{\partial \ell(\beta)}{\partial \beta} = \sum_{i=1}^N x_i (y_i - p(x_i; \beta)) = 0. \tag{C.28}$$

In section C.4 we find that the log-posterior odds of linear discriminant analysis (LDA) between two classes k and K are linear functions of x , where this linearity is a consequence of the Gaussian assumption for the class densities, as well as the assumption of a common covariance matrix [40].

$$\begin{aligned}
\log \frac{P(G = k|X = x)}{P(G = K|X = x)} &= \log \frac{\pi_k}{\pi_K} - \frac{1}{2}(\mu_k + \mu_K)^\top \Sigma^{-1}(\mu_k - \mu_K) \\
&\quad + x^\top \Sigma^{-1}(\mu_k - \mu_K) \\
&= \alpha_{k0} + \alpha_k^\top x.
\end{aligned} \tag{C.29}$$

The linear logistic model by construction has linear logits [40].

$$\log \frac{P(G = k|X = x)}{P(G = K|X = x)} = \beta_{k0} + \beta_k^\top x. \tag{C.30}$$

It seems that the models are the same. Although they have exactly the same form, the difference lies in the way the linear coefficients are estimated. The logistic regression model is more general, in that it makes less assumptions. We can write the joint density of X and G as follows where $P(X)$ denotes the marginal density of the inputs X [40].

$$P(X, G = k) = P(X)P(G = k|X). \tag{C.31}$$

For both LDA and logistic regression, the second term on the right has the following logit-linear form where the last class has again been arbitrarily chosen as the reference [40].

$$P(G = k|X = x) = \frac{\exp(\beta_{k0} + \beta_k^\top x)}{1 + \sum_{\ell=1}^{K-1} \exp(\beta_{\ell 0} + \beta_\ell^\top x)}. \quad (\text{C.32})$$

The logistic regression model leaves the marginal density of X as an arbitrary density function $P(X)$, and fits the parameters of $P(G|X)$ by maximizing the conditional likelihood. That is, the multinomial likelihood with probabilities $P(G = k|X)$. Although $P(X)$ is totally ignored, we can think of this marginal density as being estimated nonparametrically, using the empirical distribution function which places mass $1/N$ at each observation [40].

With LDA we fit the parameters by maximizing the full log-likelihood based on the joint density as follows where ϕ is the Gaussian density function [40].

$$P(X, G = k) = \phi(X; \mu_k, \Sigma) \pi_k. \quad (\text{C.33})$$

Standard normal theory leads to the estimates $\hat{\mu}_k$, $\hat{\Sigma}$, and $\hat{\pi}_k$ given in section C.4. Since the linear parameters of the logistic form are functions of the Gaussian parameters, we get their maximum-likelihood estimates by plugging in the corresponding estimates. However, unlike in the conditional case, the marginal density $P(X)$ does play a role. It is a mixture density which also involves the parameters [40].

$$P(X) = \sum_{k=1}^K \pi_k \phi(X; \mu_k, \Sigma). \quad (\text{C.34})$$

The question is what role can this additional restriction play. By relying on the additional model assumptions, we have more information about the parameters, and hence can estimate them more efficiently with lower variance. For example, observations far from the decision boundary which are down-weighted by logistic regression play a role in estimating the common covariance matrix. This means that LDA is not robust to gross outliers [40].

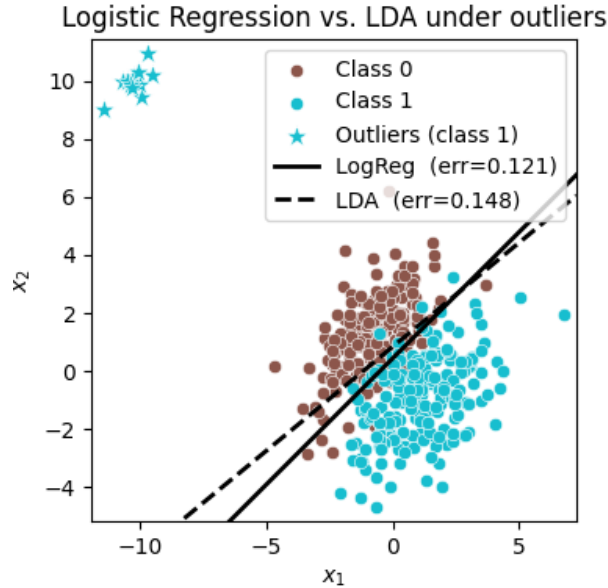


Figure C.17: Logistic regression vs. LDA decision boundaries for a toy dataset shown with error from the accuracy score.

C.6 Support Vector Machine

The support vector machine (SVM) solves the binary classification task. It is a supervised learning task where we have a set of examples $\mathbf{x}_n \in \mathbb{R}^D$ along with their corresponding binary labels $y_n \in \{1, -1\}$. Given a training set $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$ we would like to estimate parameters of the model that will give the smallest classification error. In the case of binary classification, we divide the vector space $\mathbb{R}^D$ into two parts corresponding to the positive and negative classes. In the following we consider a particularly convenient partition, which is to linearly split the space using a hyperplane [27].

C.6.1 Separating Hyperplane

Let example $\mathbf{x} \in \mathbb{R}^D$ be an element of the data space, and consider a function parameterized by $\mathbf{w} \in \mathbb{R}^D$ and $b \in \mathbb{R}$ [27].

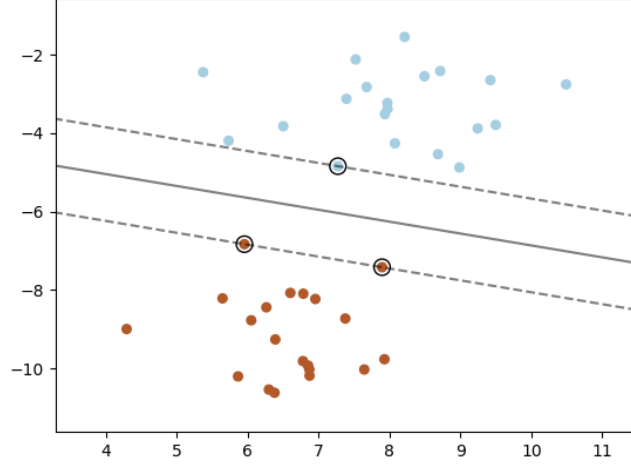


Figure C.18: Example of an SVM linearly separating two classes of data points by a margin [63].

$$f : \mathbb{R}^D \rightarrow \mathbb{R} \quad (\text{C.35})$$

$$\mathbf{x} \mapsto f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b. \quad (\text{C.36})$$

We define the hyperplane that separates the two classes in our binary classification problem as follows where the vector $\mathbf{w}$ is normal to the hyperplane and b is the intercept [27].

$$\{\mathbf{x} \in \mathbb{R}^D : f(\mathbf{x}) = 0\}. \quad (\text{C.37})$$

We can derive that $\mathbf{w}$ is a normal vector to the hyperplane by choosing any two examples $\mathbf{x}_p$ and $\mathbf{x}_q$ on the hyperplane and showing that the vector between them is orthogonal to $\mathbf{w}$. Since we have chosen $\mathbf{x}_p$ and $\mathbf{x}_q$ to be on the hyperplane, this implies that $f(\mathbf{x}_p) = 0$ and $f(\mathbf{x}_q) = 0$ and hence $\langle \mathbf{w}, \mathbf{x}_p - \mathbf{x}_q \rangle = 0$. Therefore, we obtain that $\mathbf{w}$ is orthogonal to any vector on the hyperplane [27].

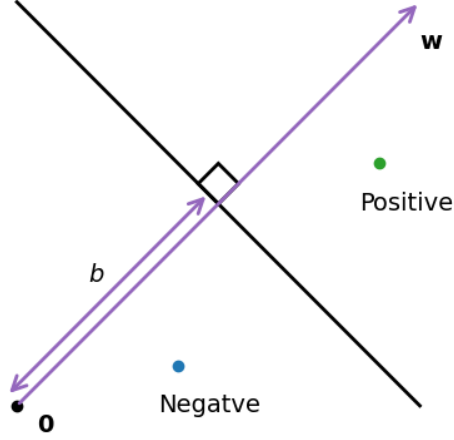


Figure C.19: SVM affine subspace

The SVM affine subspace hyperplane characterized by Equation C.37.

$$f(\mathbf{x}_p) - f(\mathbf{x}_q) = (\langle \mathbf{w}, \mathbf{x}_p \rangle + b) - (\langle \mathbf{w}, \mathbf{x}_q \rangle + b) = \langle \mathbf{w}, \mathbf{x}_p - \mathbf{x}_q \rangle = 0. \quad (\text{C.38})$$

When presented with a test example, we classify the example as positive or negative depending on the side of the hyperplane on which it occurs. Note that Equation C.37 not only defines a hyperplane, it additionally defines a direction. In other words, it defines the positive and negative side of the hyperplane. Geometrically, the positive examples lie above the hyperplane and the negative examples below the hyperplane. When training the classifier, we want to ensure that the examples with positive labels are on the positive side, and the examples with negative labels are on the negative side [27].

$$\langle \mathbf{w}, \mathbf{x}_n \rangle + b \geq 0 \quad \text{when} \quad y_n = 1 \quad (\text{C.39})$$

$$\langle \mathbf{w}, \mathbf{x}_n \rangle + b < 0 \quad \text{when} \quad y_n = -1. \quad (\text{C.40})$$

Equivalently, these two conditions can be expressed in a single equation [27].

$$y_n(\langle \mathbf{w}, \mathbf{x}_n \rangle + b) \geq 0. \quad (\text{C.41})$$

For a dataset $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$ that is linearly separable, we have infinitely many candidate hyperplanes, and therefore classifiers, that solve our classification problem without any training error. To find a unique solution, we can choose the separating hyperplane that maximizes margin between the positive and negative examples. In other words, we want the positive and negative examples to be separated by a large margin. For this purpose, we can make use of the fact that the closest point on the hyperplane to a given point $\mathbf{x}_n$ is obtained by the orthogonal projection [27].

Consider a hyperplane $\langle \mathbf{w}, \mathbf{x} \rangle + b$ and an example $\mathbf{x}_p$ that is on the positive side of the hyperplane so that $\langle \mathbf{w}, \mathbf{x} \rangle + b > 0$. We want to compute the distance $r > 0$ of $\mathbf{x}_p$ from the hyperplane. We do so by considering the orthogonal projection of $\mathbf{x}_p$ onto the hyperplane, which we denote by $\mathbf{x}'_p$. Since $\mathbf{w}$ is orthogonal to the hyperplane we know that the distance r is just a scaling of the length of this vector $\mathbf{w}$. If the length of $\mathbf{w}$ is known, then we can use this scaling factor r to work out the absolute distance between $\mathbf{x}_p$ and $\mathbf{x}'_p$. For convenience, we use a vector of unit length $\mathbf{w}/\|\mathbf{w}\|$ [27].

$$\mathbf{x}_p = \mathbf{x}'_p + r \frac{\mathbf{w}}{\|\mathbf{w}\|}. \quad (\text{C.42})$$

Say that we would like the positive examples to be further than r from the hyperplane, and the negative examples to be further than distance r in the negative direction from the hyperplane. In other words, we combine the requirements that examples are at least r distance away from the hyperplane in the positive and negative directions into one single inequality. Since we are interested only the direction, we add an assumption to our model that the parameter vector $\mathbf{w}$ is of unit length $\|\mathbf{w}\|$, where we use the Euclidean norm $\|\mathbf{w}\| = \sqrt{\mathbf{w}^\top \mathbf{w}}$. This assumption allows a more intuitive interpretation of the distance r [27].

Collecting these three requirements into a single constrained optimization problem, we obtain the following objective [27].

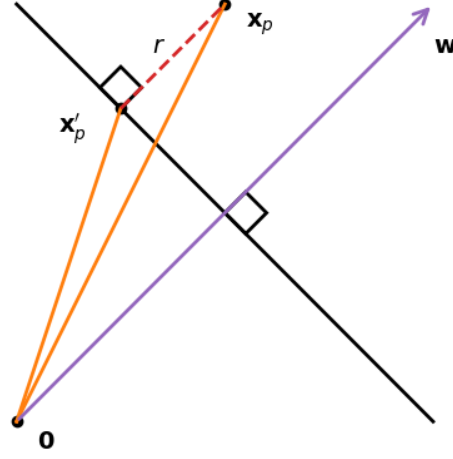


Figure C.20: SVM margin

The margin defined in Equation C.42.

$$\begin{aligned}
 & \max_{\mathbf{w}, b, r} \quad r \\
 & \text{subject to} \quad y_n(\langle \mathbf{w}, \mathbf{x}_n \rangle + b) \geq r, \\
 & \quad \quad \quad \|\mathbf{w}\| = 1, \quad r > 0.
 \end{aligned} \tag{C.43}$$

C.6.2 Probability Calibration

The SVM is a binary classifier that does not naturally lend itself to a probabilistic interpretation. Platt scaling is one approach to converting the raw output of the linear function into a calibrated class probability estimate that involves an additional calibration step [27]. The idea is to compute $q = \sigma(az + b)$, where z is the log-odds, or logit, of the SVM scores, a and b are estimated via maximum likelihood on a validation set, and $q = \sigma(\cdot)$ comes from logistic regression [58].

In other words, for the binary classification case, Platt scaling uses logistic regression on the SVM scores, fit by an additional cross-validation on the training data. The cross-

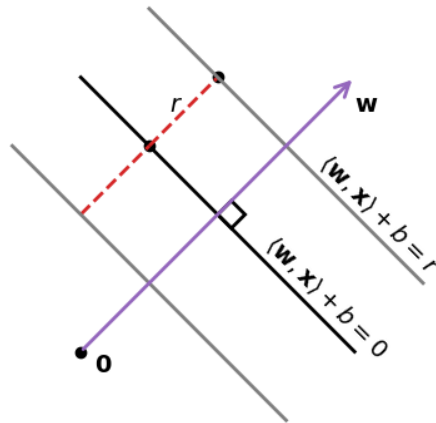


Figure C.21: The SVM objective defined by Equation C.43.

validation involved in Platt scaling is an expensive operation for large datasets [63]. Platt’s method is also known to have theoretical issues. This method can easily overfit resulting in overconfident calibrated probabilities [58].

Naturally, one could take a more Bayesian view of the classifier output by estimating a posterior distribution using Bayesian logistic regression. The Bayesian view also includes the specification of the prior, which includes design choices such as conjugacy with the likelihood. Additionally, one could consider latent functions as priors, which results in Gaussian process classification (see section C.7) [27].

C.6.3 Further Reading

The traditional derivation of the margin is known as the hard margin SVM. The reason for the expression “hard” is because the formulation does not allow for any violations of the margin condition. In the case where data is not linearly separable, we may wish to allow some examples to fall within the margin region, or even to be on the wrong side of the hyperplane. The model that allows for some classification errors is called the soft margin SVM [27].

The key geometric idea of the soft margin SVM is the to introduce a slack variable ξ_n corresponding to each example-label pair $(\mathbf{x}_n, y_n)$. We subtract the value of ξ_n from the margin, constraining ξ_n to be non-negative. To encourage correct classification of samples, ξ_n is added to the optimization objective [27].

A different approach to deriving the SVM follows the principle of empirical risk minimization. This requires minimization of a chosen loss function for our binary classification task. The ideal loss function between binary labels is the count the number of mismatches between the prediction and the label. This is denoted by $\mathbf{1}(f(\mathbf{x}_n) \neq y_n)$ and is called the zero-one loss. Unfortunately, the zero-one loss results in a combinatorial optimization problem for finding the best parameters $\mathbf{w}, b$, and in general these are more challenging to solve than continuous optimization problems. An equivalent reformulation of our objective allows us to use hinge loss [27].

Yet another equivalent optimization problem of the SVM is the dual view which is independent of the number of features. Instead, the number of parameters increases with the number of examples in the training set. This is useful for problems where we have more features than the number of examples in the training dataset [27].

The formulation of the dual SVM can be written such that the inner product in its objective occurs only between examples $\mathbf{x}_i$ and $\mathbf{x}_j$. In other words, there are no inner products between the examples and the parameters. Therefore, if we consider a set of features $\phi(\mathbf{x}_i)$ to represent $\mathbf{x}_i$, the only change in the dual SVM will be to replace the inner product. This modularity, where the choice of the SVM classification method and the choice of the feature representation $\phi(\mathbf{x})$ can be considered separately, provides flexibility for us to explore the two problems independently [27].

Since $\phi(\mathbf{x}_i)$ could be a non-linear function, we can use the SVM (which assumes a linear classifier) to construct classifiers that are nonlinear in the examples $\mathbf{x}_n$. This provides a second avenue, in addition to the soft margin, for users to deal with a dataset that is not linearly separable. Instead of explicitly defining a non-linear feature map $\phi(\cdot)$ and computing the results inner product between examples $\mathbf{x}_i$ and $\mathbf{x}_j$, we can define a similarity function

$k(\mathbf{x}_i, \mathbf{x}_j)$ between $\mathbf{x}_i$ and $\mathbf{x}_j$ [27].

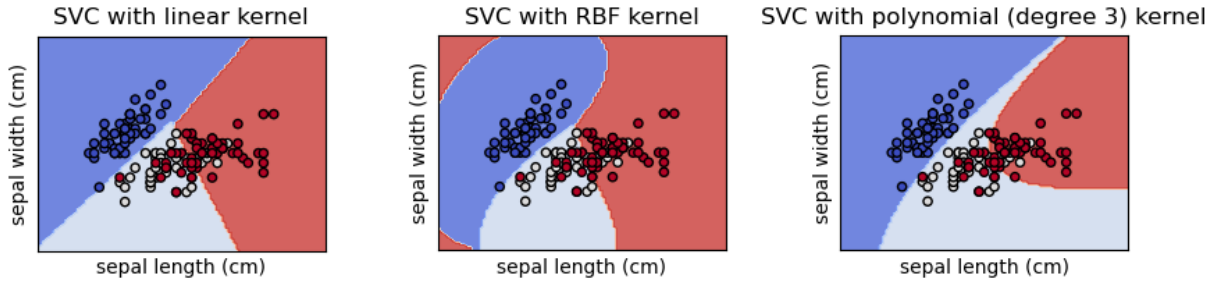


Figure C.22: Example of different support vector classification (SVC) kernels applied to the classic Iris flower dataset [63].

For a certain class of similarity functions, called kernels, the similarity function implicitly defines a non-linear feature map $\phi(\cdot)$. Kernels are by definition functions $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ for which there exists a Hilbert space $\mathcal{H}$ and a feature map $\phi : \mathcal{X} \rightarrow \mathcal{H}$ such that $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle_{\mathcal{H}}$. There is a unique reproducing kernel Hilbert space (RKHS) associated with every kernel k . The generalization from an inner product to a kernel function is known as the kernel trick [27].

C.7 Gaussian Process

The problem of machine learning (ML) is induction: We need to move from a finite training data $\mathcal{D}$ to a function f that makes predictions for all possible input values. A common method is to restrict the class of functions considered, for example only considering linear functions of the input. Another common method is to give a prior probability to all possible functions, where higher probabilities are given to functions that we consider to be more likely, for example because they are smoother than other functions. The Gaussian process (GP) belongs to the latter approach [68].

A Gaussian process is a generalization of the Gaussian probability distribution. Whereas a probability distribution describes random variables which are scalars or vectors, a stochastic

process governs the properties of functions. A function can be loosely thought of as a very long vector, each entry in the vector specifying the function value $f(x)$ at a particular input x . It turns out that if we only ask for the properties of the function at a finite number of points, then inference in the Gaussian process will give us the same answer as if we had considered all points [68].

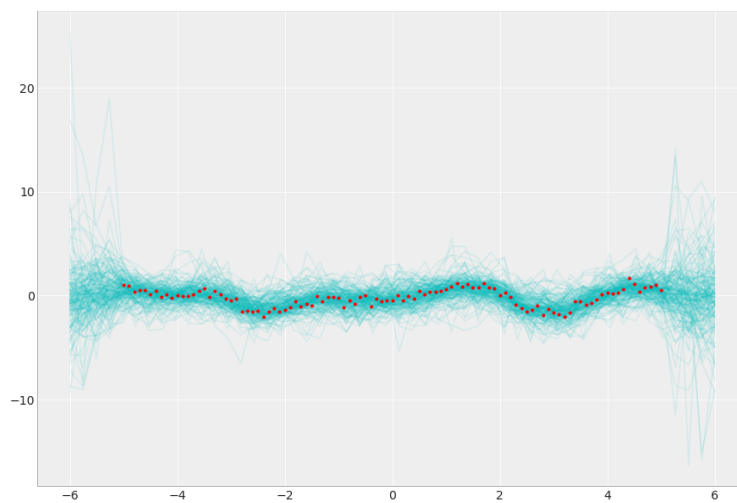


Figure C.23: Example of a Gaussian process. Data points (red) are used to fit estimator functions (cyan). We see a distribution of functions [31].

C.7.1 Preliminaries

There are two equivalent ways to interpret Gaussian processes for models. One can think of a Gaussian process as defining a distribution over functions, and inference taking place in function space: the function space view. Alternatively, the weight space view considers distributions over the weights of models. Let us consider standard linear regression as an example [68].

Weight Space View

Say we have a training set $\mathcal{D}$ of n observations, $\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid i = 1, \dots, n\}$, where $\mathbf{x}$ denotes an input vector of dimension D and y denotes a scalar output or target. The column vector inputs for all n cases are aggregated in the $D \times n$ design matrix $\mathbf{X}$, and the target are collected in the vector $\mathbf{y}$, so we can write $\mathcal{D} = (\mathbf{X}, \mathbf{y})$. The standard linear regression model with Gaussian noise is written as follows where $\mathbf{x}$ is the input vector, $\mathbf{w}$ is a vector of weights (parameters) of the linear model, f is the function value, and y is the observed target value [68].

$$f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}, \quad y = f(\mathbf{x}) + \epsilon. \quad (\text{C.44})$$

Often a bias weight or offset is included, but this can be implemented by augmenting the input vector $\mathbf{x}$ and weight vector $\mathbf{w}$, and so is not explicitly included in this notation. We have assumed that the observed values y differ from the function values $f(\mathbf{x})$ by additive noise, and further assumed that this noise follows an independent and identically distributed (IID) Gaussian distribution with zero mean and variance σ_n^2 [68].

$$\epsilon \sim \mathcal{N}(0, \sigma_n^2). \quad (\text{C.45})$$

This gives rise to the likelihood, the probability density of the observations given the parameters, which is factored as follows by the independence assumption [68].

$$\begin{aligned} p(\mathbf{y}|\mathbf{X}, \mathbf{w}) &= \prod_{i=1}^n p(y_i|\mathbf{x}_i, \mathbf{w}) \\ &= \prod_{i=1}^n \frac{1}{\sqrt{2\pi}\sigma_n} \exp\left(-\frac{(y_i - \mathbf{x}_i^\top \mathbf{w})^2}{2\sigma_n^2}\right) \\ &= \frac{1}{(2\pi\sigma_n^2)^{n/2}} \exp\left(-\frac{1}{2\sigma_n^2} \|\mathbf{y} - \mathbf{X}^\top \mathbf{w}\|^2\right) \\ &= \mathcal{N}(\mathbf{X}^\top \mathbf{w}, \sigma_n^2 \mathbf{I}). \end{aligned} \quad (\text{C.46})$$

For Bayesian formalism we specify a prior over the parameters using a zero mean Gaussian with covariance matrix Σ_p on the weights [68].

$$\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \Sigma_p). \quad (\text{C.47})$$

Inference in the Bayesian linear model is based on the posterior distribution over the weights computed by Bayes' rule [68].

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{marginal likelihood}}, \quad p(\mathbf{w}|\mathbf{y}, \mathbf{X}) = \frac{p(\mathbf{y}|\mathbf{X}, \mathbf{w})p(\mathbf{w})}{p(\mathbf{y}|\mathbf{X})}.$$

The normalizing constant known as the marginal likelihood or evidence is independent of the weights and given by the following [68].

$$p(\mathbf{y}|\mathbf{X}) = \int p(\mathbf{y}|\mathbf{X}, \mathbf{w})p(\mathbf{w})d\mathbf{w}. \quad (\text{C.48})$$

The posterior combines the likelihood and the prior capturing everything we know about the parameters, and writing only the terms from the likelihood and prior which depend on the weights we obtain the following where $\bar{\mathbf{w}} = \sigma_n^{-2}(\sigma_n^{-2}\mathbf{X}\mathbf{X}^\top + \Sigma_p^{-1})^{-1}\mathbf{X}\mathbf{y}$ [68].

$$\begin{aligned} p(\mathbf{w}|\mathbf{X}, \mathbf{y}) &\propto \exp\left(-\frac{1}{2\sigma_n^2}(\mathbf{y} - \mathbf{X}^\top\mathbf{w})^\top(\mathbf{y} - \mathbf{X}^\top\mathbf{w})\right) \exp\left(-\frac{1}{2}\mathbf{w}^\top\Sigma_p^{-1}\mathbf{w}\right) \\ &\propto \exp\left(-\frac{1}{2}(\mathbf{w} - \bar{\mathbf{w}})^\top\left(\frac{1}{\sigma_n^2}\mathbf{X}\mathbf{X}^\top + \Sigma_p^{-1}\right)(\mathbf{w} - \bar{\mathbf{w}})\right). \end{aligned} \quad (\text{C.49})$$

The form of the posterior distribution is also Gaussian with mean $\bar{\mathbf{w}}$ and covariance matrix $\mathbf{A}^{-1}$ where $\mathbf{A} = \sigma_n^{-2}\mathbf{X}\mathbf{X}^\top + \Sigma_p^{-1}$ [68].

$$p(\mathbf{w}|\mathbf{X}, \mathbf{y}) \sim \mathcal{N}\left(\frac{1}{\sigma_n^2}\mathbf{A}^{-1}\mathbf{X}\mathbf{y}, \mathbf{A}^{-1}\right). \quad (\text{C.50})$$

To make predictions for a test case we average over all possible parameter values weighted by their posterior probability. This is in contrast to non-Bayesian schemes where a single parameter is typically chosen by some optimization criterion. Thus the predictive distribution

for $f_* = f(\mathbf{x}_*)$ at $\mathbf{x}_*$ is given by averaging the output of all output linear models with respect to the Gaussian posterior. The predictive distribution is again Gaussian [68].

$$\begin{aligned}
 p(f_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) &= \int p(f_*|\mathbf{x}_*, \mathbf{w})p(\mathbf{w}|\mathbf{X}, \mathbf{y})d\mathbf{w} \\
 &= \int \mathbf{x}_*^\top \mathbf{w} p(\mathbf{w}|\mathbf{X}, \mathbf{y})d\mathbf{w} \\
 &= \mathcal{N}\left(\frac{1}{\sigma_n^2} \mathbf{x}_*^\top \mathbf{A}^{-1} \mathbf{X} \mathbf{y}, \mathbf{x}_*^\top \mathbf{A}^{-1} \mathbf{x}_*\right).
 \end{aligned} \tag{C.51}$$

The Bayesian linear model suffers from limited expressiveness. A simple idea to overcome this problem is to first project the inputs into some high dimensional space using a set of basis functions and then apply the linear model in this space instead of directly on the inputs themselves. For example, a scalar input x could be projected into the space of powers of x : $\phi(x) = (1, x, x^2, x^3, \dots)^\top$ to implement polynomial regression. This idea is also used in classification where a dataset which is not linearly separable in the original data space may become linearly separable in a high dimensional feature space [68].

We assume a given basis function $\phi(\mathbf{x})$ which maps a D dimensional input vector $\mathbf{x}$ into an N dimensional feature space. Further let the matrix $\Phi(\mathbf{X})$ be the aggregation of columns $\phi(\mathbf{x})$ for all cases in the training set. Now the model is as follows where the vector of parameters now has length N [68].

$$f(\mathbf{x}) = \phi(\mathbf{x})^\top \mathbf{w}. \tag{C.52}$$

The predictive distribution becomes the following with $\Phi = \Phi(\mathbf{X})$ and $\mathbf{A} = \sigma_n^{-2} \Phi \Phi^\top + \Sigma_p^{-1}$ [68].

$$f_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y} \sim \mathcal{N}\left(\frac{1}{\sigma_n^2} \phi(\mathbf{x}_*)^\top \mathbf{A}^{-1} \Phi \mathbf{y}, \phi(\mathbf{x}_*)^\top \mathbf{A}^{-1} \phi(\mathbf{x}_*)\right). \tag{C.53}$$

Equation C.53 can be rewritten in the following way where $\phi(\mathbf{x}_*) = \phi_*$ and $\mathbf{K} = \Phi^\top \Sigma_p \Phi$ [68].

$$\begin{aligned}
f_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y} &\sim \mathcal{N}(\phi_*^\top \Sigma_p \Phi (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{y}, \\
&\phi_*^\top \Sigma_p \phi_* - \phi_*^\top \Sigma_p \Phi (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \Phi^\top \Sigma_p \phi_*).
\end{aligned}
\tag{C.54}$$

To make predictions using Equation C.53 we need to invert the $\mathbf{A}$ matrix of size $N \times N$ which may not be convenient if the dimension of the feature space N is large. In Equation C.54 we need to invert matrices of size $n \times n$ which is more convenient when $n < N$. The feature space in Equation C.54 only appears in the form of $\Phi^\top \Sigma_p \Phi$, $\phi_*^\top \Sigma_p \Phi$, or $\phi_*^\top \Sigma_p \phi_*$. Thus the entries of these matrices are invariably of the form $\phi(\mathbf{x})^\top \Sigma_p \phi(\mathbf{x}')$ where $\mathbf{x}$ and $\mathbf{x}'$ are in either the training or the test sets [68].

Let us define $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \Sigma_p \phi(\mathbf{x}')$. We call $k(\mathbf{x}, \mathbf{x}')$ a covariance function or kernel. Notice that $\phi(\mathbf{x})^\top \Sigma_p \phi(\mathbf{x}')$ is an inner product with respect to Σ_p . As Σ_p is positive definite we can define $\Sigma_p^{1/2}$. Then defining $\psi(\mathbf{x}) = \Sigma_p^{1/2} \phi(\mathbf{x})$ we obtain a simple dot product representation $k(\mathbf{x}, \mathbf{x}') = \psi(\mathbf{x}) \psi(\mathbf{x}')$ [68].

If an algorithm is defined solely in terms of inner products in input space then it can be lifted into feature space by replacing occurrences of those inner products by $k(\mathbf{x}, \mathbf{x}')$. This is called the kernel trick. This technique is particularly valuable in situations where it is more convenient to compute the kernel than the feature vectors themselves. This often leads to considering the kernel as the object of primary interest, and its corresponding feature space as having secondary practical importance [68].

Function Space View

An alternative and equivalent way of reaching identical results to the weight space view is possible by considering inference directly in function space, and we can use a Gaussian process to describe a distribution over functions [68].

Definition 1. A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution [68].

A Gaussian process is completely specified by its mean function and covariance function, which we define as $m(\mathbf{x})$ and $k(\mathbf{x}, \mathbf{x}')$ respectively for a real process $f(\mathbf{x})$ [68].

$$\begin{aligned} m(\mathbf{x}) &= \mathbb{E}[f(\mathbf{x})] \\ k(\mathbf{x}, \mathbf{x}') &= \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))]. \end{aligned} \tag{C.55}$$

We write a Gaussian process as follows [68].

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')). \tag{C.56}$$

A simple example of a Gaussian process can be obtained from our Bayesian linear regression model $f(\mathbf{x}) = \phi(\mathbf{x})^\top \mathbf{w}$ with prior $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \Sigma_p)$ [68].

$$\begin{aligned} \mathbb{E}[f(\mathbf{x})] &= \phi(\mathbf{x})^\top \mathbb{E}[\mathbf{w}] = 0 \\ \mathbb{E}[f(\mathbf{x})f(\mathbf{x}')] &= \phi(\mathbf{x})^\top \mathbb{E}[\mathbf{w}\mathbf{w}^\top] \phi(\mathbf{x}') = \phi(\mathbf{x})^\top \Sigma_p \phi(\mathbf{x}'). \end{aligned} \tag{C.57}$$

Thus $f(\mathbf{x})$ and $f(\mathbf{x}')$ are jointly Gaussian with zero mean and covariance given by $\phi(\mathbf{x})^\top \Sigma_p \phi(\mathbf{x}')$. Our running example of a covariance function will be the squared exponential (SE) covariance function. For this particular covariance function, we see that the covariance is almost unity between variables whose corresponding inputs are close, and decreases as their distance in the input space increases [68].

$$\text{cov}(f(\mathbf{x}_p), f(\mathbf{x}_q)) = k(\mathbf{x}_p, \mathbf{x}_q) = \exp\left(-\frac{1}{2}|\mathbf{x}_p - \mathbf{x}_q|^2\right). \tag{C.58}$$

The specification of the covariance function implies a distribution over functions. We can draw samples from the distribution of functions evaluated at any number of points. We choose a number of input points $\mathbf{X}_*$ and write the corresponding covariance matrix elementwise using Equation C.58. Then we generate a random Gaussian vector with this covariance matrix and further can plot the generated values as a function of the inputs [68].

$$\mathbf{f}_* \sim \mathcal{N}(\mathbf{0}, K(\mathbf{X}_*, \mathbf{X}_*)). \tag{C.59}$$

Prediction with Noise Free Observations

We are not primarily interested in drawing random functions from the prior, but want to incorporate the knowledge that the training data provides about the function. Let us consider the simple special case where the observations are noise free. That is, we know $\{(\mathbf{x}_i, f_i) | i = 1, \dots, n\}$. The joint distribution of the training outputs $\mathbf{f}$ and the test outputs $\mathbf{f}_*$ according to the prior are as follows [68].

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N} \left(\mathbf{0}, \begin{bmatrix} K(\mathbf{X}, \mathbf{X}) & K(\mathbf{X}, \mathbf{X}_*) \\ K(\mathbf{X}_*, \mathbf{X}) & K(\mathbf{X}_*, \mathbf{X}_*) \end{bmatrix} \right). \quad (\text{C.60})$$

If there are n training points and n_* test points then $K(\mathbf{X}, \mathbf{X}_*)$ denotes the $n \times n_*$ matrix of covariances evaluated at all pairs of training and test points, and similarly for the other entries $K(\mathbf{X}, \mathbf{X})$, $K(\mathbf{X}_*, \mathbf{X}_*)$, and $K(\mathbf{X}_*, \mathbf{X})$. To get the posterior distribution over functions we need to restrict this joint prior distribution to contain only those functions which agree with the observed data points [68].

Graphically we may think of generating functions from the prior, and rejecting the ones that disagree with the observations, although this strategy is computationally inefficient. Though in probabilistic terms this operation is simple, corresponding to conditioning the joint Gaussian prior distribution on the observations. In the following, function values $\mathbf{f}_*$ corresponding to test inputs $\mathbf{X}_*$ can be sampled from the joint posterior distribution by evaluating the mean and covariance matrix [68].

$$\begin{aligned} \mathbf{f}_* | \mathbf{X}_*, \mathbf{X}, \mathbf{f} &\sim \mathcal{N}(K(\mathbf{X}_*, \mathbf{X})K(\mathbf{X}, \mathbf{X})^{-1}\mathbf{f}, \\ &K(\mathbf{X}_*, \mathbf{X}_*) - K(\mathbf{X}_*, \mathbf{X})K(\mathbf{X}, \mathbf{X})^{-1}K(\mathbf{X}, \mathbf{X}_*)). \end{aligned} \quad (\text{C.61})$$

C.7.2 Classification

We have considered regression problems where the targets are real valued. Classification problems aim to assign an input $\mathbf{x}$ to one of C classes $\mathcal{C}_1, \dots, \mathcal{C}_C$. These problems can either be binary ($C = 2$) or multiclass ($C > 2$). Let us focus on probabilistic classification

where test predictions take the form of class probabilities. Since generalization to test cases inherently involves some level of uncertainty, it is natural to attempt to make predictions in a way that reflects these uncertainties [68].

Both classification and regression can be viewed as function approximation problems. Unfortunately, the solution of classification problems using Gaussian processes is more demanding than for regression problems. This is because regression problems can assume that the likelihood function is Gaussian. A Gaussian process prior combined with a Gaussian likelihood gives rise to a posterior Gaussian process over functions and everything remains analytically tractable. For classification models where the targets are discrete class labels, the Gaussian likelihood is inappropriate, and we must use methods of approximate inference since exact inference is not feasible [68].

We have seen how Gaussian process regression (GPR) can be obtained by generalizing linear regression. Logistic regression describes an analog of linear regression in the classification case. It is generalized to yield Gaussian process classification (GPC) using again the ideas behind the generalization of linear regression for GPR [68].

For binary discriminative classification the output of a regression model can turn into a class probability using a response function (the inverse of a link function). This squashes its arguments which can lie in the domain $(-\infty, \infty)$ into the domain $[0, 1]$ guaranteeing a valid probabilistic interpretation. An example is the linear logistic regression model (section C.5 for more information) which combines the linear model with the logistic response function [68].

$$p(\mathcal{C}_1|\mathbf{x}) = \lambda(\mathbf{x}^\top \mathbf{w}) \quad \text{where} \quad \lambda(z) = \frac{1}{1 + \exp(-z)}. \quad (\text{C.62})$$

Let us consider linear models for binary classification which form the foundation of Gaussian process classification models. We use the labels $y = 1$ and $y = -1$ to distinguish the two classes. The likelihood is given by $p(y = 1|\mathbf{x}, \mathbf{w}) = \sigma(\mathbf{x}^\top \mathbf{w})$ where $\mathbf{w}$ is the weights vector and $\sigma(z)$ can be any sigmoid function. When using the logistic $\sigma(z) = \lambda(z)$ from Equation C.62 the model is simply called logistic regression. At the probability of the two

classes must sum to 1, we have $p(y = -1|\mathbf{x}, \mathbf{w}) = 1 - p(y = 1|\mathbf{x}, \mathbf{w})$. Thus for a data points $(\mathbf{x}_i, y_i)$ the likelihood is given by $\sigma(\mathbf{x}_i^\top \mathbf{w})$ if $y_i = 1$, and $1 - \sigma(\mathbf{x}_i^\top \mathbf{w})$ if $y_i = -1$ [68].

For binary classification the basic idea behind Gaussian process prediction is that we place a GP prior over the latent function $f(\mathbf{x})$ and then squash this through the logistic function to obtain a prior on $\pi(\mathbf{x}) = p(y = 1|\mathbf{x}) = \sigma(f(\mathbf{x}))$. Note that π is a deterministic function of f , and since f is stochastic, so is π . The latent function f is a nuisance function that we do not observe values of itself, and instead only observe the inputs $\mathbf{X}$ and the class labels $\mathbf{y}$. We are not particularly interested in the values of f , but rather in π , in particular for test cases $\pi(\mathbf{x}_*)$. The purpose of f is solely to allow a convenient formulation of the model, and the computational goal is to integrate out f [68].

Inference is naturally divided into two steps: First we compute the distribution of the latent variable corresponding to a test case in the following where $p(\mathbf{f}|\mathbf{X}, \mathbf{y}) = p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|\mathbf{X})/p(\mathbf{y}|\mathbf{X})$ is the posterior over the latent variables [68].

$$p(f_*|\mathbf{X}, \mathbf{y}, \mathbf{x}_*) = \int p(f_*|\mathbf{X}, \mathbf{x}_*, \mathbf{f})p(\mathbf{f}|\mathbf{X}, \mathbf{y})d\mathbf{f}. \quad (\text{C.63})$$

Second we use this distribution over the latent f_* to produce a probabilistic prediction [68].

$$\bar{\pi}_* = p(y_* = 1|\mathbf{X}, \mathbf{y}, \mathbf{x}_*) = \int \sigma(f_*)p(f_*|\mathbf{X}, \mathbf{y}, \mathbf{x}_*)df_*. \quad (\text{C.64})$$

In the regression case with Gaussian likelihood, computation of predictions was straightforward as the relevant integrals were Gaussian and could be computed analytically. In classification the non-Gaussian likelihood in Equation C.63 makes the integral analytically intractable. Similarly, Equation C.64 can be analytically intractable for certain sigmoid functions. Thus we need to use approximations such as Markov chain Monte Carlo (MCMC) [68].

C.7.3 Covariance Functions

A covariance function encodes our assumptions about the function which we wish to learn. In supervised learning, the notion of similarity between data points is also crucial. It is a basic assumption that points $\mathbf{x}_p$ and $\mathbf{x}_q$ that are close are likely to have a similar target value y . Under the Gaussian process view it is the covariance function that defines nearness or similarity [68].

A stationary covariance function is a function of $\mathbf{x} - \mathbf{x}'$. Thus it is invariant to translations in the input space. If further the covariance function is a function only of $|\mathbf{x} - \mathbf{x}'|$ then it is called isotropic, and is thus invariant to all rigid motions. The squared exponential covariance function is isotropic. These functions are also known as radial basis functions (RBFs). Its formulation is as follows where ℓ is the characteristic length scale [68].

$$k(\mathbf{x}, \mathbf{x}') = \exp \left(-\frac{|\mathbf{x} - \mathbf{x}'|^2}{2\ell^2} \right). \quad (\text{C.65})$$

Furthermore, if a covariance function depends only on $\mathbf{x}$ and $\mathbf{x}'$ through $\mathbf{x} \cdot \mathbf{x}'$ it is called a dot product covariance function. The linear covariance function is a special case of the dot product covariance function. The linear covariance function is written as follows where $\mathbf{c}$ is constant [68].

$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x} - \mathbf{c})(\mathbf{x}' - \mathbf{c}). \quad (\text{C.66})$$

A general name for a function k of two arguments mapping a pair of inputs $\mathbf{x} \in \mathcal{X}, \mathbf{x}' \in \mathcal{X}$ into $\mathbb{R}$ is a kernel. A real kernel is said to be symmetric if $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$. Covariance functions must be symmetric by definition [68].

C.7.4 Sparse Approximation

The best way to perform GP inference and training is to compute a Cholesky decomposition of the $N \times N$ Gram matrix. Unfortunately, this take $\mathcal{O}(N^3)$ time. Instead, we can use an approximation method based on inducing points, also called pseudoinputs, which are like a

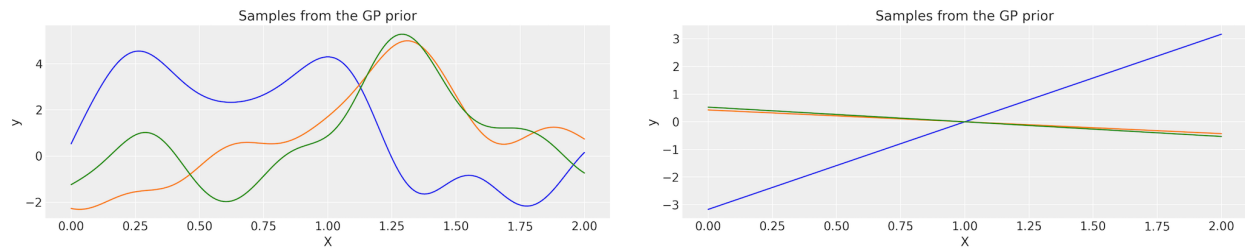


Figure C.24: Example functions drawn from a GP prior with a given covariance function. (Left) The squared exponential covariance function. (Right) The linear covariance function [29].



Figure C.25: Samples from additional covariance functions. (Top Left) The Matérn $5/2$ kernel. (Top Right) The Matérn $3/2$ kernel. (Bottom Left) The exponential kernel. (Bottom Right) The polynomial kernel [29].

learned summary of the training data that we can condition on, rather than conditioning on all of it [58].

Let $\mathbf{X}$ be the observed inputs, and $\mathbf{f}_X = f(\mathbf{X})$ be the unknown vector of function values for which we have observations $\mathbf{y}$. Let $\mathbf{f}_*$ be the unknown function values at one or more test points $\mathbf{X}_*$. Finally, let us assume we have M additional inputs $\mathbf{Z}$ with unknown function values $\mathbf{f}_Z$ also denoted by $\mathbf{u}$. The exact joint prior has the following form [58].

$$p(\mathbf{f}_X, \mathbf{f}_*) = \int p(\mathbf{f}_*, \mathbf{f}_X, \mathbf{f}_Z) d\mathbf{f}_Z = \int p(\mathbf{f}_*, \mathbf{f}_X | \mathbf{f}_Z) p(\mathbf{f}_Z) d\mathbf{f}_Z = \mathcal{N} \left(\mathbf{0}, \begin{pmatrix} \mathbf{K}_{X,X} & \mathbf{K}_{X,*} \\ \mathbf{K}_{*,X} & \mathbf{K}_{*,*} \end{pmatrix} \right). \quad (\text{C.67})$$

We can choose $\mathbf{f}_Z$ in such a way that it acts as a sufficient statistic for the data, so that we can predict $\mathbf{f}_*$ just using $\mathbf{f}_Z$ instead of $\mathbf{f}_X$, and thus we approximate the prior as follows [58].

$$p(\mathbf{f}_*, \mathbf{f}_X, \mathbf{f}_Z) = p(\mathbf{f}_* | \mathbf{f}_X, \mathbf{f}_Z) p(\mathbf{f}_X | \mathbf{f}_Z) p(\mathbf{f}_Z) \approx p(\mathbf{f}_* | \mathbf{f}_Z) p(\mathbf{f}_X | \mathbf{f}_Z) p(\mathbf{f}_Z). \quad (\text{C.68})$$

From this we derive the following train and test conditionals [58].

$$p(\mathbf{f}_X | \mathbf{f}_Z) = \mathcal{N}(\mathbf{f}_X | \mathbf{K}_{X,Z} \mathbf{K}_{Z,Z}^{-1} \mathbf{f}_Z, \mathbf{K}_{X,X} - \mathbf{Q}_{X,X}) \quad (\text{C.69})$$

$$p(\mathbf{f}_* | \mathbf{f}_Z) = \mathcal{N}(\mathbf{f}_* | \mathbf{K}_{*,Z} \mathbf{K}_{Z,Z}^{-1} \mathbf{f}_Z, \mathbf{K}_{*,*} - \mathbf{Q}_{*,*}). \quad (\text{C.70})$$

The above equations can be seen as exact inference on noise-free observations $\mathbf{f}_Z$. To gain computational speedups, we can make further approximations to the terms $\tilde{\mathbf{Q}}_{X,X} = \mathbf{K}_{X,X} - \mathbf{Q}_{X,X}$ and $\tilde{\mathbf{Q}}_{*,*} = \mathbf{K}_{*,*} - \mathbf{Q}_{*,*}$. We can then derive the approximate prior $q(\mathbf{f}_X, \mathbf{f}_*) = \int q(\mathbf{f}_X | \mathbf{f}_Z) q(\mathbf{f}_* | \mathbf{f}_Z) p(\mathbf{f}_Z) d\mathbf{f}_Z$, which we then condition on the observations in the usual way. All of these approximations result in a training cost of $\mathcal{O}(M^3 + NM^2)$, and then take $\mathcal{O}(M)$ time for the predictive mean for each test case, and $\mathcal{O}(M^2)$ time for the predictive variance.

Compare this to $\mathcal{O}(N^3)$ training time and $\mathcal{O}(N)$ and $\mathcal{O}(N^2)$ testing time for exact inference [58].

The deterministic inducing conditional (DIC) approximation, or the subset of regressors (SOR) approximation, results from assuming $\tilde{\mathbf{Q}}_{X,X} = \mathbf{0}$ and $\tilde{\mathbf{Q}}_{*,*} = \mathbf{0}$, so the conditionals are deterministic. This can result in an underestimate of the predictive variance. One way to overcome the overconfidence of DIC is to only assume $\tilde{\mathbf{Q}}_{X,X} = \mathbf{0}$ but let $\tilde{\mathbf{Q}}_{*,*} = \mathbf{K}_{*,*} - \mathbf{Q}_{*,*}$ be exact. This is called the deterministic training conditional (DTC). Lastly, the fully independent training conditional (FITC) approximation assumes $q(\mathbf{f}_X|\mathbf{f}_Z)$ is fully factorized. This throws away less uncertainty than the SOR and DTC methods, since it does not make any deterministic assumptions about the relationship between $\mathbf{f}_X$ and $\mathbf{f}_Z$ [58].

$$q_{\text{SOR}}(\mathbf{f}_X, \mathbf{f}_*) = \mathcal{N} \left(\mathbf{0}, \begin{pmatrix} \mathbf{Q}_{X,X} & \mathbf{Q}_{X,*} \\ \mathbf{Q}_{*,X} & \mathbf{Q}_{*,*} \end{pmatrix} \right) \quad (\text{C.71})$$

$$q_{\text{DTC}}(\mathbf{f}_X, \mathbf{f}_*) = \mathcal{N} \left(\mathbf{0}, \begin{pmatrix} \mathbf{Q}_{X,X} & \mathbf{Q}_{X,*} \\ \mathbf{Q}_{*,X} & \mathbf{K}_{*,*} \end{pmatrix} \right) \quad (\text{C.72})$$

$$q_{\text{FITC}}(\mathbf{f}_X, \mathbf{f}_*) = \mathcal{N} \left(\mathbf{0}, \begin{pmatrix} \mathbf{Q}_{X,X} - \text{diag}(\mathbf{Q}_{X,X} - \mathbf{K}_{X,X}) & \mathbf{Q}_{X,*} \\ \mathbf{Q}_{*,X} & \mathbf{K}_{*,*} \end{pmatrix} \right). \quad (\text{C.73})$$

The simplest approach to selecting the inducing points from the training set is to throw away some of the data. In this case, we could pick random examples. However, intuitively it makes more sense to try to pick a subset that in some sense covers the original data, so it contains approximately the same information without redundancy. Clustering algorithms such as k -means are a popular heuristic approach. We can also use coreset methods which can provably find such an information preserving subset. Furthermore, we can approach learning the inducing points by treating them like kernel hyperparameters, and choosing them so as to maximize the log marginal likelihood [58].

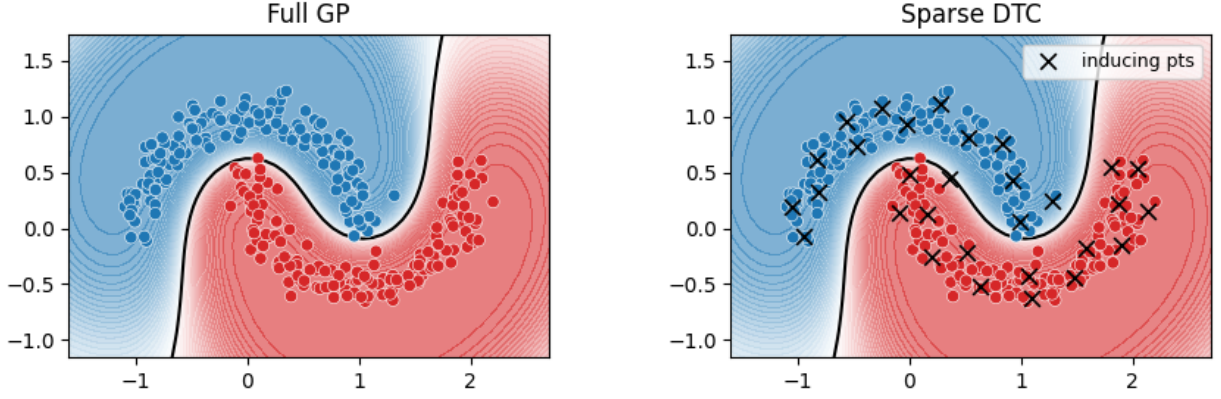


Figure C.26: Sparse GP comparison.

C.8 Deep Neural Network

Examples of machine learning (ML) models include logistic regression corresponding to the model $p(y|\mathbf{x}, \mathbf{w}) = \text{Ber}(y|\sigma(\mathbf{w}^\top \mathbf{x}))$ and generalized linear models (GLMs) which generalizes to other kinds of output distributions such as Poisson. However, these models make the strong assumption that the input-output mapping is linear. A simple way of increasing the flexibility of such models is to perform a feature transformation, by replacing $\mathbf{x}$ with $\phi(\mathbf{x})$. For example, we can use a polynomial transform, given by $\phi(\mathbf{x}) = [1, \mathbf{x}, \mathbf{x}^2, \mathbf{x}^3, \dots]$. This is called basis function expansion and the model becomes the following [57].

$$f(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{W}\phi(\mathbf{x}) + \mathbf{b}. \quad (\text{C.74})$$

The updated model still linear in the parameters $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b})$. This makes model fitting easy, however, having to specify the feature transformation by hand is very limiting. A natural extension is to give the feature extractor its own parameter $\boldsymbol{\theta}_2$ giving the following where $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)$ and $\boldsymbol{\theta}_1 = (\mathbf{W}, \mathbf{b})$ [57].

$$f(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{W}\phi(\mathbf{x}; \boldsymbol{\theta}_2) + \mathbf{b}. \quad (\text{C.75})$$

This process can be repeated recursively to create more and more complex functions, and if we compose L functions we get the following where $f_\ell(\mathbf{x}) = f(\mathbf{x}; \boldsymbol{\theta}_\ell)$ is the function at layer ℓ [57].

$$f(\mathbf{x}; \boldsymbol{\theta}) = f_L(f_{L-1}(\cdots(f_1(\mathbf{x}))\cdots)). \quad (\text{C.76})$$

This is the key idea behind deep neural networks (DNNs). The term “DNN” actually encompasses a larger family of models in which we compose differentiable functions into any kind of directed acyclic graph (DAG) mapping input to output. Equation C.76 is the simplest example where the DAG is a chain. This is known as a feedforward neural network (FFNN) or multilayer perceptron (MLP) and assumes that the input is a fixed-dimensional vector such as $\mathbf{x} \in \mathbb{R}^D$ [57].

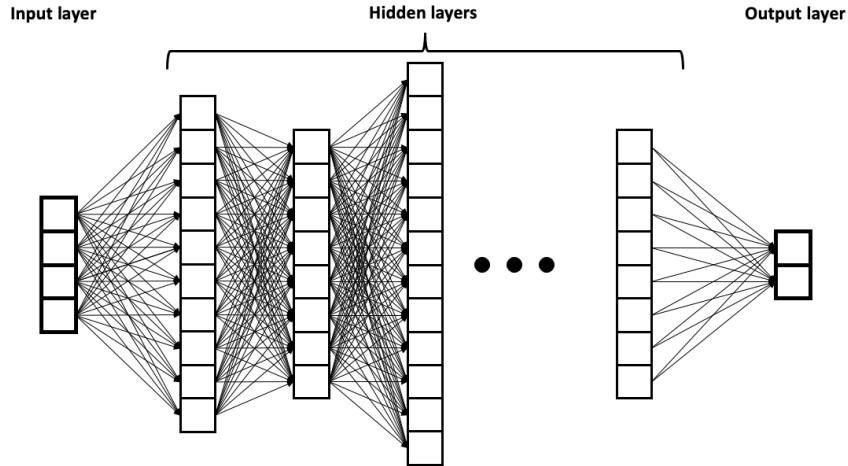


Figure C.27: A multilayer perceptron (MLP) [16].

Historically, the perceptron is a deterministic version of logistic regression with a mapping of the following form where $H(\cdot)$ is the Heaviside step function, also known as a linear threshold function [57].

$$f(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{1}(\mathbf{w}^\top \mathbf{x} + b \geq 0) = H(\mathbf{w}^\top \mathbf{x} + b). \quad (\text{C.77})$$

In this form, the MLP is a stack of perceptrons, each of which involves the non-differentiable Heaviside function. This makes models difficult to train, which is why it has never been widely used. However, we can replace the Heaviside function $H : \mathbb{R} \rightarrow \{0, 1\}$ with a differentiable activation function $\varphi : \mathbb{R} \rightarrow \mathbb{R}$. More precisely, we define the hidden units $\mathbf{z}_l$ at each layer l to be a linear transformation of the hidden units at the previous layer passed elementwise through this activation function [57].

$$\mathbf{z}_l = f_l(\mathbf{z}_{l-1}) = \varphi_l(\mathbf{b}_l + \mathbf{W}_l \mathbf{z}_{l-1}). \quad (\text{C.78})$$

The quantity that is passed to the activation function is called the pre-activations [57].

$$\mathbf{a}_l = \mathbf{b}_l + \mathbf{W}_l \mathbf{z}_{l-1}. \quad (\text{C.79})$$

If we now compute L of these functions together, as in Equation C.76, then we can compute the gradient of the output with respect to the parameters in each layer using the chain rule, also known as backpropagation. We can then pass the gradient to an optimizer, and thus minimize some training objective. The term “MLP” almost always refers to this differentiable form of the model, rather than the historical version with non-differentiable linear threshold units [57].

We are free to use any kind of differentiable activation function at each layer. However, if we use a linear activation function, then the whole model reduces to a regular linear model. In the early days of neural networks, a common choice of activation function was to use a sigmoid (logistic) function, which can be seen as a smooth approximation of the Heaviside function [57].

$$\sigma(a) = \frac{1}{1 + e^{-a}}. \quad (\text{C.80})$$

However, the sigmoid function saturates at 1 for large positive inputs, and at 0 for large negative inputs. Another common choice is the tanh function, which has a similar shape, but saturates at -1 and 1. In the saturated regimes, the gradient of the output with respect

to the input will be close to zero, so any gradient signal from higher layers will not be able to propagate back to earlier layers. This is called the vanishing gradient problem, and it makes it hard to train the model using gradient descent [57].

One of the keys to train very deep models is to use non-saturating activation functions. The most common of these is rectified linear unit (ReLU). The ReLU functions simply turns off negative inputs, and passes positive inputs unchanged [57].

$$\text{ReLU}(a) = \max(a, 0) = a\mathbf{1}(a > 0). \quad (\text{C.81})$$

It can be shown that an MLP with one hidden layer is a universal function approximator, meaning it can model any suitably smooth function, given enough hidden units, to any desired level of accuracy. Intuitively, the reason for this is that each hidden unit can specify a half plane, and a sufficiently large combination of these can divide any region of space, to which any response can be associated [57].

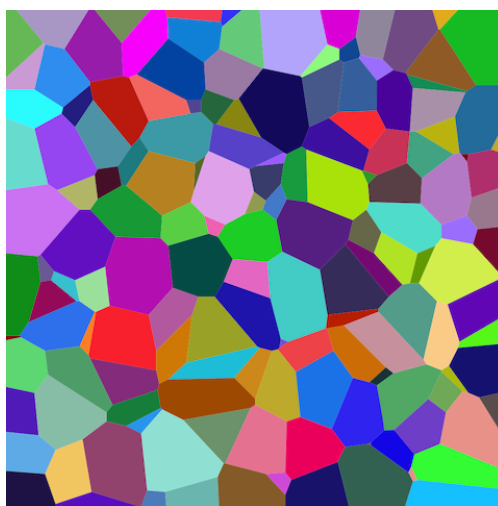


Figure C.28: Voronoi diagram visualization of the universal function approximation where each hidden unit specifies a half plane [45].

However, various arguments, both experimental and theoretical, have shown that deep

networks work better than shallow ones. The reason is that later layers can leverage the features that are learned by earlier layers. That is, the function is defined in a compositional or hierarchical way. Although we could fit any dataset with a single hidden layer model, intuitively, it is easier if the model learns to detect the separating characteristics of the dataset using the hidden units in an early layer, and then uses these features to define a simple linear classifier in a later layer [57].

C.8.1 Convolutional Neural Network

The convolutional neural network (CNN) is used to apply neural networks for images. In this case, we replace matrix multiplication with a convolution operation. The basic idea is to divide the input into overlapping 2D image patches, and to compare each patch with a set of small weight matrices, or filters, which represents part of an object [57].

This can be thought of as a form of template matching. We learn these templates from data. Since the templates are small (often just 3×3 or 5×5), the number of parameters needed for learning is small. Additionally, since we use convolution to do template matching, instead of matrix multiplication, the model will be translationally invariant. This is useful for tasks such as image classification, where the goal is to classify if an object is present regardless of its location [57].

The 1D convolution between two function $f, g : \mathbb{R}^D \rightarrow \mathbb{R}$ is defined as follows [57].

$$[f \otimes g](z) = \int_{\mathbb{R}^D} f(u)g(z - u)du. \quad (\text{C.82})$$

Now suppose we replace the functions with finite-length vectors, which we can think of as functions defined on a finite set of points. Let f be evaluated at the points $\{-L, -L + 1, \dots, 0, 1, \dots, L\}$ to yield the weight vector (also called a filter or kernel) $w_{-L} = f(-L)$ up to $w_L = f(L)$. Now let g be evaluated at the points $\{-N, \dots, N\}$ to yield the feature vector $x_{-N} = g(-N)$ up to $x_N = g(N)$. Then Equation C.82 becomes the following. We see that we flip the weight vector $\mathbf{w}$, and then drag it over the the $\mathbf{x}$ vector, summing up the local

windows at each point [57].

$$[\mathbf{w} \circledast \mathbf{x}](i) = w_{-L}x_{i+L} + \cdots + w_{-1}x_{i+1} + w_0x_i + w_1x_{i-1} + \cdots + w_Lx_{i-L}. \quad (\text{C.83})$$

When we do not flip $\mathbf{w}$ first, then we get the cross correlation [57].

$$[\mathbf{w} \circledast \mathbf{x}](i) = w_{-L}x_{i-L} + \cdots + w_{-1}x_{i-1} + w_0x_i + w_1x_{i+1} + \cdots + w_Lx_{i+L}. \quad (\text{C.84})$$

If the weight vector is symmetric, then cross correlation and convolution are the same. In the deep learning literature, the term “convolution” is often used to mean cross correlation. We can also evaluate the weights $\mathbf{w}$ on the domain $\{0, 1, \dots, L-1\}$ and the features $\mathbf{x}$ on the domain $\{0, 1, \dots, N-1\}$, to eliminate negative indices [57].

$$[\mathbf{w} \circledast \mathbf{x}](i) = \sum_{u=0}^{L-1} w_u x_{i+u}. \quad (\text{C.85})$$

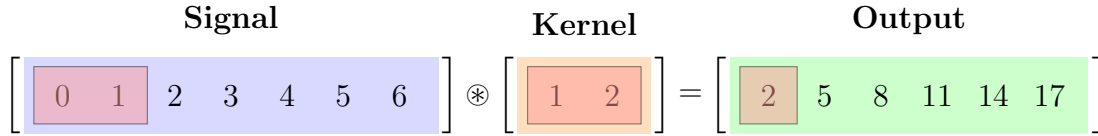


Figure C.29: 1D discrete convolution of a signal with a kernel.

In 2D, Equation C.85 becomes the following where the 2D filter $\mathbf{W}$ has size $H \times W$ [57].

$$[\mathbf{W} \circledast \mathbf{X}](i, j) = \sum_{u=0}^{H-1} \sum_{v=0}^{W-1} w_{u,v} x_{i+u, j+v}. \quad (\text{C.86})$$

We can think of 2D convolution as template matching, since the output at a point (i, j) will be large if the corresponding image centered on (i, j) is similar to $\mathbf{W}$. More generally, we can think of convolution as a form of feature detection. The resulting output $\mathbf{Y} = \mathbf{W} \circledast \mathbf{X}$ is therefore called a feature map [57].

Signal		Kernel		Output
$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix}$	$\circledast$	$\begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix}$	$=$	$\begin{bmatrix} 19 & 25 \\ 37 & 43 \end{bmatrix}$

Figure C.30: 2D discrete convolution of a signal with a kernel.

In Figure C.30 we see that convolving a 3×3 image with a 2×2 filter results in a 2×2 output. This is called valid convolution, since we only apply the filter to valid parts of the input, and we do not let it slide off the ends. If we want the output to have the same size as the input, we can use zero-padding, where we add a border of zeros to the image. This is called same convolution [57].

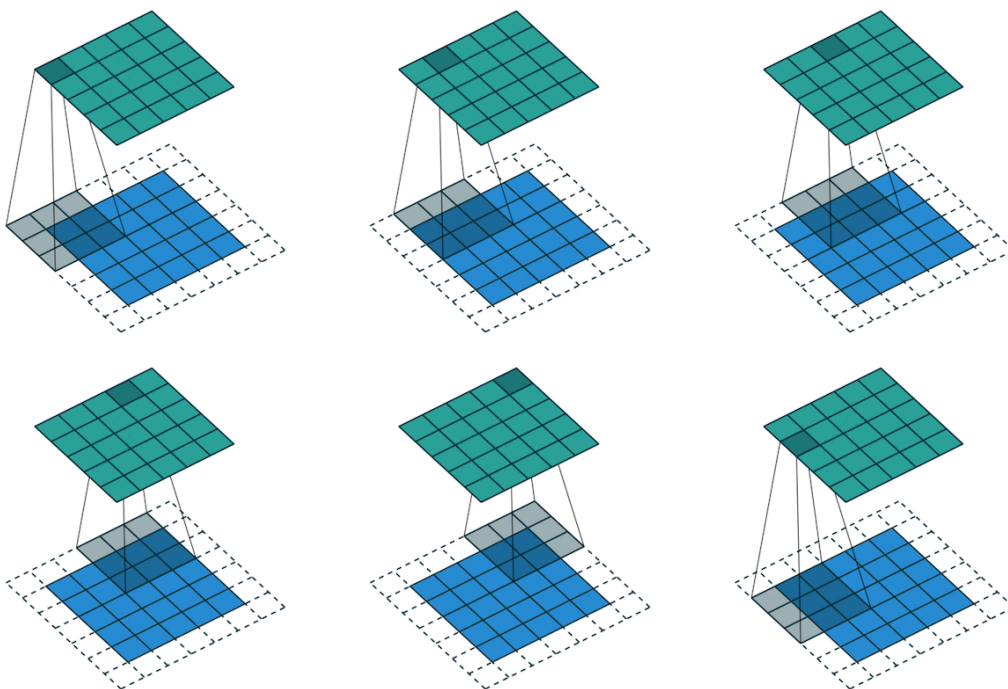


Figure C.31: The first 6 of 25 convolutions with padding [28].

Since each output pixel is generated by a weighted combination of inputs in its receptive field, neighboring outputs will be very similar in value, since their inputs are overlapping. We can reduce this redundancy and speedup computation by skipping every s number of inputs. This is called strided convolution [57].

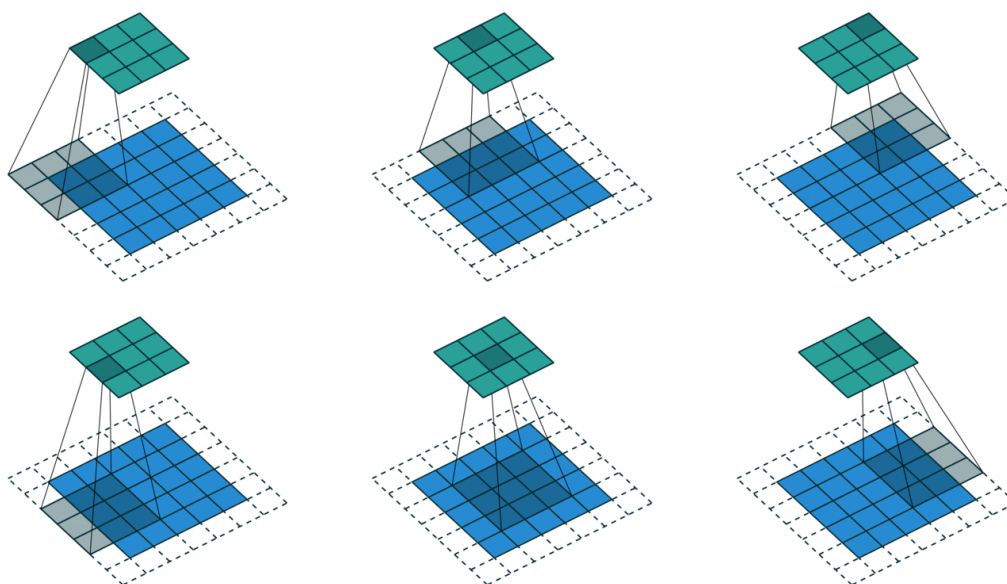


Figure C.32: The first 6 in 9 convolutions with padding and strides [28].

Convolution will preserve information about the location of input features, a property known as equivariance. In some cases we want to be invariant to the location. For example, when performing image classification, we may just want to know if an object of interest is present anywhere in the image. A simple way to achieve this is called max pooling, which computes the maximum over its incoming values. An alternative is to use average pooling, which replaces the maximum with the mean. A common design pattern is to create a CNN by alternating convolutional layers with max pooling layers, followed by a final linear classification layer at the end [57].

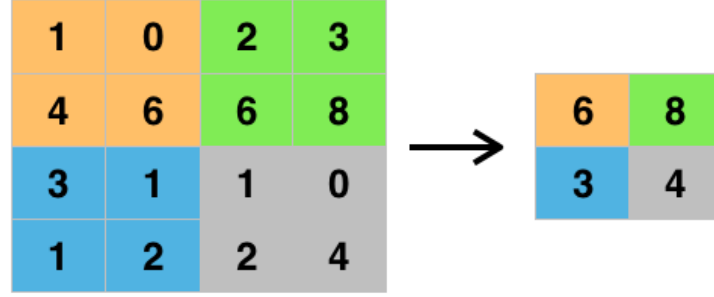


Figure C.33: Max pooling [3].

C.8.2 Bayesian Neural Network

Deep neural networks (DNNs) are usually trained using a regularized maximum likelihood objective to find a single setting of parameters. However, large flexible models like neural networks can represent many functions, corresponding to different parameter settings, which fit the training data well, yet generalize in different ways. This phenomenon is known as underspecification [58].

Considering all of these different models together can lead to improved accuracy and uncertainty representation. This can be done by computing the posterior predictive distribution using Bayesian model averaging. The main challenges in applying Bayesian inference to DNNs are specifying suitable priors, and efficiently computing the posterior, which is challenging due to the large number of parameters and the large datasets [58].

$$p(y|\mathbf{x}, \mathcal{D}) = \int p(y|\mathbf{x}, \boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathcal{D}) d\boldsymbol{\theta} \quad \text{where} \quad p(\boldsymbol{\theta}|\mathcal{D}) \propto p(\boldsymbol{\theta}) p(\mathcal{D}|\boldsymbol{\theta}). \quad (\text{C.87})$$

Consider the generalized MLP with $L - 1$ hidden layers and a linear output as described in Equation C.76 [58].

$$f(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{W}_L(\cdots \varphi(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)) + \mathbf{b}_L. \quad (\text{C.88})$$

The most common choice of priors is to use a factored Gaussian prior [58].

$$\mathbf{W}_\ell \sim \mathcal{N}(\mathbf{0}, \alpha_\ell^2 \mathbf{I}), \quad \mathbf{b}_\ell \sim \mathcal{N}(\mathbf{0}, \beta_\ell^2 \mathbf{I}). \quad (\text{C.89})$$

Initializing this model’s parameters at a particular random value is like sampling a point from this prior over functions. In the limit of infinitely wide neural networks, a neural network defines a Gaussian process (see section C.7) with a fixed kernel. Indeed, an MLP with one hidden layer, whose width goes to infinity, and which has a Gaussian prior on all the parameters, converges to a Gaussian process with a well-defined kernel. These kernels can be used in modeling non-stationary covariance structure [58].

Monte Carlo dropout (MCD) is a very simple and widely used method for approximating the Bayesian predictive distribution. Usually stochastic dropout layers are added as a form of regularization during training, and are turned off at test time. However, the idea of MCD is to also perform random sampling during testing [58].

More precisely, we drop out each hidden unit according to a Bernoulli(p) distribution, and repeat this procedure K times to create K distinct models. We then create an equally weighted average of the predictive distribution for each of these models as shown below. One drawback of MCD is that it is slow at test time. However, this can be overcome by distilling the model’s predictions into a deterministic student network [58].

$$p(y|\mathbf{x}, \mathcal{D}) \approx \frac{1}{K} \sum_{k=1}^K p(y|\mathbf{x}, \boldsymbol{\theta}^k). \quad (\text{C.90})$$

Another very simply approximation to the posterior is to only be Bayesian about the weights in the final layer. This is called the “Bayesian last layer” approximation. In more detail, let $\mathbf{z} = f(\mathbf{x}; \boldsymbol{\theta})$ be the predicted outputs of the model before any optional final nonlinearity. We assume this has the form $\mathbf{z} = \mathbf{w}_L^\top \phi(\mathbf{x}; \boldsymbol{\theta})$, where $\phi(\mathbf{x})$ are the features extracted by the first $L - 1$ layers. Then, we can use standard techniques, such as Markov chain Monte Carlo (described by section C.1), to compute $p(\mathbf{w}_L|\mathcal{D}) = \mathcal{N}(\boldsymbol{\mu}_L, \boldsymbol{\Sigma}_L)$, given $\phi(\cdot)$ [58].

MCMC methods like Hamiltonian Monte Carlo (HMC) are generally considered to be the

gold standard for posterior approximation, since they do not make strong assumptions about the form of the posterior. However, a significant limitation of standard MCMC procedures, including HMC, is that they require access to the full training set at each step. Stochastic gradient MCMC methods, such as Stochastic gradient Langevin dynamics (SGLD), operate instead using mini-batches of data, offering a scalable alternative [58].

Many conventional approximate inference methods, such as variational inference, focus on approximating the posterior $p(\boldsymbol{\theta}|\mathcal{D})$ in a local neighborhood around one of the posterior modes. While this is often not a major limitation in classical machine learning, modern deep neural networks have highly multi-modal posteriors, with parameters in different modes giving rise to very different functions [58].

On the other hand, the functions in a neighborhood of a single mode may make fairly similar predictions. So using a local approximation to compute the posterior predictive will underestimate uncertainty and generalize more poorly. A simple alternative method is to train multiple models, and then to approximate the posterior using an equally weighted mixture of delta functions as follows where M is the number of models. This approach is called deep ensembles [58].

$$p(\boldsymbol{\theta}|\mathcal{D}) \approx \frac{1}{M} \sum_{m=1}^M \delta(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_m). \quad (\text{C.91})$$

Once we have an approximation $q(\boldsymbol{\theta}|\mathcal{D})$ of the parameter posterior $p(\boldsymbol{\theta}|\mathcal{D})$, we can use it to approximate the posterior predictive distribution [58].

$$p(y|\mathbf{x}, \mathcal{D}) = \int p(y|\mathbf{x}, \boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathcal{D}) d\boldsymbol{\theta}. \quad (\text{C.92})$$

We often approximate this integral using Monte Carlo where $\boldsymbol{\theta}^s \sim q(\boldsymbol{\theta}|\mathcal{D})$ [58].

$$p(y|\mathbf{x}, \mathcal{D}) \approx \frac{1}{S} \sum_{s=1}^S p(y|\mathbf{x}, \boldsymbol{\theta}^s). \quad (\text{C.93})$$

Appendix D

METHODS: EVALUATION

D.1 Cross-Validation

D.1.1 Training Phase

Performance should not be evaluated using the same data that was used for training. Therefore, the first step is to split the data into a training set and a testing set. This should be done before starting work on the data, whether it is training an ML model or even doing simple statistics for identifying interesting features. Note that we want the output variable distribution to be approximately the same in the training and testing sets. This is called stratification [80].

The split between train and test sets is arbitrary. With the same machine learning algorithm, two different data splits will lead to two different observed performances, both of which are noisy estimates of the expected generalization performance of prediction models built with this learning procedure. A common strategy to obtain better estimates consists of performing multiple splits of the whole dataset into training and testing sets, a so-called cross-validation loop [80].

For each split of cross-validation, a model is trained using the training set, and the performances are computed using the testing set. The performances over all the testing sets are then aggregated. k -fold cross-validation consists of splitting the data into k sets called folds of approximately equal size. It ensures that each sample in the dataset is used exactly once for testing. Splitting out 10-20% for the testing set is a good trade-off, which amounts to $k = 5$ or 10 [80].

Often an additional validation set is useful to make choices on the model to maximize prediction performance, such as, make changes on the architecture, tune hyperparameters,

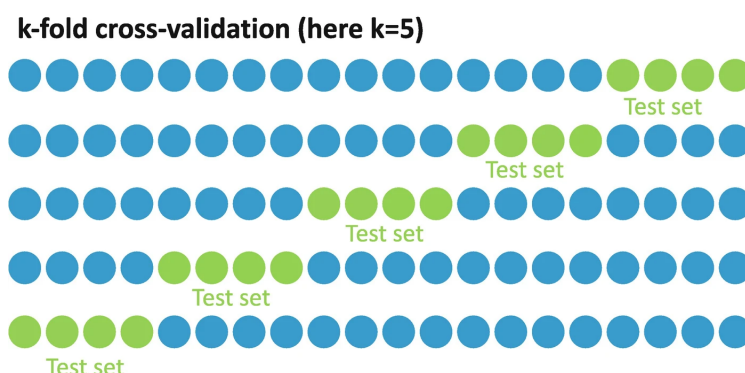


Figure D.1: Example of 5-fold cross-validation [80].

or perform early stopping. As the test set performance is our best estimate of prediction performance, we could double the test set as a validation set. However, in such a situation, the performances reported on the testing set will have an optimistic bias, since a data dependent choice has been made on this test set. There are two main solutions to this issue [80].

The first one is usually applied when the model training is fast and the dataset is of small size. It is called nested cross-validation. It consists of running two loops of cross-validation, one nested into the other. The inner loop serves for hyperparameter tuning or model selection, while the outer loop is used to evaluate the performance. Nested cross-validation should be preferred for very small datasets as it gives better testing power [80].

The second solution is to separate from the whole dataset the test set, which will only be used to evaluate the performances. Then, the remainder of the dataset can be further split into training data and data used to make modeling choices, called the validation set. The training and validation sets will be used in a cross-validation manner, and to experiment with different models and tune parameters. It is absolutely crucial that the test set is isolated at the very beginning, before any experiment is done. It should be left untouched and used only at the end of the study to report the performances [80].

Data leakage denotes cases where some information from the training set has leaked into the test set. As a consequence, the estimation of the performances is likely to be

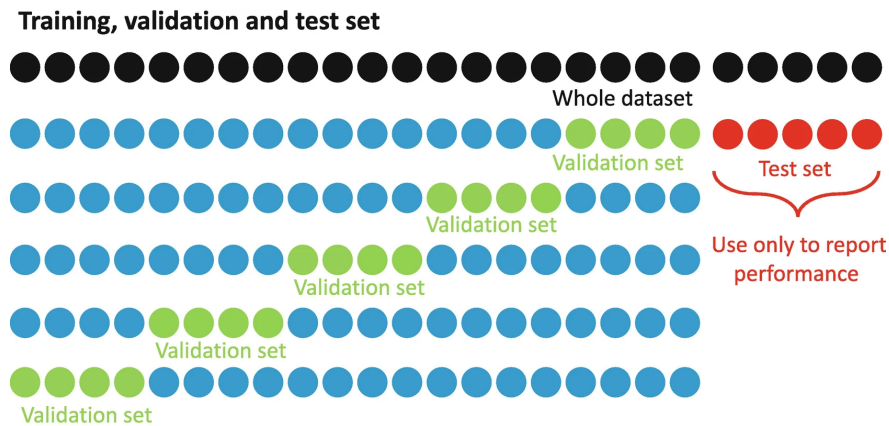


Figure D.2: Example of cross-validation with training, validation, and test sets [80].

optimistic. A first basic cause of data leakage is to use the whole dataset for performing various operations on the data. A very common example is to perform feature selection using the whole dataset and then to use the selected feature for model training. A similar situation is when dimensionality reduction is performed on the whole dataset [80].

D.1.2 Statistical Testing

Internal Validity

Train and test splits, cross-validation, and the like seek to estimate the expected generalization performance of a learning procedure. Keeping test data rigorously independent from algorithm development minimizes the bias of this estimation. However, there are multiple sources of arbitrary variations in these estimates. A systematic study of machine learning benchmarks shows that their most important sources of variance are as follows [80].

- Choice of test split: A given test set is an arbitrary sample of the actual population that we are trying to generalize to. As a result, the corresponding measure of performance is an imperfect estimate of the actual expected performance. Using multiple splits, and thus multiple test sets, improves the estimation, though it makes computing confidence

intervals hard [80].

- Hyperparameter optimization: The choice of hyperparameters is imperfect, for example, because of limited resources to tune there hyperparameters. Another attempt to tune hyperparameters would lead to a slightly different choice. Thus, benchmarks do not give an absolute characterization of a learning procedure but are muddled by imperfect hyperparameters [80].
- Random seeds: Random choices in a learning procedure, such as initial weights, random dropout for neural networks, or bootstraps in bagging, lead to uncontrolled fluctuations in benchmarking results that do not characterize the ability of the algorithm to generalize to new data [80].

With all these sources of arbitrary variance, the question is whether given benchmarks of a learning procedure performance, or improvement, is likely to generalize reliably to new data or rather to be due to benchmarking fluctuations. To answer this question, we must account for estimation error in the expected generalization performance from the different sources of uncontrolled variance in the benchmarks. A simple statistical testing procedure can be described as follows [80].

Training and testing a prediction pipeline multiple times is needed to estimate the variability of the performance measure. The simplest solution is to do this several times while varying the arbitrary factors, such as split between the train and test sets or random initialization. The resulting set of performance measures is similar to bootstrap samples and can be used to draw conclusions on the distribution of performances in a test set. Confidence intervals can be computed using percentiles of this distribution. Two learning procedures can be compared by counting the number of times that one outperforms the other. Outperforming 75% of the times is typically considered as a reliable improvement [80].

Note these procedures do not perform classic null hypothesis significance testing, which is difficult here. In particular, the standard error across the various runs should not be used

instead of the standard deviation. The standard error is the standard deviation divided by the number of runs. The number of runs can be made arbitrarily large given enough compute power, thus making the standard error arbitrarily small. But in no way does the uncertainty due to the limited test data vanish [80].

Furthermore, in repeated splits or cross-validation, the runs are not independent. Specifically, it is invalid to use a standard hypothesis test, such as a t -test, across the different folds of a cross-validation. Another reason not to rely on null hypothesis testing is that their statistical significance only asserts that the expected performance, or improvement, is non-zero over a test population of infinite size. From a practical perspective, we care about meaningful improvements on tests sets of finite size, which is related to the notion of acceptance tests, as opposed to significance, in the Neyman-Pearson framework of statistical testing [80].

External Validity

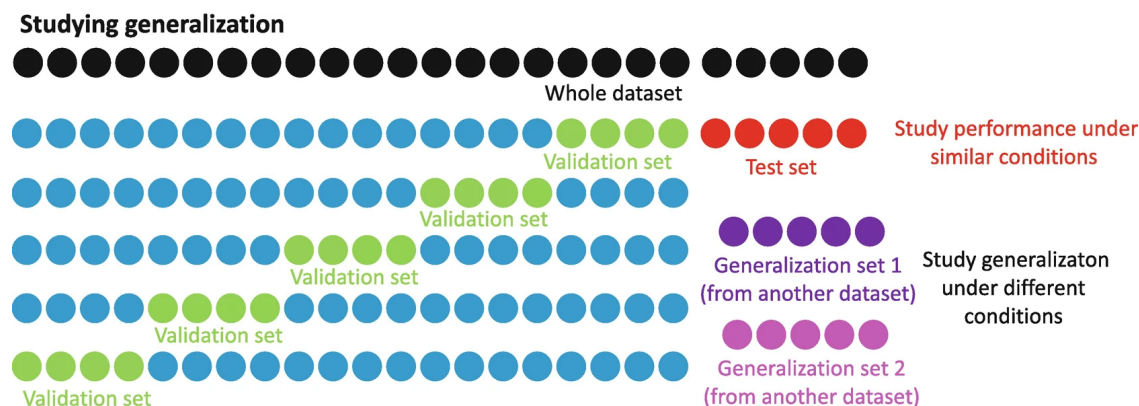


Figure D.3: External validation example that performs out-of-distribution (OOD) tests with generalization sets from another dataset [80].

The procedures described above characterize the expected error of a learning procedure applied on a given population. A related but different question is that of characterizing

the error of a given predictive model on a study population. This is related to the notion of external validity, and is important for two reasons. First, it characterizes the specific predictive model that will be used in practice. Second, characterizing the model on the target population may be important, as it may differ markedly from the study population [80].

Indeed, the techniques in the previous description rely on splitting the initial dataset into training and testing (or validation) sets. Hence, these different sets are by construction drawn from the same population and have similar characteristics. They only demonstrate the ability of the model to generalize to new but similar data. For example, to better assess model utility, guidelines on evaluating clinical prediction models insist on external validation using data collected later in time, or in a different geographical area [80].

Testing whether a prediction model can generalize to dissimilar data is important as it is all too frequent that the study sample, on which the model was developed, does not represent the target population. Participants from on study may not be representative of the target population. This can be due to inclusion/exclusion criteria or due to uncontrolled biases. To assess generalization ability, a common practice is to use one or several additional datasets for testing, these datasets being acquired using different protocols and at different sites [80].

External validation of a predictive model relies on an independent test set and not cross-validation. Statistical testing thus amounts to deriving confidence intervals or null hypothesis significance testing for the metric of interest on this test set. A general and good option, applicable to all situations, is to approximate the sampling distribution of the metric of interest by bootstrapping the test set. When comparing two classifiers, a McNemar's test can be useful to test whether the observed difference in errors can be explained solely by sampling noise [80].

Appendix E

METHODS: ANALYSIS

E.1 Meta-Analysis

E.1.1 Motivation

Science is generally assumed to be a cumulative process, and never in history have we had access to more evidence in the form of published research articles than we do today. In principle, this development should make us enthusiastic about the prospects of science. If science is cumulative, then more published research equals more evidence. Yet, it is not that easy. It is an inconvenient truth that the scientific process, when left to its own devices, will not automatically move us to the best of all possible worlds [39].

Meta-analysis is helpful for critically appraising bodies of evidence in their entirety. That is, as long as we acknowledge its own limitations and biases. A meta-analysis can be described as an analysis of analyses. In conventional studies, the units of analysis are a number people, places, or objects. In meta-analysis, primary studies themselves become the elements of analysis. The aim of meta-analysis is to combine, summarize, and interpret all available evidence related to a clearly defined research field or question. There are at least three distinct ways that evidence from multiple studies can be synthesized [39].

Traditional or narrative reviews are often written by experts and authorities of a research field. There are no strict rules on how studies in a narrative review have to be selected and how to define the scope of the review. There are also no fixed rules on how to draw conclusions from the reviewed evidence. Overall, this can lead to biases favoring the opinion of the author. Nevertheless, when written in a balanced way, narrative reviews can be helpful for readers to get an overall impression of the relevant research questions and available evidence of a field [39].

Systemic reviews try to summarize evidence using clearly defined and transparent rules. Research questions are determined beforehand, and there is an explicit, reproducible methodology through which studies are selected and reviewed. System reviews aim to cover all available evidence. They also assess the validity of evidence using predefined standards and present a synthesis of outcomes in a systemic way [39].

Meta-analyses can be seen as an advanced type of systemic review. However, there is one aspect which makes meta-analyses special. Meta-analyses aim to combine results from previous studies in a quantitative way. The goal of meta-analyses is to integrate quantitative outcomes reported in the selected studies into one numerical estimate. Compared to systemic reviews, meta-analyses often have to be more exclusive concerning the kind of evidence that is summarized. To perform a meta-analysis, it is usually necessary that studies used the same design and type of measurement, and delivered the same intervention [39].

Synthesis Limitations

Certain common pitfalls of meta-analyses can be associated with systemic problems of the scientific process, while others can be traced back to flaws of meta-analyses themselves [39].

- The Apples and Oranges problem: Even with the strictest inclusion criteria, studies in a meta-analysis will never be absolutely identical. Meta-analysis means to calculate a numerical estimate which represents the results of all studies. Such an estimate becomes meaningless when studies do not share the properties that matter to a specific research question. Results of a meta-analysis allow us to pool effects and quantify if and how much this effect may vary across different settings. Variation between studies can often be unproblematic, and even insightful if it is correctly incorporated into the aims and problem specification of a meta-analysis [39].
- The Garbage In, Garbage Out problem: The quality of evidence produced by a meta-analysis heavily depends on the quality of the studies it summarizes. If the results reported in our included findings are biased, or simply incorrect, the results of the

meta-analysis will be equally flawed. This problem can be mitigated to some extent by assessing the quality or risk of bias of the included studies. Though, even the most rigorous meta-analysis will not always be able to balance this out. However, even such disappointing outcomes can be informative, and help guide future research [39].

- The File Drawer problem: The file drawer problem refers to the issue that not all relevant research findings are published, and therefore missing in our meta-analysis. Not being able to integrate all evidence in a meta-analysis would be undesirable, but at least tolerable if we could safely assume that research findings are missing at random in the published literature. Unfortunately, they are not, there is good reason to believe that studies with negative or disappointing results are systematically underrepresented in the published literature and that there is a so called publication bias [39].
- The Researcher Agenda problem: When defining the scope of a meta-analysis, searching and selecting studies, and ultimately pooling outcomes measures, researchers have to make a myriad of choices. Experts are deeply invested in the research area they are examining. As such, they may hold strong opinions about certain topics, and may intentionally or unintentionally influence the results in the direction that fits their beliefs. One way to reduce the researcher agenda problem is pre-registration of a detailed analysis plan before beginning with data collection [39].

E.1.2 Effect Sizes

To perform a meta-analysis, we have to find an effect size which can be summarized across all studies. In particular, the selected effect size measure for a meta-analysis should be comparable, computable, reliable, and interpretable. An effect size is defined as a metric quantifying the relationship between two entities. It captures the direction and magnitude of this relationship. For example, correlations describe how well we can predict the values of a variable through the values of another and so can be seen as a form of effect size [39].

In mathematical notation, it is common to use the Greek letter θ as the symbol for a true effect size. More precisely, θ_k represents the true effect size of a study k . Importantly, the true effect size is not identical with the observed effect size that we find in the published results of the study. The observed effect size $\hat{\theta}$ is only an estimate of the true effect size. The observed effect size of study k is therefore $\hat{\theta}_k$. Note that $\hat{\theta}_k$ differs from θ_k because of the sampling error ϵ_k [39].

$$\hat{\theta}_k = \theta_k + \epsilon_k. \quad (\text{E.1})$$

We can assume that studies with smaller ϵ will deliver a more precise estimate of the true effect size. When pooling the results of different studies, meta-analyses give studies with greater precision (less sampling error) a higher weight, because they are better estimators of the true effect. The question is how we can know how large the sampling error is [39].

Unsurprisingly, the true effect of a study θ_k is unknown, and so ϵ_k is also unknown. Often, however, we can approximate the expected sampling error. A common way to represent ϵ is through the standard error (SE). The standard error is defined as the standard deviation of the sampling distribution. A sampling distribution is the distribution of a metric we get when we draw random samples with the same sample size n from our population many times [39].

Within-Group Study

The arithmetic mean is probably the most commonly used central tendency measure, and can easily be pooled in a meta-analysis [39].

$$\bar{x} = \frac{\sum_{i=1}^n x_i}{n}. \quad (\text{E.2})$$

The standard error of the mean is calculated as the ratio of the sample standard deviation s and the square root of the sample size n [39].

$$SE_{\bar{x}} = \frac{s}{\sqrt{n}}. \quad (\text{E.3})$$

Between-Group Study

The between-group mean difference MD_{between} is defined as the raw, unstandardized difference of the means for two independent groups. Between-group mean differences can be calculated when a study contained at least two groups, as is usually the case in controlled trials or other types of experimental studies. In meta-analyses, mean differences can only be used when all the studies measured the outcome of interest on exactly the same scale. The mean difference is defined as the mean of group 1, $\bar{x}_1$, minus the mean of group 2, $\bar{x}_2$ [39].

$$MD_{\text{between}} = \bar{x}_1 - \bar{x}_2. \quad (\text{E.4})$$

The standard error can be obtained using the following formula where n_1 represents the sample size of group 1, n_2 the sample size of group 2, and s_{pooled} the pooled standard deviation of both groups [39].

$$SE_{MD_{\text{between}}} = s_{\text{pooled}} \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}. \quad (\text{E.5})$$

Using the standard deviation of group 1, s_1 , and group 2, s_2 , the value of s_{pooled} can be calculated as follows [39].

$$s_{\text{pooled}} = \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{(n_1 - 1) + (n_2 - 1)}}. \quad (\text{E.6})$$

Standardized Between-Group Study

The standardized between-group mean difference SMD_{between} is defined as the difference in means between two independent groups, standardized by the pooled standard deviation s_{pooled} . SMD_{between} can be compared between studies, even if those studies did not measure the outcome of interest using the same instruments. In the literature, the standardized mean

difference is also often called Cohen's d . In contrast to unstandardized mean differences, SMD_{between} expresses the difference between two groups in units of standard deviations. This can be achieved by dividing the raw mean difference of two groups, $\bar{x}_1$ and $\bar{x}_2$, through the pooled standard deviation s_{pooled} of both groups [39].

$$SMD_{\text{between}} = \frac{\bar{x}_1 - \bar{x}_2}{s_{\text{pooled}}}. \quad (\text{E.7})$$

Standardized mean differences are often interpreted using the conventions of Cohen, however, these are rules of thumb at best. It is usually much better to interpret standardized mean differences based on their implications. For example, for many serious diseases even a very small statistical effect can still have a large impact on the population level [39].

- $SMD \approx 0.20$: small effect.
- $SMD \approx 0.50$: moderate effect.
- $SMD \approx 0.80$: large effect.

The standardized error of SMD_{between} can be calculated using this formula where n_1 and n_2 are the sample sizes of group 1 and group 2 [39].

$$SE_{SMD_{\text{between}}} = \sqrt{\frac{n_1 + n_2}{n_1 n_2} + \frac{SMD_{\text{between}}^2}{2(n_1 + n_2)}}. \quad (\text{E.8})$$

Effect Size Correction

The standardized mean difference has been found to have an upward bias when the sample of a study is small, especially when $n \leq 20$. This small sample bias means that SMDs systematically overestimate the true effect size when the total sample size of a study is small, which is unfortunately often the case in practice. It is therefore sensible to correct the standardized mean differences of all included studies for small-sample bias, which produces

an effect size called Hedges' g . In this formula, n represents the total sample size of the study [39].

$$g = SMD \times \left(1 - \frac{3}{4n - 9}\right). \quad (\text{E.9})$$

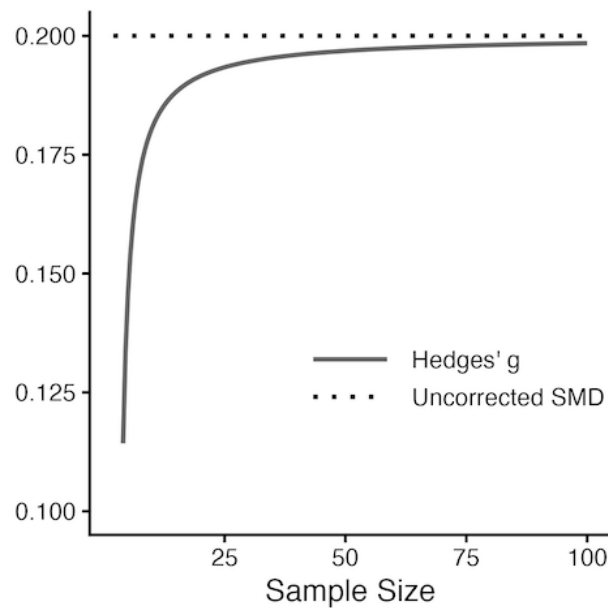


Figure E.1: Example of Hedges' g and Cohen's d over sample size.

E.1.3 Pooling Effect Sizes

Fixed-Effect Model

The fixed-effect model assumes that all effect sizes stem from a single, homogeneous population. It states that all studies share the same true effect size. According to the fixed-effect model, the only reason why a study k 's observed effect size $\hat{\theta}_k$ deviates from θ is because of its sampling error ϵ_k . This sampling error causes the observed effect to deviate from the overall, true effect. We can describe the relationship as the following [39].

$$\hat{\theta}_k = \theta + \epsilon_k. \quad (\text{E.10})$$

In the formula of the fixed-effect model, the true effect size is symbolized by θ , not θ_k as in Equation E.1. Previously, we only made statements about the true effect size of one individual study k . The fixed-effect tells us that a study's true effect size θ_k , and the overall, pooled effect size θ , are identical. The formula of the fixed-effect models tells us that there is only one reason why observed effect sizes θ_k deviate from the true overall effect: because of the sampling error ϵ_k . Furthermore, all things being equal, as the sample size becomes larger, the sampling error becomes smaller [39].

If we want to calculate the pooled effect size under the fixed-effect model, we therefore simply use a weighted average of all studies. To calculate the weight w_k for each study k , we can use the standard error, which we square to obtain the variance s_k^2 of each effect size. Since a lower variance indicates higher precision, the inverse of the variance is used to determine the weight of each study [39].

$$w_k = \frac{1}{s_k^2}. \quad (\text{E.11})$$

Once we know the weights, we can calculate the weighted average, our estimate of the true pooled effect $\hat{\theta}$. This method is the most common approach to calculate average effects in meta-analyses. Since we use the inverse of the variance, it is often called inverse-variance weighting, or simply inverse-variance meta-analysis [39].

$$\hat{\theta} = \frac{\sum_{k=1}^K \hat{\theta}_k w_k}{\sum_{k=1}^K w_k}. \quad (\text{E.12})$$

Random-Effects Model

The fixed-effect model assumes that all our studies are part of a homogeneous population. However, it is simply unrealistic that studies in a meta-analysis are always completely homogeneous. It is likely that we can anticipate considerable between-study heterogeneity in

the true effects. The random-effects model assumes that there is not only one true effect size but a distribution of true effect sizes. The goal of the random-effects model is therefore not to estimate the one true effect size of all studies, but the mean of the distribution of true effects [39].

Similar to the fixed-effect model, the random-effects model starts by assuming that an observed effect size $\hat{\theta}_k$ is an estimator of the study's true effect size θ_k , burdened by sampling error ϵ_k [39].

$$\hat{\theta}_k = \theta_k + \epsilon_k. \quad (\text{E.13})$$

The fact that we use θ_k instead of θ is an important difference. The random-effects model only assumes that θ_k is the true effect size of one single study k . It stipulates that there is a second source of error, denoted by ζ_k . This second source of error is introduced by the fact that even the true effect size θ_k of study k is only part of an over-arching distribution of true effect sizes with mean μ [39].

$$\theta_k = \mu + \zeta_k. \quad (\text{E.14})$$

The random-effects model tells us that there is a hierarchy of two processes. The observed effect sizes of a study deviate from their true value because of the sampling error. But even the true effect sizes are only a draw from a universe of true effects, whose mean μ we want to estimate as the pooled effect of our meta-analysis. We can express the random-effects model in one line. This formula makes it clear that our observed effect size deviates from the pooled effect μ because of two error terms, ζ_k and ϵ_k [39].

$$\hat{\theta}_k = \mu + \zeta_k + \epsilon_k. \quad (\text{E.15})$$

A crucial assumption of the random-effects model is that the size of ζ_k is independent of k . That is, the size of ζ_k is a product of chance, and chance alone. This is known as the exchangeability assumption of the random-effects model. All true effect sizes are assumed

to be exchangeable in so far as we have nothing that could tell us how big ζ_k will be in some study k before seeing the data [39].

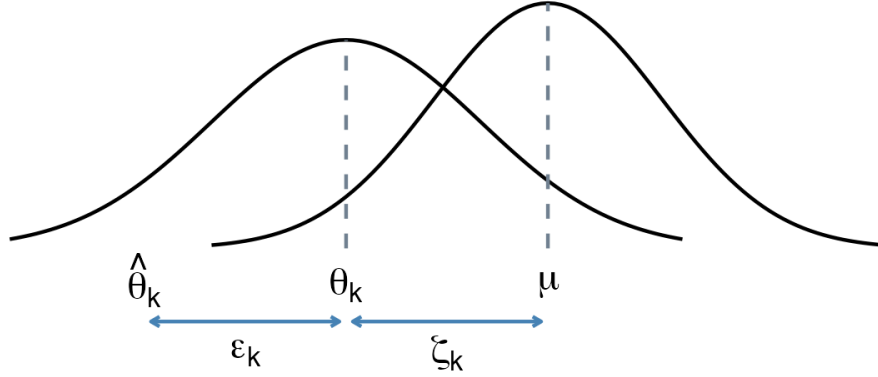


Figure E.2: Illustration of parameters from the random-effects model.

The challenge associated with the random-effects model is that we have to take the error ζ_k into account. To do this, we have to estimate the variance of the distribution of true effect sizes. This variance is known as τ^2 . Once we know the value of τ^2 , we can include the between-study heterogeneity when determining the inverse-variance weight of each effect size. In the random-effects model, we therefore calculate an adjusted random-effects weight w_k^* for each observation [39].

$$w_k^* = \frac{1}{s_k^2 + \tau^2}. \quad (\text{E.16})$$

Using the adjusted random-effects weights, we then calculate the pooled effect size using the inverse variance method, just as we do using the fixed-effect model [39].

$$\hat{\theta} = \frac{\sum_{k=1}^K \hat{\theta}_k w_k^*}{\sum_{k=1}^K w_k^*}. \quad (\text{E.17})$$

There are several methods to estimate τ^2 , and it is an ongoing research question which of

these estimators performs best for different kinds of data. Overall, estimators of τ^2 fall into two categories. Some, like the DerSimonian-Laird and Sidik-Jonkman estimator, are based on closed-form expressions. The restricted maximum likelihood, Paule-Mandel estimator, and empirical Bayes estimator find the optimal value of τ^2 through an iterative algorithm [39].

The Knapp-Hartung adjustments try to control for the uncertainty in our estimate of the between-study heterogeneity. While significance tests of the pooled effect usually assume a normal distribution known as Wald tests, the Knapp-Hartung method is based a t -distribution. Applying a Knapp-Hartung adjustment is usually sensible. Several studies have shown that these adjustments can reduce the chance of false positives, especially when the number of studies is small [39].

E.1.4 Between-Study Heterogeneity

The extent to which true effect sizes vary within a meta-analysis is called between-study heterogeneity. For example, the random-effects model assumes that between-study heterogeneity causes the true effect sizes of studies to differ. It therefore includes an estimate of τ^2 , which quantifies this variance in true effects. This allows us to calculate the pooled effect, defined as the mean of the true effect size distribution [39].

High heterogeneity can be caused by the fact that there are two or more subgroups of studies in our data that have a different true effect. In extreme cases, very high heterogeneity can mean that the studies have nothing in common, and that it makes no sense to interpret the pooled effect at all. Every good meta-analysis should not only report an overall effect but also state how trustworthy this estimate is. An essential part of this is to quantify and analyze the between-study heterogeneity [39].

Cochran's Q

When we want to quantify between-study heterogeneity, the difficulty is to identify how much of the variation can be attributed to the sampling error, and how much to true effect size

differences. Traditionally, meta-analysts have used Cochran's Q for this purpose. Cochran's Q is defined as a weighted sum of squares (WSS). It uses the deviation of each study's observed effect $\hat{\theta}_k$ from the summary effect $\hat{\theta}$, weighted by the inverse of the study's variance w_k [39].

$$Q = \sum_{k=1}^K w_k (\hat{\theta}_k - \hat{\theta})^2. \quad (\text{E.18})$$

The amount to which individual effects deviate from the summary effect, the residuals, is squared. Because of the weighting by w_k , the value of Q does not only depend on how much of $\hat{\theta}_k$ deviates from $\hat{\theta}$ but also on the precision of studies. If the standard error of an effect size is low (and thus the precision is high), even small deviations from the summary effect will be given a higher weight, leading to higher values of Q . The value of Q can be used to check if there is excess variation in our data, meaning more variation than can be expected from sampling error alone. If this is the case, we can assume that the rest of the variation is due to between-study heterogeneity [39].

It is assumed that Q will approximately follow a χ^2 distribution with $K - 1$ degrees of freedom where K is the number of studies in our meta-analysis. That is, this assumption holds if effect size differences are only caused by sampling error. Thus the mean of a χ^2 distribution with $K - 1$ degrees of freedom tells us the value of Q we can expect through sampling error alone [39].

Cochran's Q can be used to test if the variation in a meta-analysis significantly exceeds the amount we would expect under the null hypothesis of no heterogeneity. Although Q is commonly used and reported in meta-analyses, it has several flaws. A practical concern is that Q increases both when the number of studies K , and when the precision (the sample size of a study) increases. Therefore, Q and whether it is significant highly depends on the size of a meta-analysis, and thus its statistical power. From this it follows that we should not only rely on the significance of a Q -test when assessing heterogeneity [39].

Higgins & Thompson's I^2 Statistic

The I^2 statistic is directly based on Cochran's Q . It is defined as the percentage of variability in the effect sizes that is not caused by sampling error. I^2 draws on the assumption that Q follows a χ^2 distribution with $K - 1$ degrees of freedom under the null hypothesis of no heterogeneity. It quantifies, in percent, how much the observed value of Q exceeds the expected Q value when there is no heterogeneity. The value of I^2 cannot be lower than 0%, so if Q happens to be smaller than $K - 1$, we simply use 0 instead of a negative value [39].

$$I^2 = \frac{Q - (K - 1)}{Q}. \quad (\text{E.19})$$

It is common to use the I^2 statistic to report the between-study heterogeneity in meta-analyses, and the popularity of this statistic may be associated with the fact that there is a rule of thumb on how we can interpret it [39].

- $I^2 = 25\%$: low heterogeneity
- $I^2 = 50\%$: moderate heterogeneity
- $I^2 = 75\%$: substantial heterogeneity.

The H^2 Statistic

The H^2 statistic is also derived from Cochran's Q , and similar to I^2 . It describes the ratio of the observed variation, measured by Q , and the expected variance due to sampling error. The computation of H^2 is a little more elegant than the one of I^2 because we do not have to artificially correct its value when Q is smaller than $K - 1$. Values greater than one indicate the presence of between-study heterogeneity. Compared to I^2 , it is far less common to find this statistic reported in published meta-analyses [39].

$$H^2 = \frac{Q}{K - 1}. \quad (\text{E.20})$$

Heterogeneity Variance τ^2 & Standard Deviation τ

As previously discussed, τ^2 quantifies the variance of the true effect sizes underlying our data. When we take the square root, we obtain τ , which is the standard deviation of the true effect sizes. A great asset of τ is that it is expressed on the same scale as the effect size metric. The value of τ tells us something about the range of the true effect sizes. For example, we can calculate the 95% confidence interval of the true effect sizes by multiplying τ with 1.96, and then adding and subtracting this value from the pooled effect size [39].

Assessing Heterogeneity

Cochran's Q and whether it is significant highly depends on the size of a meta-analysis, and thus its statistical power. We should therefore not only rely on Q when assessing between-study heterogeneity. I^2 , on the other hand, is not sensitive to changes in the number of studies in the analysis. It is also relatively easy to interpret. It is recommended to include I^2 with confidence intervals as a heterogeneity measure in a meta-analysis report [39].

However, despite its common use in the literature, I^2 is not a perfect measure for heterogeneity either. It still heavily depends on the precision of the included studies. I^2 is simply the percentage of variability not caused by sampling error ϵ . If our studies become increasingly large, the sampling error tends to zero, while at the same time, I^2 tends to 100% simply because the studies have a greater sample size. Since H^2 behaves similarly to I^2 , the same caveats also apply to this statistic [39].

The value of τ^2 , on the other hand, is sensitive to the number of studies, and their precision. Yet, it is often difficult to interpret how relevant the amount of variance τ^2 is from a practical standpoint. Prediction intervals (PIs) are a good way to overcome this limitation, giving us a range into which we can expect the effects of future studies to fall based on present evidence. In addition to reporting I^2 with confidence intervals, one should also report prediction intervals in meta-analyses [39].

To calculate the 95% prediction intervals around the overall effect $\hat{\mu}$, we use both the

estimated between-study heterogeneity variance $\hat{\tau}^2$, as well as the standard error of the pooled effect $SE_{\hat{\mu}}$, to compute the standard deviation of the prediction interval SD_{PI} , using a t -distribution with $K - 1$ degrees of freedom [39].

$$\hat{\mu} \pm t_{K-1,0.975} \sqrt{SE_{\hat{\mu}}^2 + \hat{\tau}^2} = \hat{\mu} \pm t_{K-1,0.975} SD_{PI}. \quad (\text{E.21})$$

E.1.5 Forest Plots

The most common way to visualize meta-analyses is through forest plots. See Figure E.3 for an example created from a dataset containing studies examining the effect of so-called “third wave” psychotherapies on perceived stress in college students. For each study, the standardized mean difference between a treatment and control group at post-test was calculated, and Hedges’ g small sample correction was applied [39].

For each study, a graphical representation of the effect size is provided. This visualization shows the point estimate of a study on the horizontal axis. This point estimate is supplemented by a line, which represents the range of the confidence interval calculated for the observed effect size. Usually, the points estimate is surrounded by a square. The size of this square is determined by the weight of the effect size: Studies with a larger weight are given a larger square [39].

At the bottom of the plot, a diamond shape represents the average effect. The length of the diamond symbolizes the confidence interval of the pooled result on the horizontal axis. Typically, forest plots also include a vertical reference line, which indicates the point on the horizontal axis equal to no effect [39].

E.1.6 Subgroup Analysis

Outlier and influence analysis can help us identify which studies contribute to observed heterogeneity. They are performed after seeing the data, and often because of the results we found. These analyses may tell us that some study does not properly follow the expectations

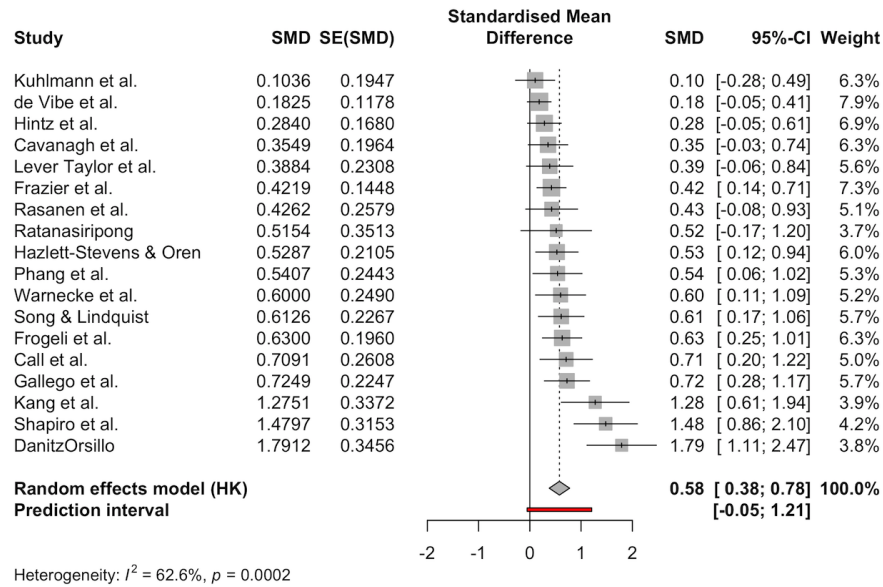


Figure E.3: Example of a forest plot from the Third Wave dataset.

of our model, but not why this is that case. This makes it clear that we need a different approach, one that allows us to identify why a specific heterogeneity pattern can be found in our data [39].

Subgroup analyses, also known as moderator analyses, are one way to do this. They allow us to test specific hypotheses, describing why some type of study produces lower or higher effects than another. The idea behind subgroup analysis is that meta-analysis is not only about calculating an average effect size but that it can also be a tool to investigate variation in our evidence. In subgroup analyses, we see heterogeneity not merely as a nuisance but as interesting variation which may or may not be explainable by a scientific hypothesis [39].

In subgroup analyses, we hypothesize that studies in our meta-analysis do not stem from one overall population. Instead, we assume that they fall into different subgroups and that each subgroup has its own true overall effect. Our aim is to reject the null hypothesis that there is no difference in effect sizes between subgroups. The calculation of a subgroup analysis

consists of two parts. First, we pool the effect in each subgroup. Subsequently, the effects of the subgroups are compared using a statistical test [39].

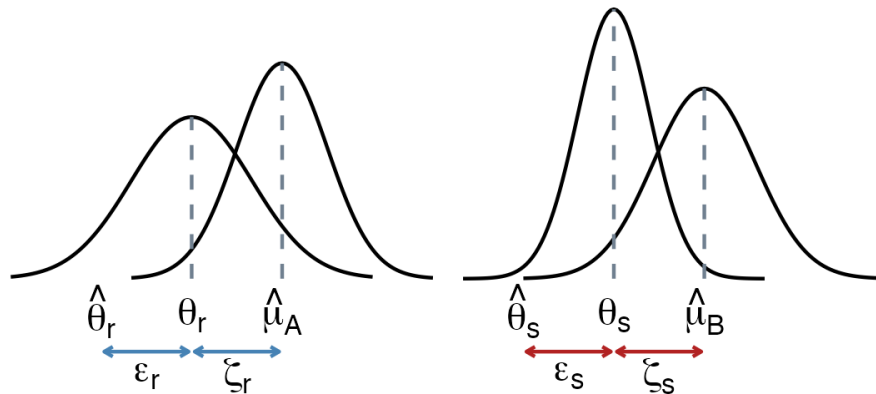


Figure E.4: Illustration of parameters from a subgroup analysis also known as a fixed-effects (plural) model.

Pooling the Effect in Subgroups

The first part is straightforward, as the same criteria as those for a meta-analysis without subgroups apply. If we assume that all studies in a subgroup stem from the same population, and have one shared true effect, we can use the fixed-effect mode. However, as mentioned previously, it is often unrealistic that this assumption holds in practice, even when we partition our studies into smaller subgroups [39].

Therefore, the alternative is to use a random-effects model. This assumes that studies within a subgroup are drawn from a universe of populations, the mean of which we want to estimate. The difference to a normal meta-analysis is that we conduct several separate random-effects meta-analyses, one for each subgroup. This results in a pooled effect $\hat{\mu}_g$ for each subgroup g [39].

Since each subgroup gets its own separate meta-analysis, estimates of the τ^2 heterogeneity will also differ from subgroup to subgroup. In practice, however, the individual heterogeneity values $\hat{\tau}_g^2$ are often replaced with a version of τ^2 that was pooled across subgroups. This means that all subgroups are assumed to share a common estimate of the between-study heterogeneity. This is done for practical reasons. When the number of studies in a subgroup is small, for example $k_g < 5$, it is likely that the estimate of τ^2 will be imprecise. In this case, it is better to calculate a pooled version of τ^2 that is used across all subgroups [39].

Comparing the Subgroup Effects

In the next step, we assess if there is a true difference between the G subgroups. The assumption is that the subgroups are different, meaning that at least one subgroup is part of a different population of studies. A way to test this is to pretend that the pooled effect of a subgroup is nothing more than the observed effect size of one large study. Similar to when assessing the heterogeneity of a normal meta-analysis, we want to know if difference in effect sizes exist only due to sampling error, or because of true differences in the effect sizes [39].

Therefore, we use the value of Q to determine if the subgroup differences are large enough to not be explainable by sampling error alone. Pretending that the subgroup effects are observed effect sizes, we calculate the value of Q . This observed Q value is compared to its expected value assuming a χ^2 distribution with $G - 1$ degrees of freedom. If the observed value of Q is substantially larger than the expected one, the p -value of the Q test will become significant. This indicates that there is a difference in the true effect sizes between subgroups [39].

Borenstein and Higgins (2013) argue that the subgroups we choose to analyze cannot always be seen as random draws from a universe of possible subgroups, but represent fixed levels of a characteristic we want to examine. Take employment status as an example. This features two fixed subgroups, “employed” and “unemployed”. Borenstein and Higgins call the model for subgroup analyses the fixed-effects (plural) model, where the word “plural” is added because we have to differentiate it from the standard fixed-effect model. It is also

known in the literature as a mixed-effects model [39].

E.1.7 Meta-Regression

Usually, regression models are based on data comprising individual persons or specimens, for which both the independent x and dependent y values are measured. In meta-regression, this concept is applied to entire studies. The variable x represents characteristics of studies, for example, the year in which it was conducted. Based on this information, a meta-regression model tries to predict y , the study's effect size [39].

In a conventional regression, we want to estimate the value y_i of a person or specimen i using a predictor (or covariate) x_i with a regression coefficient β [39].

$$y_i = \beta_0 + \beta_1 x_i. \quad (\text{E.22})$$

In meta-regression, the variable y we want to predict is the observed effect size $\hat{\theta}_k$ of study k [39].

$$\hat{\theta}_k = \theta + \beta x_k + \epsilon_k + \zeta_k. \quad (\text{E.23})$$

This formula contains two extra terms, ϵ_k and ζ_k . The same terms found in the random-effects model from Equation E.15. The first one, ϵ_k , is the sampling error through which the effect size of a study deviates from its true effect. The second error, ζ_k , denotes that even the true effect size of the study is only sampled from an overarching distribution of effect sizes. Since the meta-regression formula includes a fixed effect (the β coefficient) as well as a random effect (ζ_k), the model used in meta-regression is often called a mixed-effects model [39].

Meta-Regression with a Categorical Predictor

Subgroup analysis (from subsection E.1.6) is nothing else than a meta-regression with a categorical predictor, where such categorical variables can be included through dummy-

coding, such as the following [39].

$$D_g = \begin{cases} 0 : & \text{Subgroup A} \\ 1 : & \text{Subgroup B} \end{cases} \quad (\text{E.24})$$

To specify a subgroup analysis in the form of a meta-regression, we simply have to replace the covariate x_k with D_g [39].

$$\hat{\theta}_k = \theta + \beta D_g + \epsilon_k + \zeta_k. \quad (\text{E.25})$$

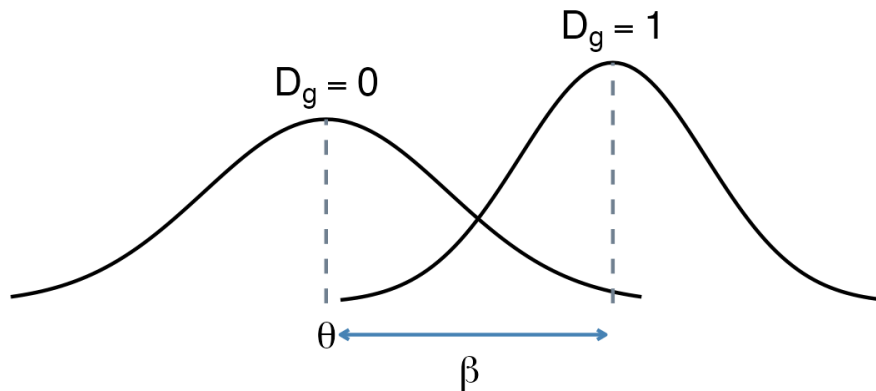


Figure E.5: Meta-regression with a categorical predictor (subgroup analysis).

Meta-Regression with a Continuous Predictor

When people speak of a meta-regression, they usually think of models in which a continuous variable is used as the predictor. The aim of the meta-regression model is to find values of θ and β which minimize the difference between the predicted effect size and the true effect size of studies. Taking into account both the sampling error ϵ_k and between-study heterogeneity

ζ_k , meta-regression tried to find a model that generalizes well, not only to the observed effect sizes but to the universe of all possible studies of interest [39].

An important detail about meta-regression models is that they can be seen as an extension of the random-effects model. The random-effects model is nothing but a meta-regression without a slope term. Since it contains no slope, the random-effects model simply predicts the same value for all studies. That is, the estimate of the pooled effect size μ , which is equivalent to the intercept [39].

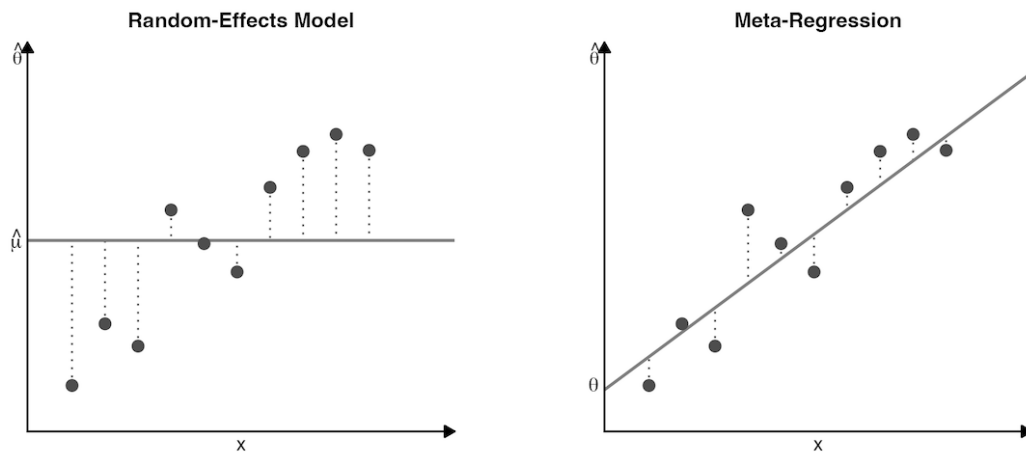


Figure E.6: Example of a random-effects model versus a meta-regression.

In the first step, the calculation of a meta-regression therefore closely resembles the one of a random-effects meta-analysis, in that the between-study heterogeneity τ^2 is estimated. In the next step, the fixed weights θ and β are estimated. In meta-regression, weighted least squares (WLS) is used to find the regression line that fits the data best. WLS makes sure that studies with a smaller standard error are given a higher weight [39].

If the meta-regression model fits the data well, the true effect sizes should deviate less from the regression line compared to the pooled effect $\hat{\mu}$. If this is the case, the predictor x explains some of the heterogeneity variance in our meta-analysis. The fit of the meta-regression model can therefore be assessed by checking how much of the heterogeneity variance it explains [39].

The predictors included in the mixed-effects model should minimize the amount of residual heterogeneity variance denoted by $\hat{\tau}_{\text{MEM}}^2$. The R_*^2 index is used to quantify the percentage of variation explained by the meta-regression model. The asterisk is added to differentiate from the R^2 index used in conventional regressions, since meta-regression deals with true effect sizes instead of observed data points [39].

R_*^2 uses the amount of residual heterogeneity variance that even the meta-regression slope cannot explain. It is put in relation to the total heterogeneity that we initially found in our random-effects model $\hat{\tau}_{\text{REM}}^2$. Subtracting this fraction from 1 leaves us with the percentage of between-study heterogeneity explained by the predictor, and is expressed in Equation E.26. We can also say that it is how much the mixed-effects model has reduced the heterogeneity variance compared to the initial random-effects pooling model, in percent [39].

$$R_*^2 = 1 - \frac{\hat{\tau}_{\text{MEM}}^2}{\hat{\tau}_{\text{REM}}^2} = \frac{\hat{\tau}_{\text{REM}}^2 - \hat{\tau}_{\text{MEM}}^2}{\hat{\tau}_{\text{REM}}^2}. \quad (\text{E.26})$$

Usually, we are not only in the amount of heterogeneity explained by the regression model but also if the regression weight of our predictor x is significant. Both in conventional and meta-regression, the significance of a regression weight is commonly assessed through a Wald test. This involves calculating the test statistic z , by dividing the estimate of β through its standard error [39].

$$z = \frac{\beta}{SE_{\hat{\beta}}}. \quad (\text{E.27})$$

Under the null hypothesis that $\beta = 0$, this z -statistic follows a standard normal distribution. This allows us to calculate a corresponding p -value. Like in normal meta-analysis models, we can also use the Knapp-Hartung adjustment, which results in a test statistic based on the t -distribution [39].

E.1.8 Multilevel Meta-Analysis

When people talk about multilevel meta-analysis, they are referring to three-level meta-analysis models. A three-level model contains three pooling steps. First, researchers themselves pool the results of individual participants in their primary studies, and report the aggregated effect size. Then, on level 2, these effect sizes are nested within several clusters denoted by κ . These clusters can either be within individual studies or across subgroups of studies. Lastly, pooling the aggregated cluster effects leads to the overall true effect size μ [39].

For an example of dependence introduced within a study, consider scientists conducting a study. They may have collected data from multiple sites, compared multiple interventions to one single control group, or used different questionnaires to measure the same outcome. In all these scenarios, we can assume that some kind of dependency is introduced within the reported study [39].

As an example of dependence introduced across studies, think of a meta-analysis that focuses on some psychological mechanism. This meta-analysis includes studies which were conducted in different cultural regions of the world. Depending on the type of psychological mechanism, it could be that results of studies conducted in the same cultural region are more similar compared to those conducted in a different culture [39].

The formula for the three-level model follows. $\hat{\theta}_{ij}$ is an estimate of the true effect size θ_{ij} . The term ij can be read as some effect size i nested in cluster j . Parameter κ_j is the average effect size in cluster j , and μ the overall average population effect. In contrast to the random-effects model, this formula contains two heterogeneity terms. One is $\zeta_{(2)ij}$, which stands for the within-cluster heterogeneity on level 2. The other is $\zeta_{(3)j}$, the between-cluster heterogeneity on level 3 [39].

$$\text{(Level 1)} \quad \hat{\theta}_{ij} = \theta_{ij} + \epsilon_{ij} \quad (\text{E.28})$$

$$\text{(Level 2)} \quad \theta_{ij} = \kappa_j + \zeta_{(2)ij} \quad (\text{E.29})$$

$$\text{(Level 3)} \quad \kappa_j = \mu + \zeta_{(3)j} \quad (\text{E.30})$$

$$\hat{\theta}_{ij} = \mu + \zeta_{(3)j} + \zeta_{(2)ij} + \epsilon_{ij}. \quad (\text{E.31})$$

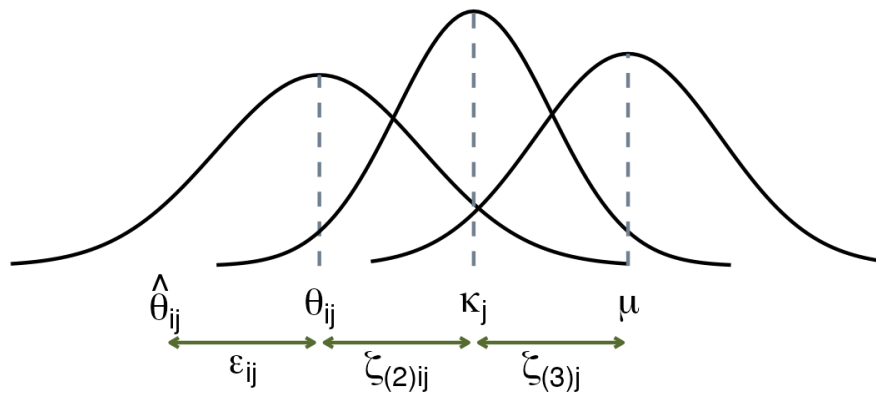


Figure E.7: The parameters of a multilevel meta-analysis from one subgroup cluster.

Subgroup Analyses in Three-Level Models

We can add regression terms to a multilevel model, which leads to three-level mixed-effects model where θ is the intercept and β is the regression weight of a predictor variable x [39].

$$\hat{\theta}_{ij} = \theta + \beta x_i + \zeta_{(3)j} + \zeta_{(2)ij} + \epsilon_{ij}. \quad (\text{E.32})$$

Robust Variance Estimation

A hierarchical model can provide a better representation of a dataset than a conventional meta-analysis, which assumes that all effect sizes are independent. But it is still a simplifi-

cation of reality. In practice, there are often forms of dependence between effect sizes that are more complex than what we have captured so far in nested models. When several effect sizes in one study are based on the same sample, we expect their sampling errors (the ϵ_{ij} terms in Equation E.31) to be correlated [39].

The extended three-level architecture, the correlated and hierarchical effects (CHE) model, explicitly takes into account that some effect sizes within clusters are based on the same sample, and that their sampling errors are therefore correlated. In combination with the CHE model, the robust variance estimation (RVE) and its so-called sandwich estimator can be used to obtain robust confidence intervals and p -values [39].

For the robust variance estimator of a meta-regression of the conventional random-effects model, we can rewrite Equation E.23 so that it models dependent effect sizes, and is expressed in matrix notation [39].

$$\mathbf{t}_j = \mathbf{X}_j \boldsymbol{\beta} + \mathbf{u}_j + \mathbf{e}_j. \quad (\text{E.33})$$

Equation E.33 tells us that effect sizes $\mathbf{t}_j$ are predicted by regression weights $\boldsymbol{\beta}$ associated with certain covariates $\mathbf{X}_j$. It also tells us that, besides the sampling error $\mathbf{e}_j$, there are random effects for each study $\mathbf{u}_j$, thus producing a mixed-effects meta-regression model. Say that n_j is the number of effect sizes in some study j . The effect sizes in study j can then be written as the column vector $\mathbf{t}_j = (t_{j,1}, \dots, t_{j,n_j})^\top$. Similarly, $\mathbf{X}_j$ is the design matrix containing the covariate values of some study j , where $p - 1$ is the total number of covariates. The vector of regression coefficients $\boldsymbol{\beta} = (\beta_1, \dots, \beta_p)^\top$ contains no subscript j , since it is assumed to be fixed across all studies [39].

$$\mathbf{X}_j = \begin{bmatrix} x_{1,1} & \cdots & x_{1,p} \\ \vdots & \ddots & \vdots \\ x_{n_j,1} & \cdots & x_{n_j,p} \end{bmatrix}. \quad (\text{E.34})$$

Based on this formula, we can estimate the meta-regression coefficients $\hat{\boldsymbol{\beta}}$. To calculate confidence intervals and conduct significance tests of the coefficients, we need an estimate

of their variance $\mathbf{V}_{\hat{\beta}}$. This can be achieved using the robust sampling variance estimator otherwise known as the sandwich estimator [39].

$$\mathbf{V}_{\hat{\beta}} = \left(\sum_{j=1}^J \mathbf{X}_j^{\top} \mathbf{W}_j \mathbf{X}_j \right)^{-1} \left(\sum_{j=1}^J \mathbf{X}_j^{\top} \mathbf{W}_j \mathbf{A}_j \mathbf{\Phi}_j \mathbf{A}_j \mathbf{W}_j \mathbf{X}_j \right) \left(\sum_{j=1}^J \mathbf{X}_j^{\top} \mathbf{W}_j \mathbf{X}_j \right)^{-1}. \quad (\text{E.35})$$

The true dependence structure of the effect sizes in some study j is described by the $n_j \times n_j$ covariance matrix $\mathbf{\Phi}_j$. Unfortunately, it is rarely known to what extent effect sizes are correlated within and across studies. Therefore, it is necessary to make a few simplifying assumptions in our model. The CHE model, assumes that there is a known correlation ρ between effect sizes in the same study, and that ρ has the same value across all studies in our meta-analysis [39].

The weights $\mathbf{W}_j$ of each effect size takes the precision of effect size estimates into account before we can pool them. The optimal way to do this is to take the inverse of the variance $\mathbf{W}_j = \mathbf{\Phi}_j^{-1}$. As mentioned, the true values of $\mathbf{\Phi}_j$ are hardly even known, so an estimate based on our model $\hat{\mathbf{\Phi}}_j^{-1}$ is used [39].

The adjustment matrix $\mathbf{A}_j$ ensures that the estimator provides valid results even when the number of studies in our meta-analysis is small. Say, 40 studies or less. The recommended approach is to use a matrix based on the bias-reduced linearization, or CR2 method [39].

Cluster Wild Bootstrapping

Another, and sometimes favorable way to test coefficients in our model are bootstrapping procedures. The cluster wild bootstrapping (CWB) method is well suited if the total number of studies J in our meta-analysis is small. That is, compared to RVE, which can lead to overly conservative results in small samples [39].

The cluster wild bootstrap is based on the residuals of a null model fitted without any additional covariates. Residuals are transformed using an adjustment matrix $\mathbf{A}_j$, for example based on the CR2 method, to handle dependent effect sizes. A general algorithm for CWB looks like the following. Steps 3 to 5 are repeated R times. The bootstrap p -value can be

derived as the proportion of times the bootstrap test statistic was more extreme than the one based on the original data [39].

The Cluster Wild Bootstrapping Algorithm [39]

1. Calculate the full model based on the original data, and derive the test statistic of interest.
2. Fit a null model based on the original data and extract its residuals e .
3. For each study j , draw a random value from a distribution, and multiply the residuals of j by this random value.
4. Generate new, bootstrapped effect sizes by adding the transformed residuals to the predicted values of the null model based on the original data.
5. Fit the full model again, using the bootstrapped effect sizes, and calculate the test statistic again.

Appendix F

RESULTS

F.1 Posterior Sampling

Table F.1: The summary of posterior sampling.

Summary Diagnostics I							
Dataset	Pipeline	Min. ESS bulk	Min. ESS tail	Max. $\hat{R}$	Max. MCSE mean	Max. MCSE std.	Total divergences
BNCI2014.001	BDCNN	3761.0	2090.0	1.01	0.002	0.002	0.0
BNCI2014.001	BSCNN	3863.0	1981.0	1.01	0.001	0.001	0.0
BNCI2014.001	CSPBLDA	959.0	914.0	1.01	0.003	0.002	0.0
BNCI2014.001	CSPGP	712.0	1395.0	1.01	0.017	0.019	0.0
BNCI2014.001	TSBLR	3268.0	2190.0	1.01	0.001	0.001	0.0
BNCI2014.001	TSGP	1093.0	1995.0	1.01	0.011	0.022	0.0
BNCI2014.004	BDCNN	3910.0	1986.0	1.01	0.001	0.001	0.0
BNCI2014.004	BSCNN	5231.0	1902.0	1.01	0.0	0.001	0.0
BNCI2014.004	CSPBLDA	3190.0	2340.0	1.0	0.0	0.0	0.0
BNCI2014.004	CSPGP	986.0	1574.0	1.01	0.013	0.022	0.0
BNCI2014.004	TSBLR	2371.0	2215.0	1.0	0.001	0.001	0.0
BNCI2014.004	TSGP	549.0	680.0	1.01	0.023	0.027	2.0

Table F.2: The summary of posterior sampling.

Summary Diagnostics II							
Dataset	Pipeline	Min. ESS bulk	Min. ESS tail	Max. $\hat{R}$	Max. MCSE mean	Max. MCSE std.	Total divergences
Brandl2020	BDCNN	3828.0	1839.0	1.01	0.001	0.001	0.0
Brandl2020	BSCNN	6163.0	1987.0	1.01	0.0	0.001	0.0
Brandl2020	CSPBLDA	1307.0	1899.0	1.0	0.001	0.001	0.0
Brandl2020	CSPGP	1091.0	1750.0	1.01	0.012	0.016	0.0
Brandl2020	TSBLR	6168.0	2139.0	1.01	0.001	0.001	0.0
Brandl2020	TSGP	777.0	1571.0	1.01	0.012	0.021	0.0
Chang2025	BDCNN	5489.0	2071.0	1.01	0.001	0.002	0.0
Chang2025	BSCNN	4223.0	2001.0	1.01	0.001	0.001	0.0
Chang2025	CSPBLDA	2254.0	2601.0	1.01	0.001	0.001	0.0
Chang2025	CSPGP	651.0	1357.0	1.01	0.017	0.018	0.0
Chang2025	TSBLR	5397.0	2198.0	1.01	0.001	0.001	0.0
Chang2025	TSGP	1127.0	2282.0	1.01	0.013	0.018	0.0

Table F.3: The summary of posterior sampling.

Summary Diagnostics III							
Dataset	Pipeline	Min. ESS bulk	Min. ESS tail	Max. $\hat{R}$	Max. MCSE mean	Max. MCSE std.	Total divergences
Cho2017	BDCNN	3697.0	1686.0	1.01	0.0	0.001	0.0
Cho2017	BSCNN	3621.0	2106.0	1.01	0.001	0.001	0.0
Cho2017	CSPBLDA	2261.0	2320.0	1.0	0.001	0.0	0.0
Cho2017	CSPGP	568.0	1126.0	1.01	0.019	0.018	0.0
Cho2017	TSBLR	6527.0	2101.0	1.01	0.001	0.001	0.0
Cho2017	TSGP	1079.0	1584.0	1.01	0.013	0.019	0.0
Dreyer2023	BDCNN	3459.0	2003.0	1.01	0.001	0.001	0.0
Dreyer2023	BSCNN	3593.0	2027.0	1.01	0.0	0.001	0.0
Dreyer2023	CSPBLDA	1535.0	2051.0	1.01	0.001	0.0	0.0
Dreyer2023	CSPGP	489.0	523.0	1.01	0.022	0.021	1.0
Dreyer2023	TSBLR	4793.0	2222.0	1.01	0.001	0.001	0.0
Dreyer2023	TSGP	378.0	754.0	1.02	0.018	0.023	0.0

Table F.4: The summary of posterior sampling.

Summary Diagnostics IV							
Dataset	Pipeline	Min. ESS bulk	Min. ESS tail	Max. $\hat{R}$	Max. MCSE mean	Max. MCSE std.	Total divergences
Forenzo2023	BDCNN	4581.0	2181.0	1.01	0.001	0.002	0.0
Forenzo2023	BSCNN	4755.0	1869.0	1.01	0.0	0.001	0.0
Forenzo2023	CSPBLDA	1241.0	1983.0	1.0	0.002	0.001	0.0
Forenzo2023	CSPGP	350.0	198.0	1.01	0.036	0.023	7.0
Forenzo2023	TSBLR	3925.0	2228.0	1.01	0.001	0.001	0.0
Forenzo2023	TSGP	1430.0	1676.0	1.0	0.011	0.02	0.0
GrosseWentrup2009	BDCNN	3916.0	1792.0	1.01	0.001	0.001	0.0
GrosseWentrup2009	BSCNN	4038.0	1988.0	1.01	0.001	0.001	0.0
GrosseWentrup2009	CSPBLDA	1206.0	1470.0	1.0	0.001	0.001	0.0
GrosseWentrup2009	CSPGP	1199.0	1830.0	1.01	0.012	0.018	0.0
GrosseWentrup2009	TSBLR	6531.0	2092.0	1.01	0.0	0.001	0.0
GrosseWentrup2009	TSGP	1118.0	1924.0	1.0	0.012	0.023	0.0

Table F.5: The summary of posterior sampling.

Summary Diagnostics V							
Dataset	Pipeline	Min. ESS bulk	Min. ESS tail	Max. $\hat{R}$	Max. MCSE mean	Max. MCSE std.	Total divergences
GuttmannFlury2025_MI	BDCNN	4462.0	2059.0	1.01	0.001	0.002	0.0
GuttmannFlury2025_MI	BSCNN	4049.0	1980.0	1.01	0.001	0.001	0.0
GuttmannFlury2025_MI	CSPBLDA	3100.0	2485.0	1.0	0.001	0.001	0.0
GuttmannFlury2025_MI	CSPGP	584.0	700.0	1.01	0.035	0.029	0.0
GuttmannFlury2025_MI	TSBLR	6514.0	2041.0	1.01	0.001	0.001	0.0
GuttmannFlury2025_MI	TSGP	2391.0	1613.0	1.01	0.015	0.021	0.0
HefmiIch2025	BDCNN	5703.0	1398.0	1.01	0.0	0.001	0.0
HefmiIch2025	BSCNN	5059.0	1833.0	1.01	0.0	0.001	0.0
HefmiIch2025	CSPBLDA	1453.0	1978.0	1.0	0.002	0.001	0.0
HefmiIch2025	CSPGP	1305.0	1848.0	1.01	0.016	0.024	0.0
HefmiIch2025	TSBLR	4661.0	2174.0	1.01	0.001	0.002	0.0
HefmiIch2025	TSGP	349.0	526.0	1.01	0.042	0.023	0.0

Table F.6: The summary of posterior sampling.

Summary Diagnostics VI							
Dataset	Pipeline	Min. ESS bulk	Min. ESS tail	Max. $\hat{R}$	Max. MCSE mean	Max. MCSE std.	Total divergences
Kumar2024	BDCNN	4443.0	1988.0	1.01	0.001	0.001	0.0
Kumar2024	BSCNN	3747.0	1854.0	1.01	0.0	0.001	0.0
Kumar2024	CSPBLDA	2777.0	2442.0	1.01	0.001	0.0	0.0
Kumar2024	CSPGP	432.0	1032.0	1.01	0.029	0.021	0.0
Kumar2024	TSBLR	3830.0	2600.0	1.0	0.001	0.001	0.0
Kumar2024	TSGP	719.0	1313.0	1.01	0.013	0.023	0.0
Lee2019_MI	BDCNN	3648.0	2139.0	1.01	0.001	0.001	0.0
Lee2019_MI	BSCNN	3550.0	2030.0	1.01	0.0	0.001	0.0
Lee2019_MI	CSPBLDA	2115.0	2367.0	1.0	0.001	0.0	0.0
Lee2019_MI	CSPGP	633.0	1194.0	1.01	0.017	0.018	0.0
Lee2019_MI	TSBLR	6128.0	2101.0	1.01	0.001	0.001	0.0
Lee2019_MI	TSGP	547.0	1064.0	1.01	0.017	0.022	0.0

Table F.7: The summary of posterior sampling.

Summary Diagnostics VII							
Dataset	Pipeline	Min. ESS bulk	Min. ESS tail	Max. $\hat{R}$	Max. MCSE mean	Max. MCSE std.	Total divergences
Liu2024	BDCNN	4310.0	1727.0	1.01	0.001	0.002	0.0
Liu2024	BSCNN	4797.0	1992.0	1.01	0.001	0.001	0.0
Liu2024	CSPBLDA	1533.0	2133.0	1.0	0.002	0.001	0.0
Liu2024	CSPGP	303.0	790.0	1.01	0.037	0.028	1.0
Liu2024	TSBLR	3770.0	2614.0	1.0	0.002	0.002	0.0
Liu2024	TSGP	907.0	1562.0	1.01	0.017	0.024	0.0
PhysionetMI	BDCNN	4899.0	1515.0	1.01	0.0	0.001	0.0
PhysionetMI	BSCNN	4373.0	1935.0	1.01	0.001	0.001	0.0
PhysionetMI	CSPBLDA	1108.0	1447.0	1.0	0.002	0.001	0.0
PhysionetMI	CSPGP	380.0	737.0	1.02	0.023	0.022	0.0
PhysionetMI	TSBLR	6258.0	2040.0	1.01	0.001	0.001	0.0
PhysionetMI	TSGP	1359.0	1918.0	1.01	0.011	0.025	0.0

Table F.8: The summary of posterior sampling.

Summary Diagnostics VIII							
Dataset	Pipeline	Min. ESS bulk	Min. ESS tail	Max. $\hat{R}$	Max. MCSE mean	Max. MCSE std.	Total divergences
Schirrmeister2017	BDCNN	4218.0	2032.0	1.01	0.002	0.003	0.0
Schirrmeister2017	BSCNN	4635.0	1810.0	1.01	0.001	0.001	0.0
Schirrmeister2017	CSPBLDA	1824.0	2336.0	1.01	0.001	0.001	0.0
Schirrmeister2017	CSPGP	530.0	775.0	1.01	0.019	0.023	0.0
Schirrmeister2017	TSBLR	5448.0	1975.0	1.01	0.001	0.001	0.0
Schirrmeister2017	TSGP	443.0	1093.0	1.01	0.013	0.02	0.0
Shin2017A	BDCNN	5531.0	1692.0	1.01	0.001	0.002	0.0
Shin2017A	BSCNN	5134.0	1743.0	1.01	0.001	0.001	0.0
Shin2017A	CSPBLDA	1961.0	2551.0	1.01	0.002	0.001	0.0
Shin2017A	CSPGP	696.0	1160.0	1.01	0.017	0.018	0.0
Shin2017A	TSBLR	3558.0	2442.0	1.01	0.002	0.002	0.0
Shin2017A	TSGP	833.0	1594.0	1.01	0.015	0.021	0.0

Table F.9: The summary of posterior sampling.

Summary Diagnostics IX							
Dataset	Pipeline	Min. ESS bulk	Min. ESS tail	Max. $\hat{R}$	Max. MCSE mean	Max. MCSE std.	Total divergences
Stieger2021	BDCNN	2911.0	1846.0	1.01	0.0	0.0	0.0
Stieger2021	BSCNN	4631.0	2007.0	1.01	0.0	0.001	0.0
Stieger2021	CSPBLDA	1609.0	1879.0	1.0	0.0	0.0	0.0
Stieger2021	CSPGP	653.0	1215.0	1.01	0.015	0.015	0.0
Stieger2021	TSBLR	3857.0	2429.0	1.0	0.0	0.001	0.0
Stieger2021	TSGP	352.0	660.0	1.01	0.018	0.02	0.0
Weibo2014	BDCNN	5859.0	1986.0	1.01	0.001	0.002	0.0
Weibo2014	BSCNN	6055.0	1925.0	1.01	0.001	0.001	0.0
Weibo2014	CSPBLDA	1224.0	1689.0	1.0	0.002	0.001	0.0
Weibo2014	CSPGP	637.0	1156.0	1.01	0.016	0.017	0.0
Weibo2014	TSBLR	3940.0	2237.0	1.01	0.001	0.001	0.0
Weibo2014	TSGP	1170.0	1678.0	1.01	0.012	0.022	0.0

Table F.10: The summary of posterior sampling.

Summary Diagnostics X							
Dataset	Pipeline	Min. ESS bulk	Min. ESS tail	Max. $\hat{R}$	Max. MCSE mean	Max. MCSE std.	Total divergences
Yang2025	BDCNN	2856.0	2352.0	1.01	0.001	0.001	0.0
Yang2025	BSCNN	4125.0	1743.0	1.01	0.0	0.001	0.0
Yang2025	CSPBLDA	1721.0	2297.0	1.0	0.0	0.0	0.0
Yang2025	CSPGP	738.0	1256.0	1.0	0.014	0.02	0.0
Yang2025	TSBLR	3754.0	2312.0	1.01	0.001	0.001	0.0
Yang2025	TSGP	836.0	1628.0	1.01	0.012	0.021	0.0
Zhou2020	BDCNN	5678.0	1850.0	1.01	0.001	0.001	0.0
Zhou2020	BSCNN	5265.0	1839.0	1.01	0.0	0.001	0.0
Zhou2020	CSPBLDA	1462.0	1985.0	1.01	0.001	0.001	0.0
Zhou2020	CSPGP	230.0	450.0	1.01	0.059	0.019	0.0
Zhou2020	TSBLR	6400.0	2151.0	1.01	0.001	0.001	0.0
Zhou2020	TSGP	1028.0	1874.0	1.0	0.012	0.021	0.0

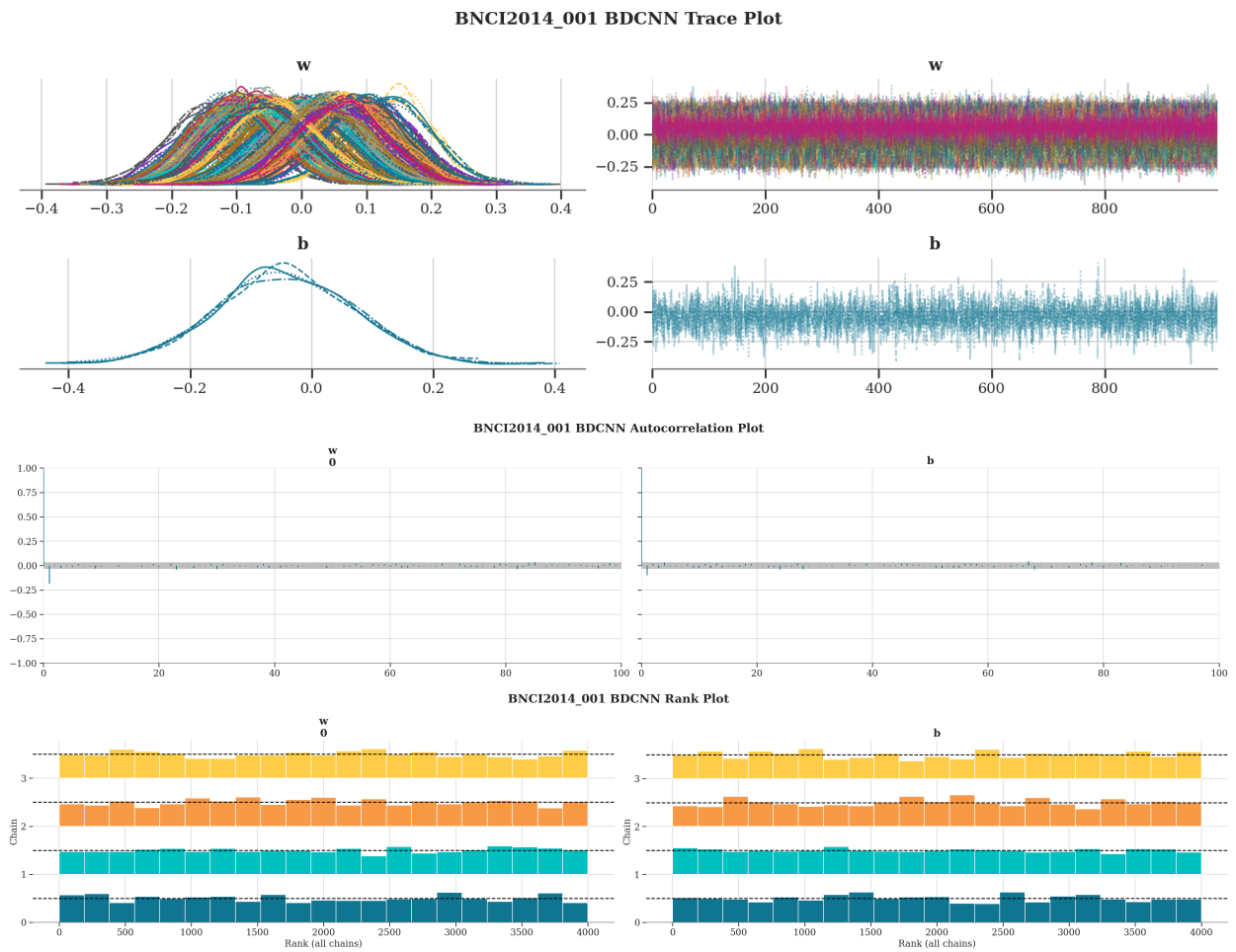


Figure F.1: The BNCI2014.001 BDCNN convergence plots.

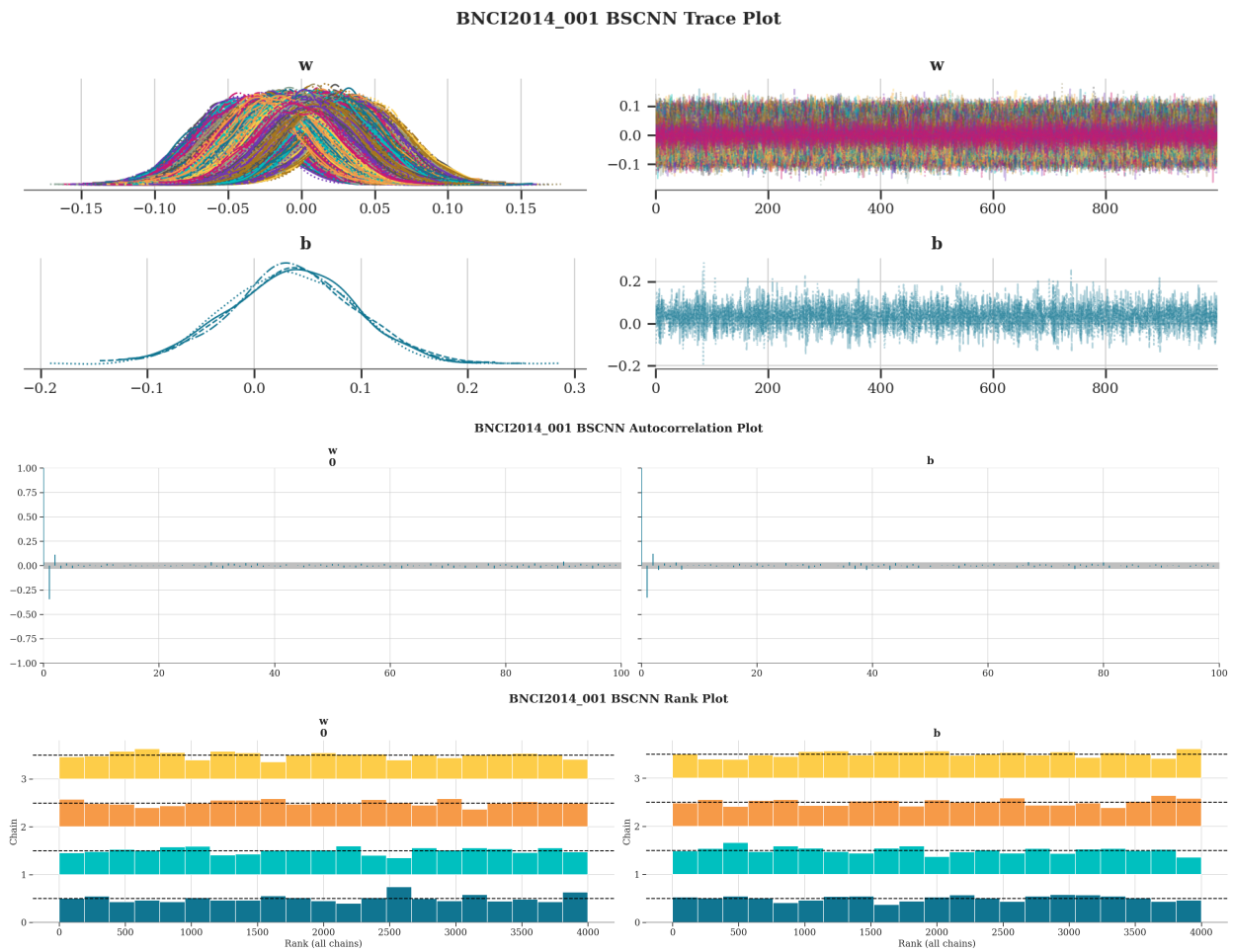


Figure F.2: The BNCI2014_001 BSCNN convergence plots.

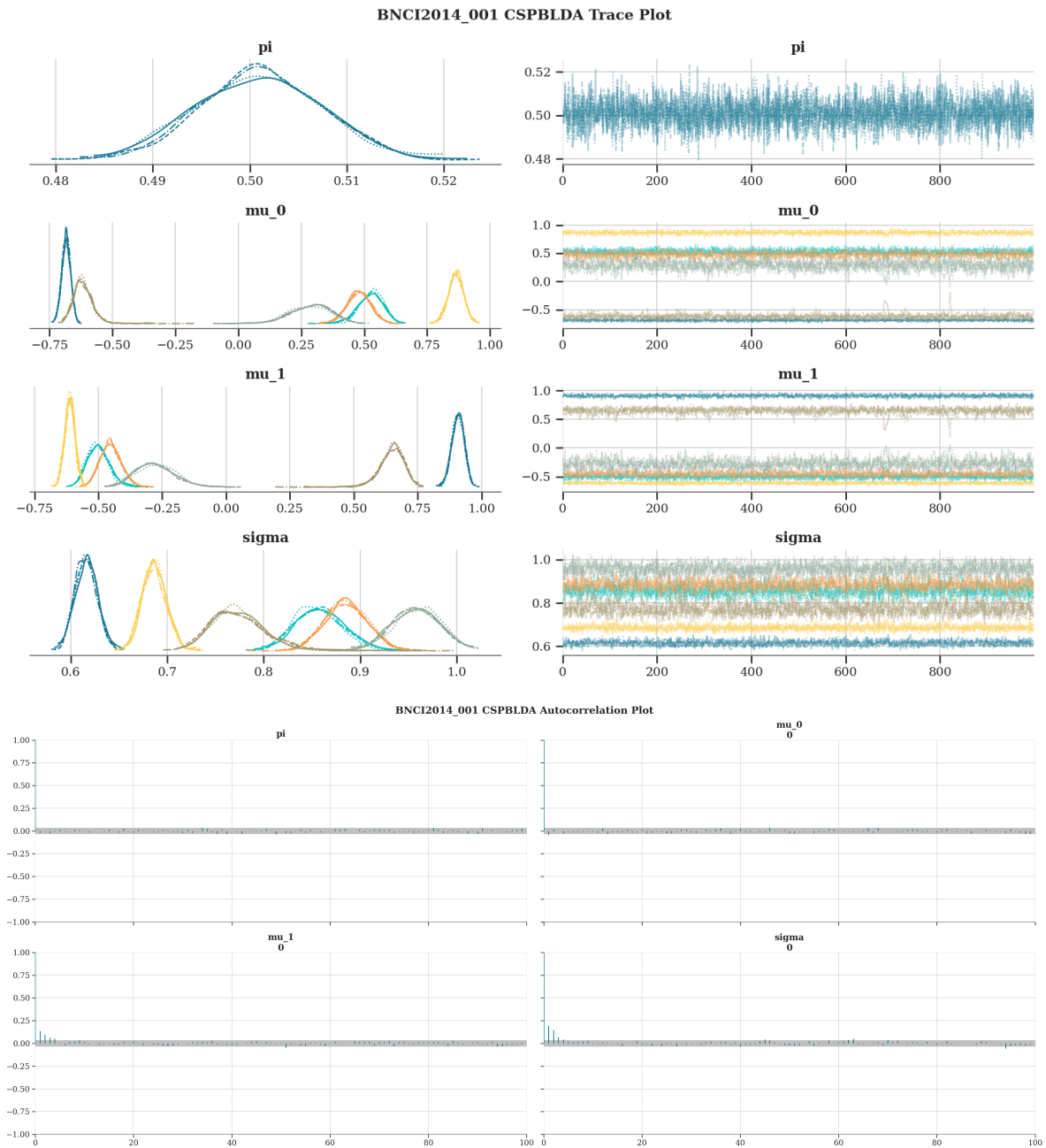


Figure F.3: The BNCI2014.001 CSPBLDA convergence plots.

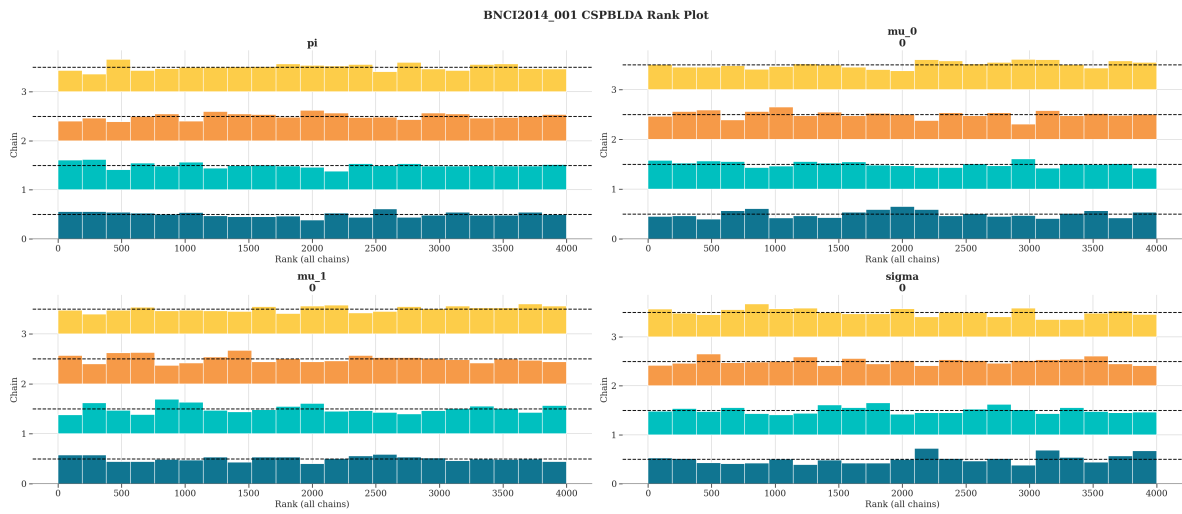


Figure F.4: The BNCI2014_001 CSPBLDA convergence plots continued.

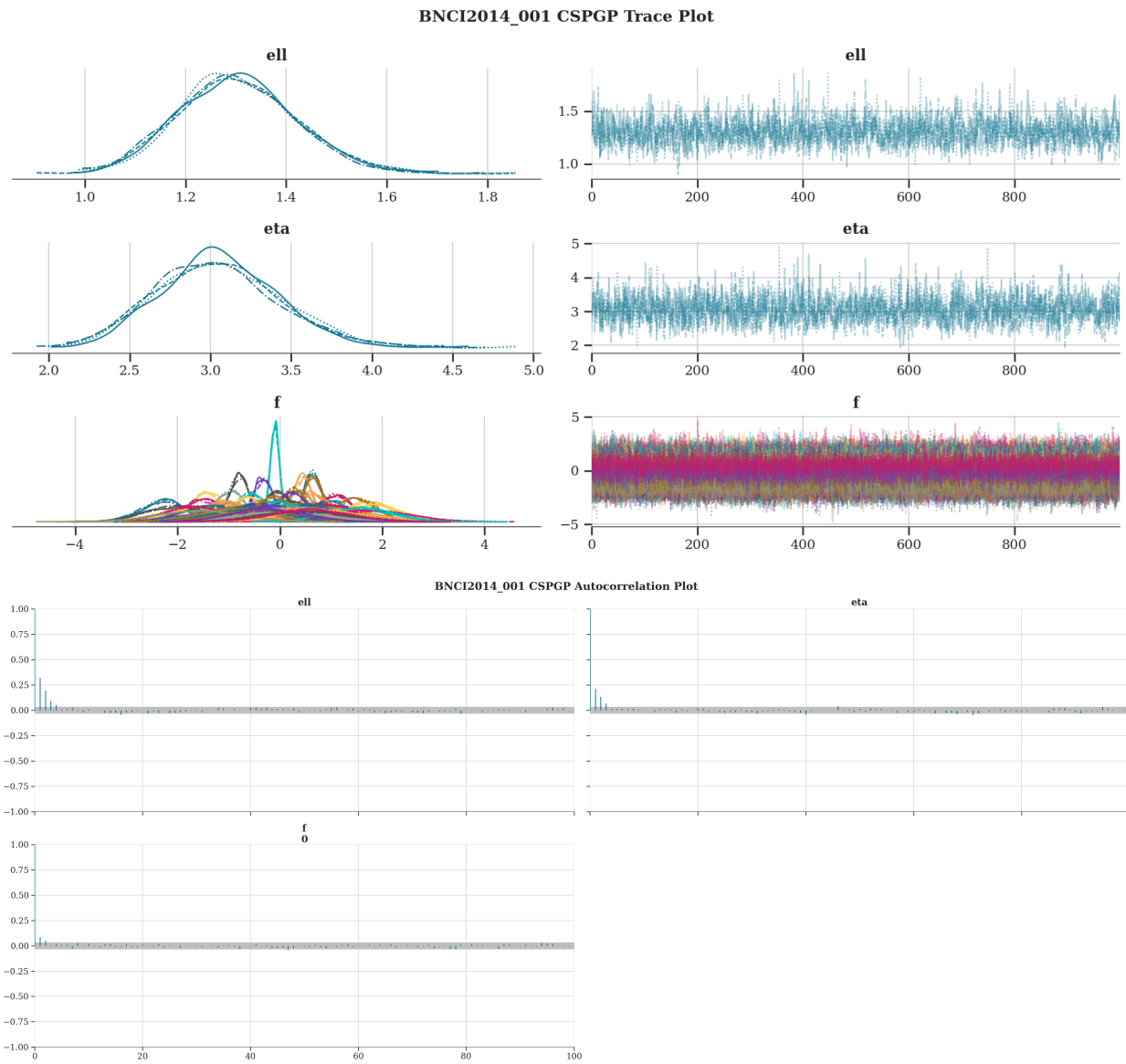


Figure F.5: The BNCI2014_001 CSPGP convergence plots.

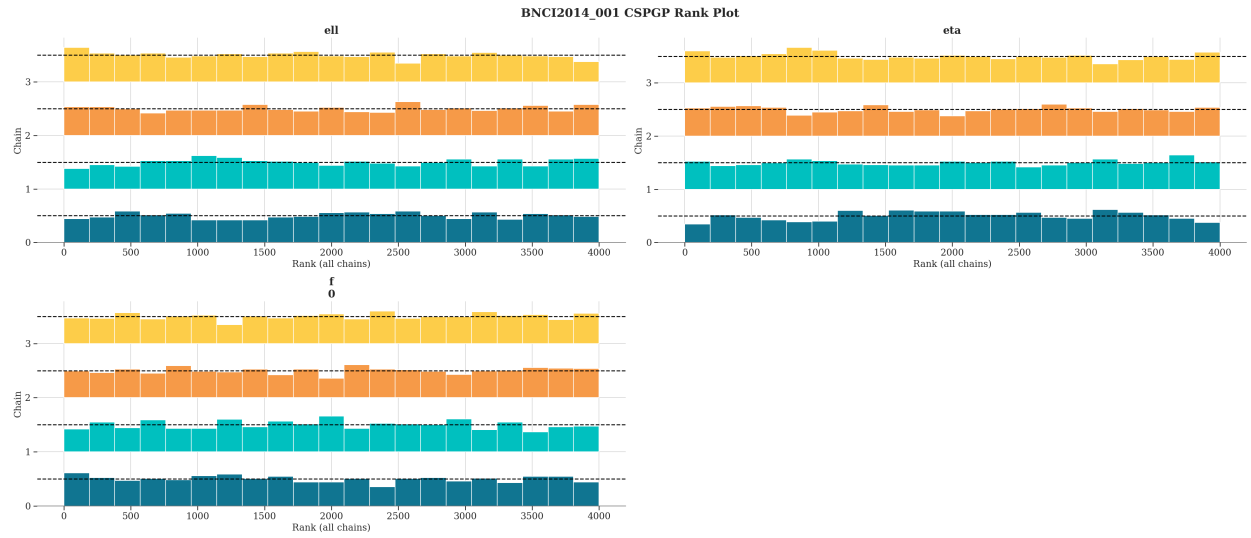


Figure F.6: The BNCI2014.001 CSPGP convergence plots continued.

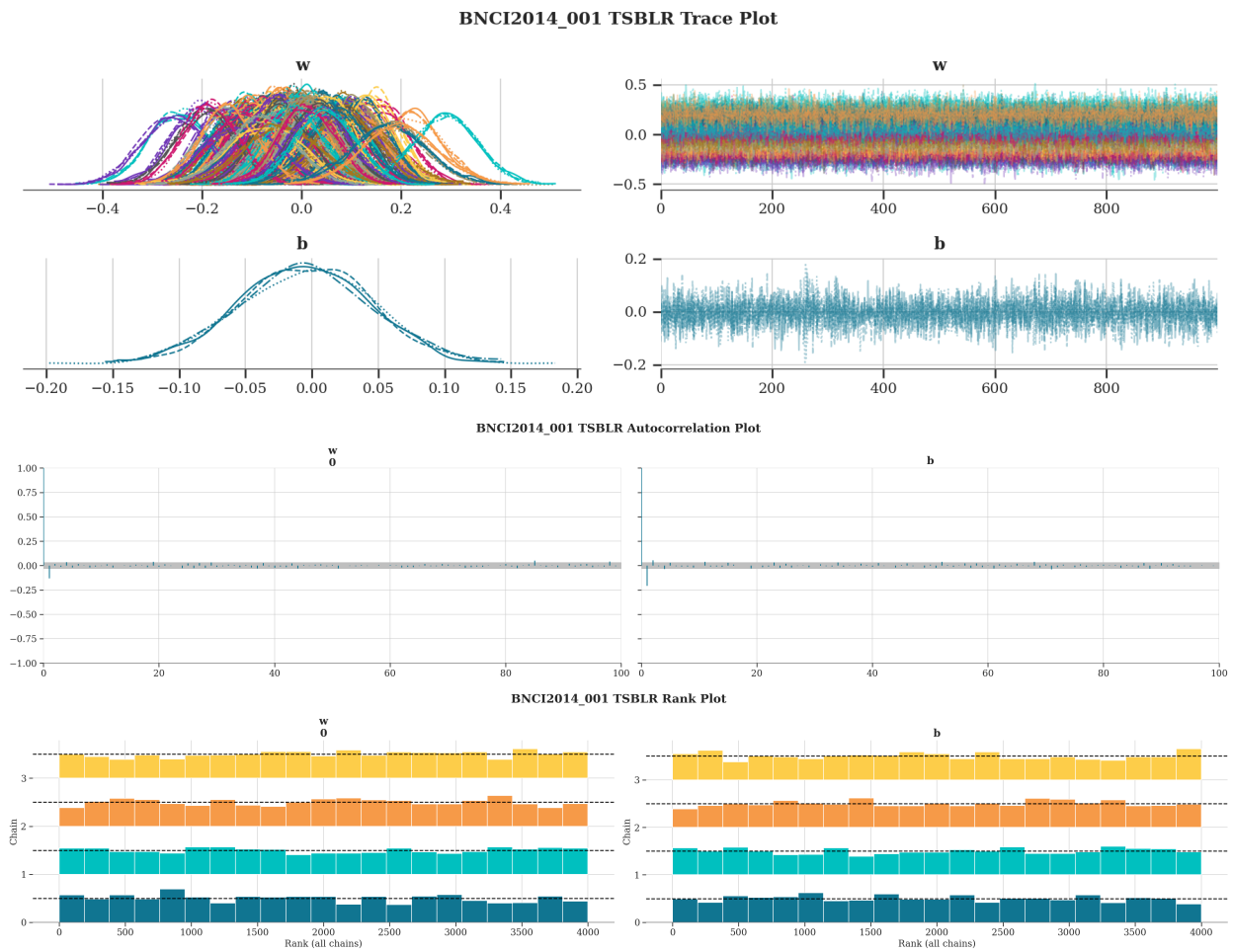


Figure F.7: The BNCI2014_001 TSBLR convergence plots.

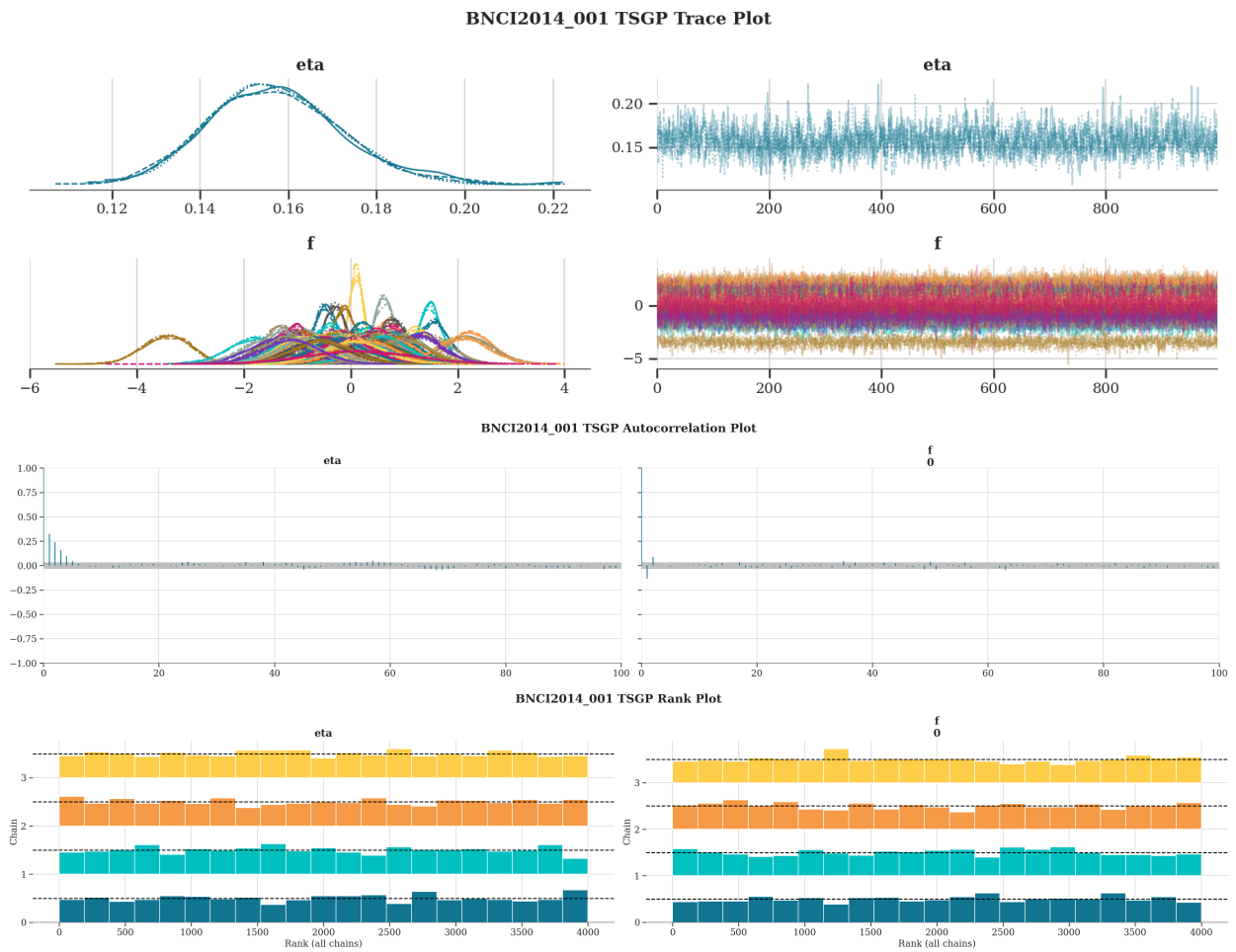


Figure F.8: The BNCI2014.001 TSGP convergence plots.

F.2 Raw Scores

Table F.11: The average raw metrics across k -fold cross-validation.

Average Metrics: BNCI2014_001							
Dataset	Pipeline	NLL	BS	AUROC	MCC	ECE	MCE
BNCI2014_001	CSPLDA	0.655	0.229	0.746	0.295	0.186	0.458
BNCI2014_001	CSPBLDA	0.719	0.25	0.732	0.283	0.222	0.486
BNCI2014_001	CSPSVM	0.748	0.261	0.737	0.239	0.239	0.554
BNCI2014_001	CSPGP	0.75	0.266	0.623	0.145	0.188	0.554
BNCI2014_001	TSLR	0.843	0.25	0.756	0.354	0.22	0.522
BNCI2014_001	TSBLR	0.679	0.227	0.766	0.351	0.19	0.419
BNCI2014_001	TSSVM	0.722	0.237	0.756	0.359	0.194	0.449
BNCI2014_001	TSGP	0.728	0.234	0.758	0.355	0.2	0.432
BNCI2014_001	SCNN	0.669	0.215	0.792	0.405	0.193	0.467
BNCI2014_001	BSCNN	0.602	0.203	0.792	0.424	0.161	0.353
BNCI2014_001	DCNN	0.582	0.187	0.81	0.46	0.161	0.481
BNCI2014_001	BDCNN	0.736	0.183	0.812	0.475	0.152	0.394

Table F.12: The average raw metrics across k -fold cross-validation.

Average Metrics: BNCI2014_004							
Dataset	Pipeline	NLL	BS	AUROC	MCC	ECE	MCE
BNCI2014_004	CSPLDA	0.616	0.212	0.793	0.376	0.178	0.426
BNCI2014_004	CSPBLDA	0.621	0.213	0.793	0.375	0.182	0.446
BNCI2014_004	CSPSVM	0.641	0.22	0.735	0.364	0.176	0.429
BNCI2014_004	CSPGP	0.66	0.225	0.741	0.336	0.172	0.428
BNCI2014_004	TSLR	0.631	0.217	0.786	0.347	0.187	0.452
BNCI2014_004	TSBLR	0.629	0.217	0.785	0.351	0.185	0.436
BNCI2014_004	TSSVM	0.638	0.22	0.783	0.343	0.186	0.436
BNCI2014_004	TSGP	0.631	0.217	0.786	0.349	0.186	0.444
BNCI2014_004	SCNN	0.615	0.197	0.811	0.454	0.159	0.455
BNCI2014_004	BSCNN	0.595	0.193	0.815	0.448	0.155	0.433
BNCI2014_004	DCNN	0.61	0.196	0.813	0.452	0.16	0.416
BNCI2014_004	BDCNN	0.745	0.198	0.815	0.451	0.163	0.418

Table F.13: The average raw metrics across k -fold cross-validation.

Average Metrics: Brandl2020							
Dataset	Pipeline	NLL	BS	AUROC	MCC	ECE	MCE
Brandl2020	CSPLDA	0.669	0.234	0.691	0.269	0.122	0.328
Brandl2020	CSPBLDA	0.694	0.238	0.691	0.282	0.13	0.314
Brandl2020	CSPSVM	0.683	0.241	0.663	0.234	0.111	0.217
Brandl2020	CSPGP	0.685	0.241	0.657	0.229	0.104	0.282
Brandl2020	TSLR	0.876	0.267	0.682	0.266	0.19	0.305
Brandl2020	TSBLR	0.676	0.236	0.7	0.277	0.138	0.236
Brandl2020	TSSVM	0.779	0.254	0.682	0.271	0.163	0.285
Brandl2020	TSGP	0.687	0.238	0.697	0.28	0.131	0.255
Brandl2020	SCNN	0.667	0.222	0.747	0.347	0.131	0.236
Brandl2020	BSCNN	0.649	0.218	0.75	0.347	0.112	0.191
Brandl2020	DCNN	0.696	0.232	0.741	0.316	0.153	0.28
Brandl2020	BDCNN	0.737	0.237	0.737	0.318	0.159	0.266

Table F.14: The average raw metrics across k -fold cross-validation.

Average Metrics: Chang2025							
Dataset	Pipeline	NLL	BS	AUROC	MCC	ECE	MCE
Chang2025	CSPLDA	0.653	0.228	0.693	0.287	0.115	0.356
Chang2025	CSPBLDA	0.651	0.227	0.692	0.298	0.114	0.344
Chang2025	CSPSVM	0.667	0.229	0.698	0.274	0.131	0.294
Chang2025	CSPGP	0.679	0.234	0.683	0.281	0.134	0.333
Chang2025	TSLR	0.96	0.258	0.698	0.291	0.215	0.43
Chang2025	TSBLR	0.629	0.217	0.723	0.324	0.125	0.322
Chang2025	TSSVM	0.735	0.234	0.691	0.282	0.146	0.317
Chang2025	TSGP	0.639	0.221	0.714	0.303	0.135	0.334
Chang2025	SCNN	0.719	0.227	0.739	0.365	0.18	0.396
Chang2025	BSCNN	0.627	0.203	0.757	0.391	0.121	0.288
Chang2025	DCNN	0.674	0.218	0.762	0.407	0.177	0.364
Chang2025	BDCNN	0.706	0.213	0.771	0.394	0.169	0.406

Table F.15: The average raw metrics across k -fold cross-validation.

Average Metrics: Cho2017							
Dataset	Pipeline	NLL	BS	AUROC	MCC	ECE	MCE
Cho2017	CSPLDA	0.676	0.24	0.637	0.197	0.076	0.481
Cho2017	CSPBLDA	0.676	0.24	0.636	0.198	0.075	0.495
Cho2017	CSPSVM	0.681	0.243	0.625	0.18	0.082	0.289
Cho2017	CSPGP	0.689	0.245	0.621	0.169	0.075	0.429
Cho2017	TSLR	0.679	0.239	0.666	0.236	0.102	0.266
Cho2017	TSBLR	0.653	0.23	0.672	0.237	0.072	0.219
Cho2017	TSSVM	0.661	0.233	0.664	0.233	0.075	0.24
Cho2017	TSGP	0.658	0.233	0.658	0.208	0.065	0.215
Cho2017	SCNN	0.622	0.212	0.738	0.348	0.079	0.213
Cho2017	BSCNN	0.63	0.212	0.744	0.361	0.089	0.21
Cho2017	DCNN	0.626	0.214	0.727	0.329	0.069	0.162
Cho2017	BDCNN	0.622	0.212	0.733	0.343	0.068	0.162

Table F.16: The average raw metrics across k -fold cross-validation.

Average Metrics: Dreyer2023							
Dataset	Pipeline	NLL	BS	AUROC	MCC	ECE	MCE
Dreyer2023	CSPLDA	0.688	0.247	0.553	0.067	0.024	0.344
Dreyer2023	CSPBLDA	0.688	0.247	0.551	0.068	0.023	0.395
Dreyer2023	CSPSVM	0.684	0.246	0.566	0.09	0.029	0.218
Dreyer2023	CSPGP	0.687	0.246	0.558	0.076	0.034	0.195
Dreyer2023	TSLR	0.637	0.218	0.726	0.326	0.083	0.191
Dreyer2023	TSBLR	0.616	0.213	0.729	0.328	0.062	0.156
Dreyer2023	TSSVM	0.628	0.217	0.725	0.321	0.07	0.179
Dreyer2023	TSGP	0.629	0.219	0.713	0.305	0.062	0.152
Dreyer2023	SCNN	0.567	0.175	0.823	0.499	0.07	0.138
Dreyer2023	BSCNN	0.614	0.175	0.822	0.506	0.064	0.142
Dreyer2023	DCNN	0.497	0.161	0.85	0.54	0.057	0.113
Dreyer2023	BDCNN	0.536	0.165	0.848	0.532	0.079	0.176

Table F.17: The average raw metrics across k -fold cross-validation.

Average Metrics: Forenzo2023							
Dataset	Pipeline	NLL	BS	AUROC	MCC	ECE	MCE
Forenzo2023	CSPLDA	0.692	0.249	0.555	0.101	0.049	0.381
Forenzo2023	CSPBLDA	0.692	0.249	0.551	0.064	0.037	0.074
Forenzo2023	CSPSVM	0.693	0.25	0.55	0.076	0.036	0.178
Forenzo2023	CSPGP	0.694	0.25	0.548	0.08	0.046	0.205
Forenzo2023	TSLR	0.798	0.273	0.614	0.172	0.163	0.392
Forenzo2023	TSBLR	0.685	0.243	0.634	0.201	0.09	0.257
Forenzo2023	TSSVM	0.705	0.252	0.613	0.178	0.101	0.41
Forenzo2023	TSGP	0.684	0.243	0.633	0.197	0.092	0.272
Forenzo2023	SCNN	0.688	0.236	0.67	0.248	0.091	0.279
Forenzo2023	BSCNN	0.663	0.229	0.681	0.272	0.076	0.29
Forenzo2023	DCNN	0.776	0.234	0.69	0.271	0.107	0.244
Forenzo2023	BDCNN	0.808	0.245	0.689	0.26	0.14	0.319

Table F.18: The average raw metrics across k -fold cross-validation.

Average Metrics: GrosseWentrup2009							
Dataset	Pipeline	NLL	BS	AUROC	MCC	ECE	MCE
GrosseWentrup2009	CSPLDA	0.667	0.226	0.711	0.303	0.108	0.254
GrosseWentrup2009	CSPBLDA	0.661	0.225	0.713	0.314	0.106	0.284
GrosseWentrup2009	CSPSVM	0.692	0.241	0.688	0.237	0.138	0.287
GrosseWentrup2009	CSPGP	0.733	0.253	0.669	0.224	0.146	0.346
GrosseWentrup2009	TSLR	1.371	0.305	0.705	0.295	0.282	0.431
GrosseWentrup2009	TSBLR	0.635	0.223	0.709	0.266	0.116	0.443
GrosseWentrup2009	TSSVM	0.661	0.231	0.702	0.284	0.129	0.269
GrosseWentrup2009	TSGP	0.705	0.24	0.689	0.282	0.129	0.347
GrosseWentrup2009	SCNN	0.922	0.262	0.721	0.308	0.211	0.372
GrosseWentrup2009	BSCNN	0.634	0.217	0.746	0.353	0.13	0.24
GrosseWentrup2009	DCNN	0.834	0.242	0.76	0.377	0.199	0.383
GrosseWentrup2009	BDCNN	0.775	0.225	0.764	0.387	0.168	0.302

Table F.19: The average raw metrics across k -fold cross-validation.

Average Metrics: GuttmanFlury2025_MI							
Dataset	Pipeline	NLL	BS	AUROC	MCC	ECE	MCE
GuttmanFlury2025_MI	CSPLDA	0.692	0.249	0.541	0.066	0.038	0.179
GuttmanFlury2025_MI	CSPBLDA	0.692	0.249	0.541	0.052	0.038	0.166
GuttmanFlury2025_MI	CSPSVM	0.693	0.25	0.508	0.05	0.019	0.026
GuttmanFlury2025_MI	CSPGP	0.69	0.249	0.543	0.019	0.049	0.102
GuttmanFlury2025_MI	TSLR	0.988	0.322	0.559	0.066	0.262	0.514
GuttmanFlury2025_MI	TSBLR	0.695	0.25	0.571	0.093	0.081	0.362
GuttmanFlury2025_MI	TSSVM	0.701	0.253	0.553	0.051	0.075	0.282
GuttmanFlury2025_MI	TSGP	0.693	0.25	0.495	-0.012	0.034	0.037
GuttmanFlury2025_MI	SCNN	0.693	0.243	0.656	0.225	0.125	0.409
GuttmanFlury2025_MI	BSCNN	0.644	0.227	0.677	0.224	0.08	0.297
GuttmanFlury2025_MI	DCNN	0.67	0.233	0.668	0.247	0.099	0.295
GuttmanFlury2025_MI	BDCNN	0.714	0.242	0.652	0.21	0.119	0.271

Table F.20: The average raw metrics across k -fold cross-validation.

Average Metrics: HefmiIch2025							
Dataset	Pipeline	NLL	BS	AUROC	MCC	ECE	MCE
HefmiIch2025	CSPLDA	0.692	0.25	0.541	0.031	0.034	0.175
HefmiIch2025	CSPBLDA	0.692	0.25	0.538	0.034	0.038	0.1
HefmiIch2025	CSPSVM	0.696	0.252	0.471	0.065	0.052	0.131
HefmiIch2025	CSPGP	0.693	0.25	0.51	-0.018	0.048	0.06
HefmiIch2025	TSLR	0.708	0.253	0.597	0.113	0.128	0.452
HefmiIch2025	TSBLR	0.688	0.247	0.591	0.164	0.08	0.346
HefmiIch2025	TSSVM	0.684	0.246	0.594	0.113	0.066	0.367
HefmiIch2025	TSGP	0.692	0.25	0.524	0.042	0.051	0.099
HefmiIch2025	SCNN	0.713	0.26	0.5	-0.022	0.11	0.485
HefmiIch2025	BSCNN	0.698	0.252	0.496	-0.016	0.062	0.322
HefmiIch2025	DCNN	0.771	0.27	0.488	-0.034	0.067	0.204
HefmiIch2025	BDCNN	0.703	0.255	0.495	-0.005	0.065	0.155

Table F.21: The average raw metrics across k -fold cross-validation.

Average Metrics: Kumar2024							
Dataset	Pipeline	NLL	BS	AUROC	MCC	ECE	MCE
Kumar2024	CSPLDA	0.686	0.245	0.622	0.142	0.097	0.525
Kumar2024	CSPBLDA	0.683	0.245	0.627	0.163	0.073	0.492
Kumar2024	CSPSVM	0.692	0.248	0.62	0.146	0.124	0.373
Kumar2024	CSPGP	0.703	0.253	0.601	0.133	0.127	0.389
Kumar2024	TSLR	0.692	0.239	0.683	0.252	0.132	0.441
Kumar2024	TSBLR	0.668	0.233	0.685	0.26	0.119	0.473
Kumar2024	TSSVM	0.681	0.237	0.681	0.248	0.128	0.482
Kumar2024	TSGP	0.677	0.235	0.684	0.256	0.123	0.415
Kumar2024	SCNN	0.634	0.221	0.729	0.314	0.125	0.338
Kumar2024	BSCNN	0.641	0.217	0.734	0.342	0.115	0.357
Kumar2024	DCNN	0.61	0.21	0.742	0.37	0.108	0.328
Kumar2024	BDCNN	0.704	0.223	0.739	0.345	0.141	0.332

Table F.22: The average raw metrics across k -fold cross-validation.

Average Metrics: Lee2019_MI							
Dataset	Pipeline	NLL	BS	AUROC	MCC	ECE	MCE
Lee2019_MI	CSPLDA	0.629	0.217	0.725	0.321	0.093	0.332
Lee2019_MI	CSPBLDA	0.631	0.218	0.724	0.322	0.094	0.32
Lee2019_MI	CSPSVM	0.637	0.223	0.712	0.292	0.108	0.253
Lee2019_MI	CSPGP	0.635	0.222	0.709	0.292	0.097	0.271
Lee2019_MI	TSLR	0.631	0.208	0.768	0.377	0.119	0.245
Lee2019_MI	TSBLR	0.598	0.202	0.767	0.381	0.096	0.22
Lee2019_MI	TSSVM	0.635	0.205	0.767	0.377	0.103	0.227
Lee2019_MI	TSGP	0.624	0.21	0.747	0.358	0.093	0.2
Lee2019_MI	SCNN	0.54	0.179	0.819	0.48	0.083	0.198
Lee2019_MI	BSCNN	0.552	0.18	0.817	0.456	0.084	0.208
Lee2019_MI	DCNN	0.525	0.174	0.826	0.485	0.084	0.205
Lee2019_MI	BDCNN	0.585	0.18	0.826	0.488	0.101	0.225

Table F.23: The average raw metrics across k -fold cross-validation.

Average Metrics: Liu2024							
Dataset	Pipeline	NLL	BS	AUROC	MCC	ECE	MCE
Liu2024	CSPLDA	0.694	0.25	0.532	0.038	0.036	0.515
Liu2024	CSPBLDA	0.693	0.25	0.538	0.043	0.033	0.368
Liu2024	CSPSVM	0.692	0.25	0.54	0.058	0.02	0.031
Liu2024	CSPGP	0.693	0.25	0.519	0.014	0.032	0.099
Liu2024	TSLR	0.702	0.254	0.542	0.066	0.071	0.363
Liu2024	TSBLR	0.697	0.252	0.541	0.059	0.058	0.343
Liu2024	TSSVM	0.693	0.25	0.53	0.057	0.03	0.087
Liu2024	TSGP	0.693	0.25	0.515	0.045	0.024	0.027
Liu2024	SCNN	0.714	0.258	0.554	0.083	0.111	0.486
Liu2024	BSCNN	0.688	0.247	0.573	0.108	0.048	0.283
Liu2024	DCNN	0.753	0.254	0.559	0.094	0.068	0.613
Liu2024	BDCNN	0.812	0.272	0.529	0.027	0.105	0.361

Table F.24: The average raw metrics across k -fold cross-validation.

Average Metrics: PhysionetMI							
Dataset	Pipeline	NLL	BS	AUROC	MCC	ECE	MCE
PhysionetMI	CSPLDA	0.677	0.234	0.673	0.233	0.084	0.323
PhysionetMI	CSPBLDA	0.678	0.234	0.674	0.24	0.085	0.333
PhysionetMI	CSPSVM	0.655	0.231	0.67	0.247	0.073	0.222
PhysionetMI	CSPGP	0.652	0.229	0.673	0.237	0.077	0.315
PhysionetMI	TSLR	0.685	0.236	0.689	0.261	0.112	0.237
PhysionetMI	TSBLR	0.642	0.225	0.685	0.263	0.073	0.188
PhysionetMI	TSSVM	0.643	0.225	0.689	0.267	0.076	0.199
PhysionetMI	TSGP	0.659	0.233	0.663	0.23	0.08	0.26
PhysionetMI	SCNN	0.655	0.212	0.738	0.347	0.085	0.182
PhysionetMI	BSCNN	0.693	0.207	0.743	0.361	0.07	0.165
PhysionetMI	DCNN	0.674	0.228	0.688	0.252	0.078	0.208
PhysionetMI	BDCNN	0.745	0.215	0.721	0.317	0.062	0.176

Table F.25: The average raw metrics across k -fold cross-validation.

Average Metrics Schirrmeister2017							
Dataset	Pipeline	NLL	BS	AUROC	MCC	ECE	MCE
Schirrmeister2017	CSPLDA	0.773	0.272	0.625	0.131	0.171	0.369
Schirrmeister2017	CSPBLDA	0.75	0.267	0.627	0.127	0.162	0.395
Schirrmeister2017	CSPSVM	0.845	0.294	0.614	0.124	0.228	0.489
Schirrmeister2017	CSPGP	0.854	0.287	0.601	0.137	0.199	0.454
Schirrmeister2017	TSLR	1.538	0.317	0.69	0.239	0.285	0.498
Schirrmeister2017	TSBLR	0.708	0.233	0.731	0.318	0.164	0.298
Schirrmeister2017	TSSVM	1.128	0.293	0.667	0.195	0.239	0.553
Schirrmeister2017	TSGP	0.833	0.255	0.719	0.307	0.196	0.311
Schirrmeister2017	SCNN	0.766	0.228	0.792	0.404	0.184	0.327
Schirrmeister2017	BSCNN	0.77	0.219	0.791	0.425	0.164	0.27
Schirrmeister2017	DCNN	0.687	0.206	0.802	0.445	0.151	0.319
Schirrmeister2017	BDCNN	0.971	0.206	0.81	0.462	0.166	0.359

Table F.26: The average raw metrics across k -fold cross-validation.

Average Metrics Shin2017A							
Dataset	Pipeline	NLL	BS	AUROC	MCC	ECE	MCE
Shin2017A	CSPLDA	0.666	0.231	0.707	0.322	0.131	0.337
Shin2017A	CSPBLDA	0.673	0.231	0.707	0.322	0.141	0.377
Shin2017A	CSPSVM	0.674	0.236	0.683	0.271	0.137	0.364
Shin2017A	CSPGP	0.695	0.242	0.658	0.251	0.141	0.361
Shin2017A	TSLR	0.917	0.283	0.663	0.254	0.229	0.445
Shin2017A	TSBLR	0.681	0.24	0.677	0.28	0.139	0.336
Shin2017A	TSSVM	0.822	0.271	0.664	0.243	0.208	0.465
Shin2017A	TSGP	0.813	0.268	0.67	0.268	0.209	0.378
Shin2017A	SCNN	0.774	0.244	0.715	0.301	0.181	0.371
Shin2017A	BSCNN	0.625	0.214	0.734	0.336	0.11	0.251
Shin2017A	DCNN	0.743	0.23	0.736	0.34	0.171	0.367
Shin2017A	BDCNN	0.657	0.218	0.741	0.37	0.139	0.331

Table F.27: The average raw metrics across k -fold cross-validation.

Average Metrics: Stieger2021							
Dataset	Pipeline	NLL	BS	AUROC	MCC	ECE	MCE
Stieger2021	CSPLDA	0.65	0.228	0.679	0.246	0.072	0.418
Stieger2021	CSPBLDA	0.65	0.228	0.679	0.246	0.071	0.397
Stieger2021	CSPSVM	0.651	0.23	0.674	0.237	0.074	0.239
Stieger2021	CSPGP	0.649	0.229	0.676	0.238	0.073	0.336
Stieger2021	TSLR	0.626	0.216	0.729	0.326	0.083	0.21
Stieger2021	TSBLR	0.62	0.215	0.729	0.326	0.079	0.208
Stieger2021	TSSVM	0.624	0.216	0.728	0.323	0.08	0.221
Stieger2021	TSGP	0.645	0.227	0.685	0.26	0.066	0.243
Stieger2021	SCNN	0.564	0.188	0.796	0.441	0.073	0.165
Stieger2021	BSCNN	0.59	0.191	0.799	0.444	0.091	0.189
Stieger2021	DCNN	0.543	0.181	0.81	0.461	0.07	0.156
Stieger2021	BDCNN	0.567	0.188	0.8	0.441	0.08	0.179

Table F.28: The average raw metrics across k -fold cross-validation.

Average Metrics: Weibo2014							
Dataset	Pipeline	NLL	BS	AUROC	MCC	ECE	MCE
Weibo2014	CSPLDA	0.577	0.199	0.746	0.368	0.096	0.288
Weibo2014	CSPBLDA	0.6	0.203	0.746	0.381	0.109	0.212
Weibo2014	CSPSVM	0.623	0.217	0.707	0.312	0.125	0.254
Weibo2014	CSPGP	0.646	0.22	0.701	0.301	0.105	0.283
Weibo2014	TSLR	1.102	0.306	0.666	0.208	0.275	0.482
Weibo2014	TSBLR	0.636	0.225	0.682	0.259	0.117	0.325
Weibo2014	TSSVM	0.747	0.262	0.642	0.18	0.175	0.464
Weibo2014	TSGP	0.716	0.256	0.67	0.174	0.189	0.504
Weibo2014	SCNN	0.873	0.263	0.716	0.324	0.213	0.37
Weibo2014	BSCNN	0.674	0.233	0.718	0.324	0.16	0.397
Weibo2014	DCNN	0.949	0.257	0.727	0.327	0.216	0.436
Weibo2014	BDCNN	0.923	0.249	0.73	0.337	0.202	0.387

Table F.29: The average raw metrics across k -fold cross-validation.

Average Metrics: Yang2025							
Dataset	Pipeline	NLL	BS	AUROC	MCC	ECE	MCE
Yang2025	CSPLDA	0.685	0.246	0.593	0.127	0.055	0.449
Yang2025	CSPBLDA	0.685	0.246	0.592	0.127	0.056	0.446
Yang2025	CSPSVM	0.69	0.248	0.583	0.112	0.077	0.277
Yang2025	CSPGP	0.686	0.246	0.588	0.11	0.065	0.329
Yang2025	TSLR	0.711	0.254	0.599	0.14	0.106	0.409
Yang2025	TSBLR	0.7	0.251	0.599	0.138	0.095	0.379
Yang2025	TSSVM	0.699	0.251	0.598	0.138	0.093	0.396
Yang2025	TSGP	0.699	0.251	0.584	0.113	0.081	0.383
Yang2025	SCNN	0.654	0.23	0.679	0.256	0.079	0.297
Yang2025	BSCNN	0.645	0.225	0.692	0.28	0.073	0.225
Yang2025	DCNN	0.575	0.194	0.778	0.404	0.074	0.163
Yang2025	BDCNN	0.656	0.203	0.779	0.407	0.11	0.206

Table F.30: The average raw metrics across k -fold cross-validation.

Average Metrics: Zhou2020							
Dataset	Pipeline	NLL	BS	AUROC	MCC	ECE	MCE
Zhou2020	CSPLDA	0.687	0.246	0.662	0.211	0.143	0.425
Zhou2020	CSPBLDA	0.687	0.246	0.661	0.205	0.145	0.449
Zhou2020	CSPSVM	0.7	0.252	0.624	0.155	0.142	0.429
Zhou2020	CSPGP	0.705	0.253	0.608	0.119	0.129	0.456
Zhou2020	TSLR	0.807	0.262	0.701	0.246	0.22	0.461
Zhou2020	TSBLR	0.674	0.236	0.697	0.271	0.169	0.406
Zhou2020	TSSVM	0.725	0.249	0.7	0.251	0.196	0.46
Zhou2020	TSGP	0.706	0.245	0.674	0.227	0.166	0.404
Zhou2020	SCNN	0.632	0.211	0.757	0.382	0.148	0.351
Zhou2020	BSCNN	0.588	0.2	0.768	0.389	0.142	0.361
Zhou2020	DCNN	0.59	0.194	0.789	0.451	0.145	0.364
Zhou2020	BDCNN	0.589	0.192	0.787	0.428	0.139	0.359

Table F.31: The metrics differences.

Metrics Difference I								
Dataset	Freq. pipeline	Bayes. pipeline	NLL	BS	AUROC	MCC	ECE	MCE
BNCI2014_001	CSPLDA	CSPBLDA	0.063	0.021	-0.014	-0.013	0.036	0.027
BNCI2014_001	CSPSVM	CSPGP	0.002	0.005	-0.114	-0.094	-0.051	0.001
BNCI2014_001	TSLR	TSBLR	-0.164	-0.022	0.01	-0.003	-0.03	-0.103
BNCI2014_001	TSSVM	TSGP	0.007	-0.003	0.002	-0.004	0.006	-0.016
BNCI2014_001	SCNN	BSCNN	-0.067	-0.011	0.0	0.019	-0.032	-0.113
BNCI2014_001	DCNN	BDCNN	0.154	-0.004	0.002	0.016	-0.009	-0.087
BNCI2014_004	CSPLDA	CSPBLDA	0.006	0.001	-0.001	-0.001	0.004	0.02
BNCI2014_004	CSPSVM	CSPGP	0.019	0.004	0.006	-0.028	-0.003	-0.0
BNCI2014_004	TSLR	TSBLR	-0.001	-0.0	-0.001	0.005	-0.003	-0.017
BNCI2014_004	TSSVM	TSGP	-0.007	-0.002	0.003	0.005	-0.0	0.008
BNCI2014_004	SCNN	BSCNN	-0.02	-0.003	0.005	-0.007	-0.005	-0.021
BNCI2014_004	DCNN	BDCNN	0.135	0.002	0.003	-0.001	0.003	0.001

Table F.32: The metrics differences.

Metrics Difference II								
Dataset	Freq. pipeline	Bayes. pipeline	NLL	BS	AUROC	MCC	ECE	MCE
Brandl2020	CSPLDA	CSPBLDA	0.025	0.005	0.0	0.013	0.008	-0.015
Brandl2020	CSPSVM	CSPGP	0.002	-0.0	-0.006	-0.005	-0.008	0.065
Brandl2020	TSLR	TSBLR	-0.201	-0.031	0.018	0.011	-0.052	-0.068
Brandl2020	TSSVM	TSGP	-0.092	-0.016	0.015	0.009	-0.033	-0.03
Brandl2020	SCNN	BSCNN	-0.018	-0.004	0.004	-0.001	-0.019	-0.045
Brandl2020	DCNN	BDCNN	0.041	0.005	-0.004	0.002	0.006	-0.014
Chang2025	CSPLDA	CSPBLDA	-0.002	-0.001	-0.001	0.01	-0.001	-0.012
Chang2025	CSPSVM	CSPGP	0.012	0.005	-0.015	0.007	0.003	0.039
Chang2025	TSLR	TSBLR	-0.332	-0.041	0.025	0.033	-0.09	-0.108
Chang2025	TSSVM	TSGP	-0.096	-0.012	0.023	0.021	-0.011	0.017
Chang2025	SCNN	BSCNN	-0.092	-0.024	0.017	0.026	-0.059	-0.108
Chang2025	DCNN	BDCNN	0.031	-0.004	0.009	-0.012	-0.008	0.042

Table F.33: The metrics differences.

Metrics Difference III								
Dataset	Freq. pipeline	Bayes. pipeline	NLL	BS	AUROC	MCC	ECE	MCE
Cho2017	CSPLDA	CSPBLDA	0.0	0.0	-0.001	0.001	-0.001	0.013
Cho2017	CSPSVM	CSPGP	0.008	0.002	-0.004	-0.011	-0.007	0.14
Cho2017	TSLR	TSBLR	-0.026	-0.008	0.005	0.001	-0.029	-0.047
Cho2017	TSSVM	TSGP	-0.002	-0.0	-0.006	-0.025	-0.009	-0.025
Cho2017	SCNN	BSCNN	0.008	0.0	0.006	0.012	0.011	-0.002
Cho2017	DCNN	BDCNN	-0.003	-0.002	0.006	0.014	-0.0	-0.0
Dreyer2023	CSPLDA	CSPBLDA	-0.0	0.0	-0.002	0.0	-0.001	0.051
Dreyer2023	CSPSVM	CSPGP	0.002	0.001	-0.008	-0.014	0.005	-0.023
Dreyer2023	TSLR	TSBLR	-0.021	-0.005	0.003	0.002	-0.021	-0.035
Dreyer2023	TSSVM	TSGP	0.001	0.002	-0.012	-0.016	-0.008	-0.027
Dreyer2023	SCNN	BSCNN	0.047	-0.0	-0.001	0.007	-0.006	0.003
Dreyer2023	DCNN	BDCNN	0.039	0.005	-0.002	-0.008	0.022	0.063

Table F.34: The metrics differences.

Metrics Differences IV								
Dataset	Freq. pipeline	Bayes. pipeline	NLL	BS	AUROC	MCC	ECE	MCE
Forenzo2023	CSPLDA	CSPBLDA	-0.0	-0.0	-0.005	-0.036	-0.012	-0.307
Forenzo2023	CSPSVM	CSPGP	0.002	0.001	-0.002	0.005	0.011	0.027
Forenzo2023	TSLR	TSBLR	-0.113	-0.03	0.021	0.029	-0.073	-0.134
Forenzo2023	TSSVM	TSGP	-0.021	-0.008	0.02	0.019	-0.009	-0.138
Forenzo2023	SCNN	BSCNN	-0.025	-0.006	0.011	0.023	-0.015	0.011
Forenzo2023	DCNN	BDCNN	0.032	0.011	-0.001	-0.011	0.033	0.075
GrosseWentrup2009	CSPLDA	CSPBLDA	-0.005	-0.001	0.002	0.011	-0.002	0.03
GrosseWentrup2009	CSPSVM	CSPGP	0.041	0.012	-0.019	-0.013	0.008	0.059
GrosseWentrup2009	TSLR	TSBLR	-0.736	-0.081	0.003	-0.029	-0.165	0.012
GrosseWentrup2009	TSSVM	TSGP	0.044	0.009	-0.013	-0.002	0.001	0.079
GrosseWentrup2009	SCNN	BSCNN	-0.288	-0.045	0.025	0.045	-0.081	-0.132
GrosseWentrup2009	DCNN	BDCNN	-0.06	-0.017	0.004	0.01	-0.031	-0.081

Table F.35: The metrics differences.

Metrics Differences V								
Dataset	Freq. pipeline	Bayes. pipeline	NLL	BS	AUROC	MCC	ECE	MCE
GuttmannFlury2025_MI	CSPLDA	CSPBLDA	-0.0	-0.0	0.0	-0.014	-0.0	-0.013
GuttmannFlury2025_MI	CSPSVM	CSPGP	-0.003	-0.001	0.035	-0.031	0.029	0.076
GuttmannFlury2025_MI	TSLR	TSBLR	-0.293	-0.072	0.012	0.026	-0.181	-0.152
GuttmannFlury2025_MI	TSSVM	TSGP	-0.008	-0.003	-0.058	-0.063	-0.042	-0.245
GuttmannFlury2025_MI	SCNN	BSCNN	-0.049	-0.016	0.021	-0.001	-0.044	-0.112
GuttmannFlury2025_MI	DCNN	BDCNN	0.044	0.008	-0.016	-0.038	0.019	-0.024
HefmiIch2025	CSPLDA	CSPBLDA	-0.0	-0.0	-0.002	0.003	0.005	-0.074
HefmiIch2025	CSPSVM	CSPGP	-0.003	-0.002	0.038	-0.083	-0.003	-0.071
HefmiIch2025	TSLR	TSBLR	-0.021	-0.006	-0.006	0.051	-0.048	-0.106
HefmiIch2025	TSSVM	TSGP	0.008	0.004	-0.07	-0.071	-0.015	-0.268
HefmiIch2025	SCNN	BSCNN	-0.016	-0.007	-0.004	0.006	-0.048	-0.163
HefmiIch2025	DCNN	BDCNN	-0.068	-0.016	0.008	0.029	-0.002	-0.048

Table F.36: The metrics differences.

Metrics Differences VI								
Dataset	Freq. pipeline	Bayes. pipeline	NLL	BS	AUROC	MCC	ECE	MCE
Kumar2024	CSPLDA	CSPBLDA	-0.002	0.0	0.004	0.021	-0.024	-0.034
Kumar2024	CSPSVM	CSPGP	0.011	0.005	-0.019	-0.013	0.003	0.016
Kumar2024	TSLR	TSBLR	-0.023	-0.006	0.002	0.008	-0.013	0.031
Kumar2024	TSSVM	TSGP	-0.004	-0.002	0.003	0.009	-0.004	-0.067
Kumar2024	SCNN	BSCNN	0.007	-0.004	0.004	0.029	-0.01	0.019
Kumar2024	DCNN	BDCNN	0.094	0.013	-0.002	-0.025	0.033	0.004
Lee2019-MI	CSPLDA	CSPBLDA	0.002	0.001	-0.001	0.001	0.0	-0.013
Lee2019-MI	CSPSVM	CSPGP	-0.002	-0.001	-0.003	-0.001	-0.011	0.019
Lee2019-MI	TSLR	TSBLR	-0.033	-0.006	-0.001	0.004	-0.022	-0.024
Lee2019-MI	TSSVM	TSGP	-0.011	0.006	-0.021	-0.019	-0.011	-0.027
Lee2019-MI	SCNN	BSCNN	0.012	0.001	-0.002	-0.025	0.0	0.01
Lee2019-MI	DCNN	BDCNN	0.061	0.006	0.0	0.003	0.017	0.02

Table F.37: The metrics differences.

Metrics Differences VII								
Dataset	Freq. pipeline	Bayes. pipeline	NLL	BS	AUROC	MCC	ECE	MCE
Liu2024	CSPLDA	CSPBLDA	-0.001	-0.0	0.006	0.005	-0.003	-0.147
Liu2024	CSPSVM	CSPGP	0.001	0.0	-0.021	-0.044	0.012	0.068
Liu2024	TSLR	TSBLR	-0.005	-0.002	-0.001	-0.007	-0.013	-0.02
Liu2024	TSSVM	TSGP	0.0	0.0	-0.015	-0.012	-0.006	-0.06
Liu2024	SCNN	BSCNN	-0.026	-0.011	0.019	0.025	-0.063	-0.203
Liu2024	DCNN	BDCNN	0.058	0.018	-0.03	-0.066	0.037	-0.252
PhysionetMI	CSPLDA	CSPBLDA	0.001	-0.001	0.001	0.007	0.001	0.01
PhysionetMI	CSPSVM	CSPGP	-0.003	-0.001	0.003	-0.009	0.003	0.094
PhysionetMI	TSLR	TSBLR	-0.043	-0.01	-0.003	0.002	-0.04	-0.049
PhysionetMI	TSSVM	TSGP	0.016	0.007	-0.027	-0.037	0.004	0.061
PhysionetMI	SCNN	BSCNN	0.039	-0.005	0.005	0.014	-0.015	-0.018
PhysionetMI	DCNN	BDCNN	0.07	-0.013	0.033	0.066	-0.016	-0.031

Table F.38: The metrics differences.

Metrics Differences VIII								
Dataset	Freq. pipeline	Bayes. pipeline	NLL	BS	AUROC	MCC	ECE	MCE
Schirrmeister2017	CSPLDA	CSPBLDA	-0.023	-0.006	0.002	-0.004	-0.009	0.026
Schirrmeister2017	CSPSVM	CSPGP	0.009	-0.007	-0.013	0.013	-0.029	-0.035
Schirrmeister2017	TSLR	TSBLR	-0.83	-0.084	0.042	0.08	-0.121	-0.2
Schirrmeister2017	TSSVM	TSGP	-0.296	-0.038	0.052	0.112	-0.043	-0.242
Schirrmeister2017	SCNN	BSCNN	0.004	-0.009	-0.001	0.022	-0.02	-0.058
Schirrmeister2017	DCNN	BDCNN	0.283	-0.0	0.008	0.017	0.015	0.04
Shin2017A	CSPLDA	CSPBLDA	0.007	0.0	-0.0	-0.0	0.01	0.04
Shin2017A	CSPSVM	CSPGP	0.021	0.007	-0.025	-0.02	0.003	-0.003
Shin2017A	TSLR	TSBLR	-0.236	-0.043	0.014	0.027	-0.09	-0.11
Shin2017A	TSSVM	TSGP	-0.009	-0.003	0.007	0.025	0.0	-0.087
Shin2017A	SCNN	BSCNN	-0.149	-0.03	0.02	0.035	-0.071	-0.12
Shin2017A	DCNN	BDCNN	-0.085	-0.012	0.005	0.03	-0.032	-0.036

Table F.39: The metrics differences.

Metrics Differences IX								
Dataset	Freq. pipeline	Bayes. pipeline	NLL	BS	AUROC	MCC	ECE	MCE
Stieger2021	CSPLDA	CSPBLDA	0.0	-0.0	-0.0	0.0	-0.0	-0.021
Stieger2021	CSPSVM	CSPGP	-0.003	-0.001	0.002	0.001	-0.002	0.097
Stieger2021	TSLR	TSBLR	-0.006	-0.001	0.0	-0.0	-0.004	-0.002
Stieger2021	TSSVM	TSGP	0.021	0.011	-0.043	-0.063	-0.014	0.022
Stieger2021	SCNN	BSCNN	0.025	0.003	0.003	0.003	0.018	0.023
Stieger2021	DCNN	BDCNN	0.024	0.007	-0.01	-0.02	0.01	0.024
Weibo2014	CSPLDA	CSPBLDA	0.023	0.004	-0.0	0.013	0.012	-0.076
Weibo2014	CSPSVM	CSPGP	0.023	0.003	-0.006	-0.012	-0.02	0.029
Weibo2014	TSLR	TSBLR	-0.467	-0.081	0.016	0.051	-0.158	-0.158
Weibo2014	TSSVM	TSGP	-0.032	-0.006	0.027	-0.006	0.014	0.039
Weibo2014	SCNN	BSCNN	-0.199	-0.03	0.002	0.0	-0.054	0.028
Weibo2014	DCNN	BDCNN	-0.027	-0.008	0.003	0.01	-0.014	-0.049

Table F.40: The metrics differences.

Metrics Differences X								
Dataset	Freq. pipeline	Bayes. pipeline	NLL	BS	AUROC	MCC	ECE	MCE
Yang2025	CSPLDA	CSPBLDA	0.0	0.0	-0.0	-0.0	0.0	-0.003
Yang2025	CSPSVM	CSPGP	-0.004	-0.002	0.004	-0.002	-0.012	0.052
Yang2025	TSLR	TSBLR	-0.011	-0.004	-0.0	-0.002	-0.011	-0.03
Yang2025	TSSVM	TSGP	-0.001	-0.0	-0.014	-0.025	-0.012	-0.013
Yang2025	SCNN	BSCNN	-0.01	-0.005	0.013	0.024	-0.006	-0.072
Yang2025	DCNN	BDCNN	0.081	0.009	0.001	0.003	0.037	0.044
Zhou2020	CSPLDA	CSPBLDA	0.0	0.0	-0.0	-0.006	0.001	0.024
Zhou2020	CSPSVM	CSPGP	0.004	0.001	-0.016	-0.037	-0.013	0.027
Zhou2020	TSLR	TSBLR	-0.133	-0.025	-0.005	0.025	-0.051	-0.055
Zhou2020	TSSVM	TSGP	-0.019	-0.004	-0.027	-0.024	-0.03	-0.055
Zhou2020	SCNN	BSCNN	-0.044	-0.01	0.011	0.007	-0.006	0.01
Zhou2020	DCNN	BDCNN	-0.0	-0.002	-0.002	-0.023	-0.006	-0.005

F.3 Meta-Analysis

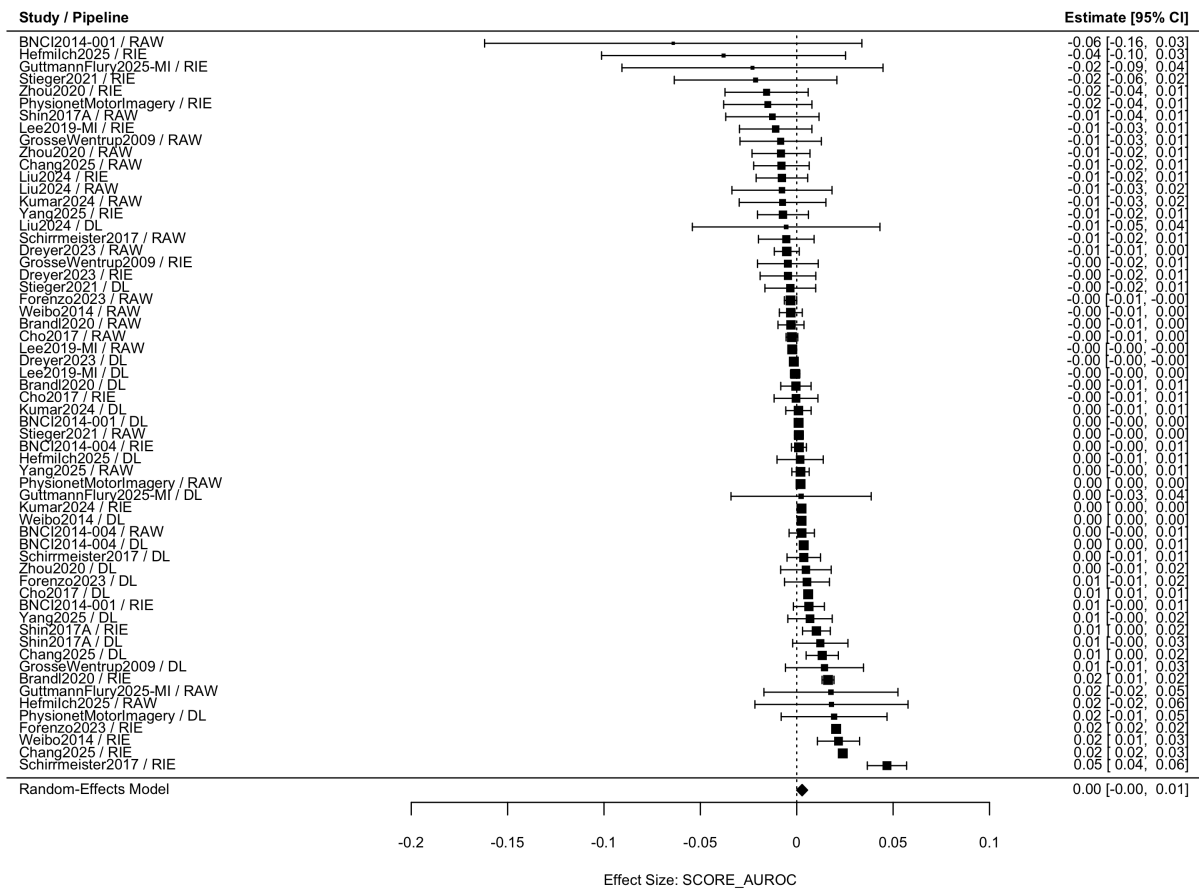


Figure F.9: The AUROC intercept-only meta-analysis forest plot.

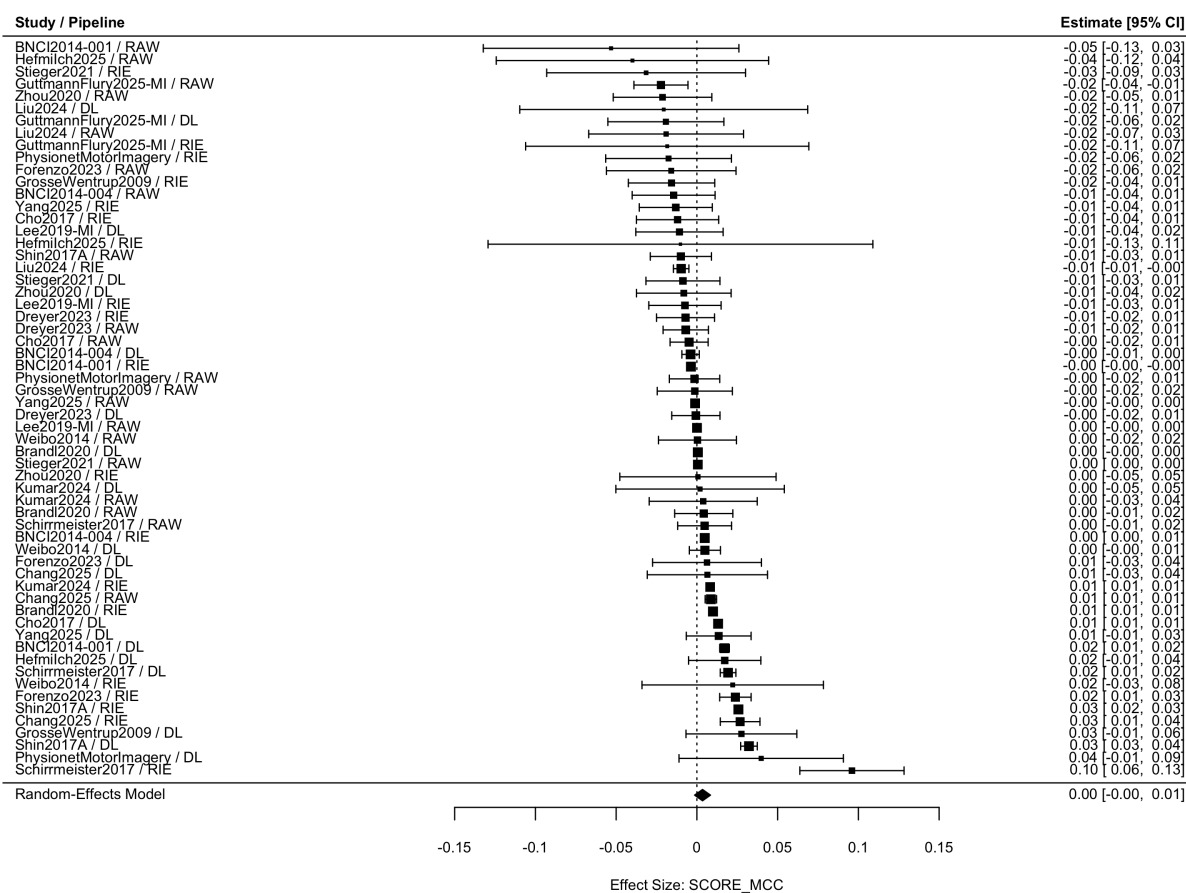


Figure F.10: The MCC intercept-only meta-analysis forest plot.

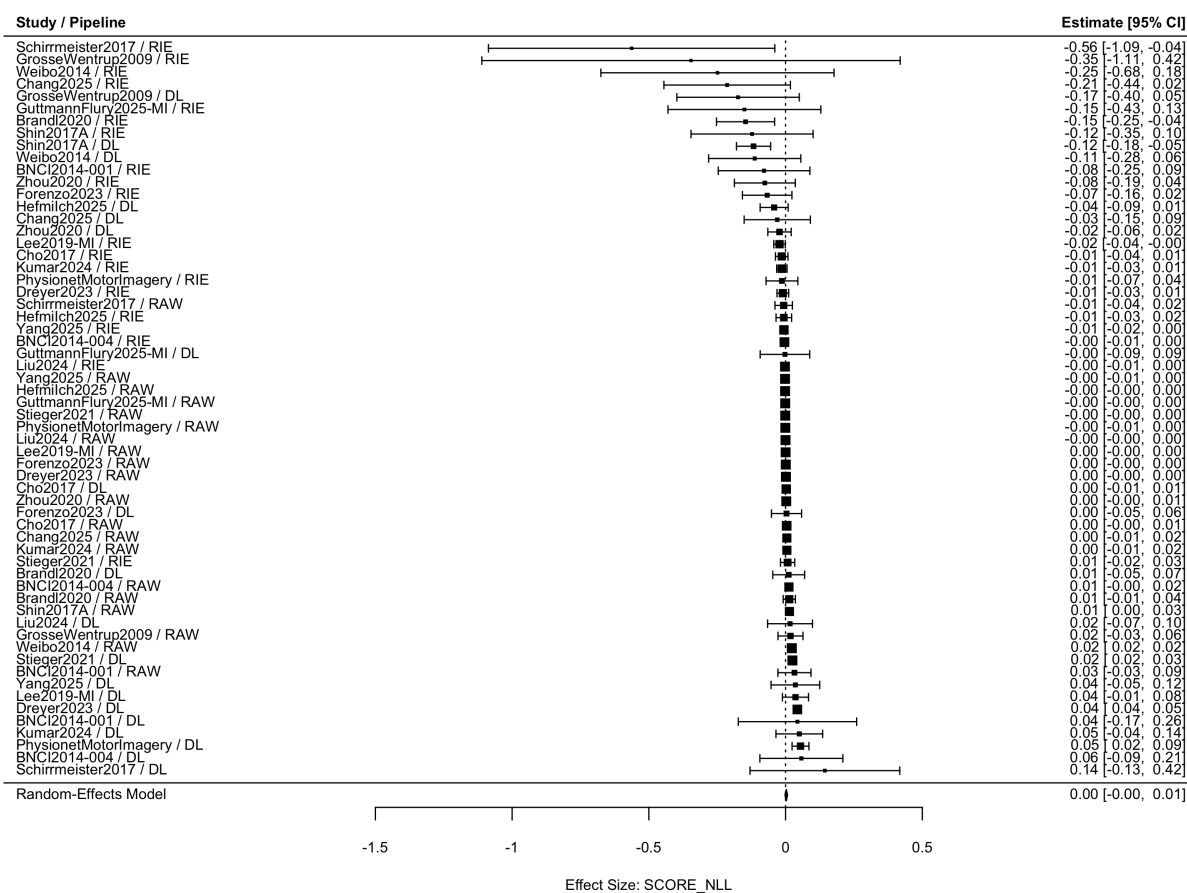


Figure F.11: The NLL intercept-only meta-analysis forest plot.

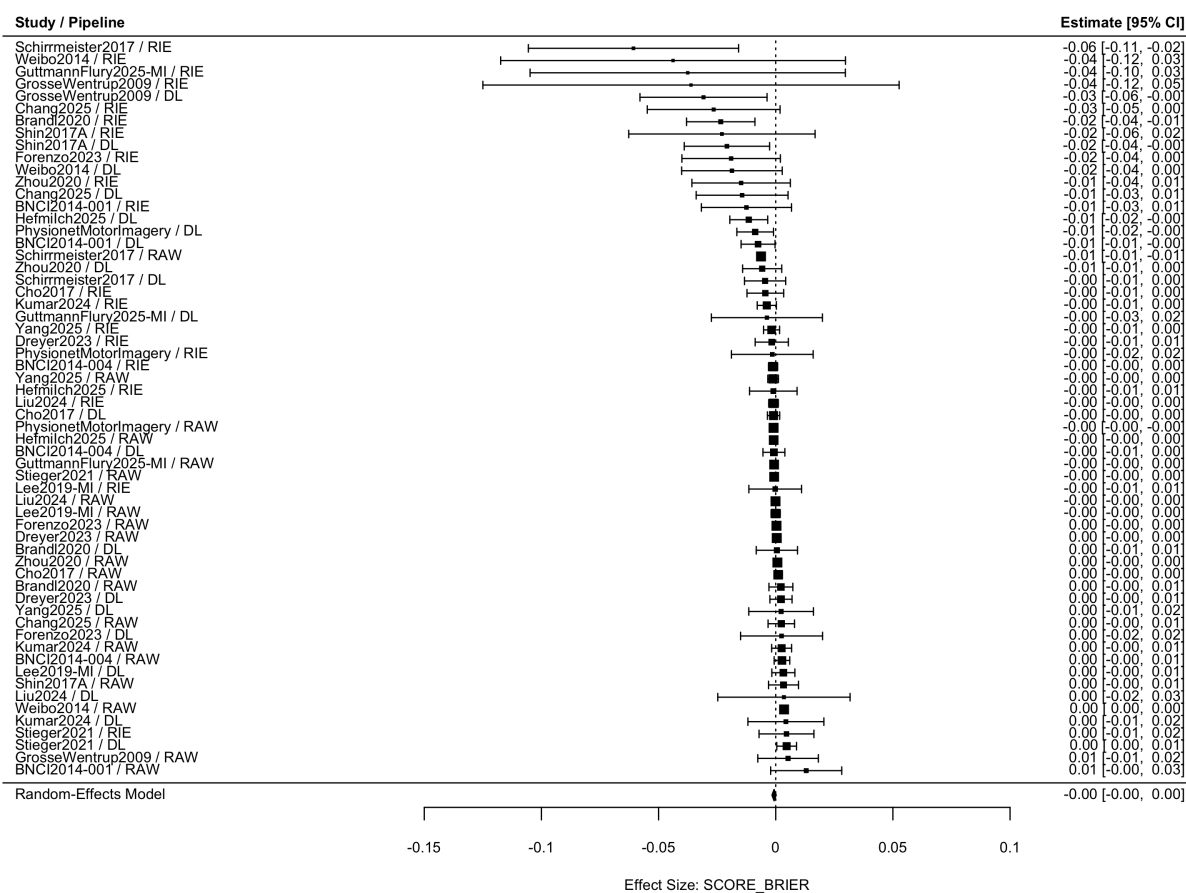


Figure F.12: The Brier score intercept-only meta-analysis forest plot.

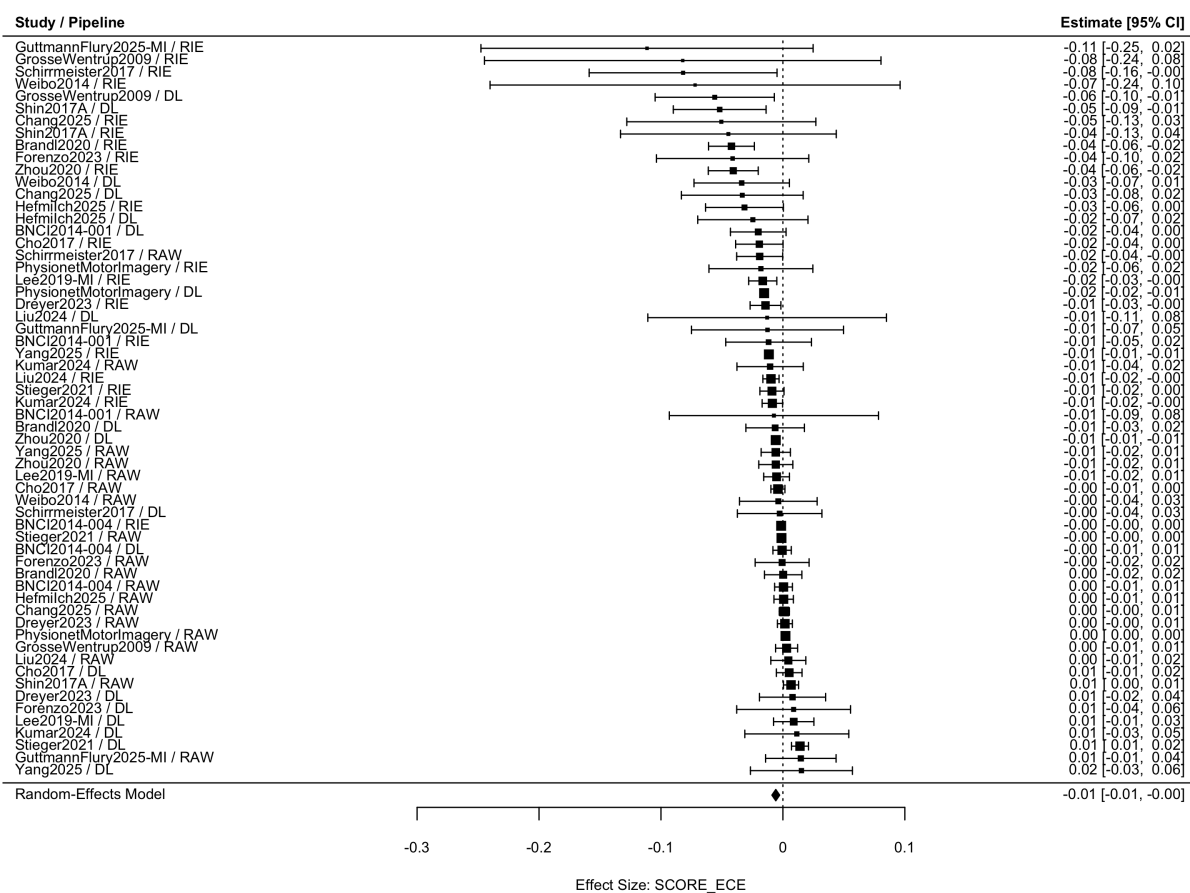


Figure F.13: The ECE intercept-only meta-analysis forest plot.

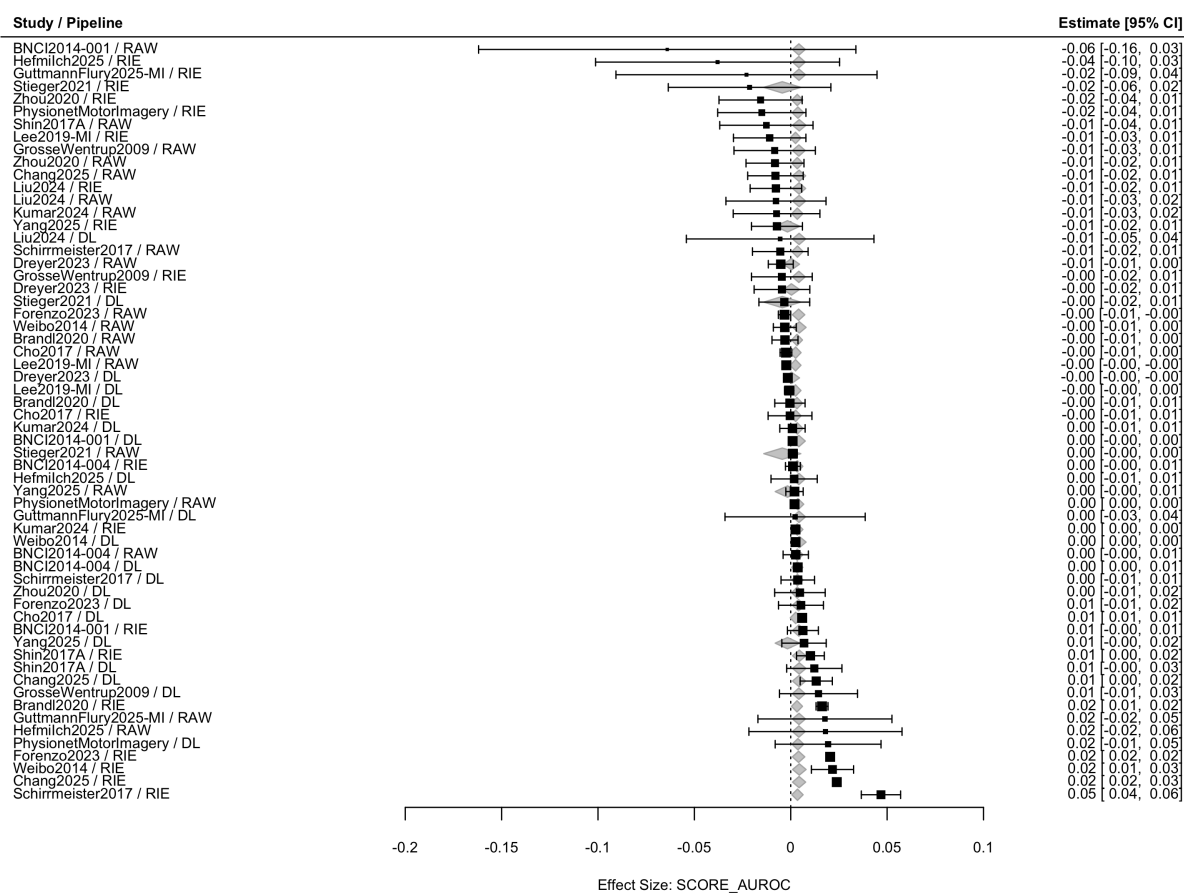


Figure F.14: The AUROC meta-regression forest plot.

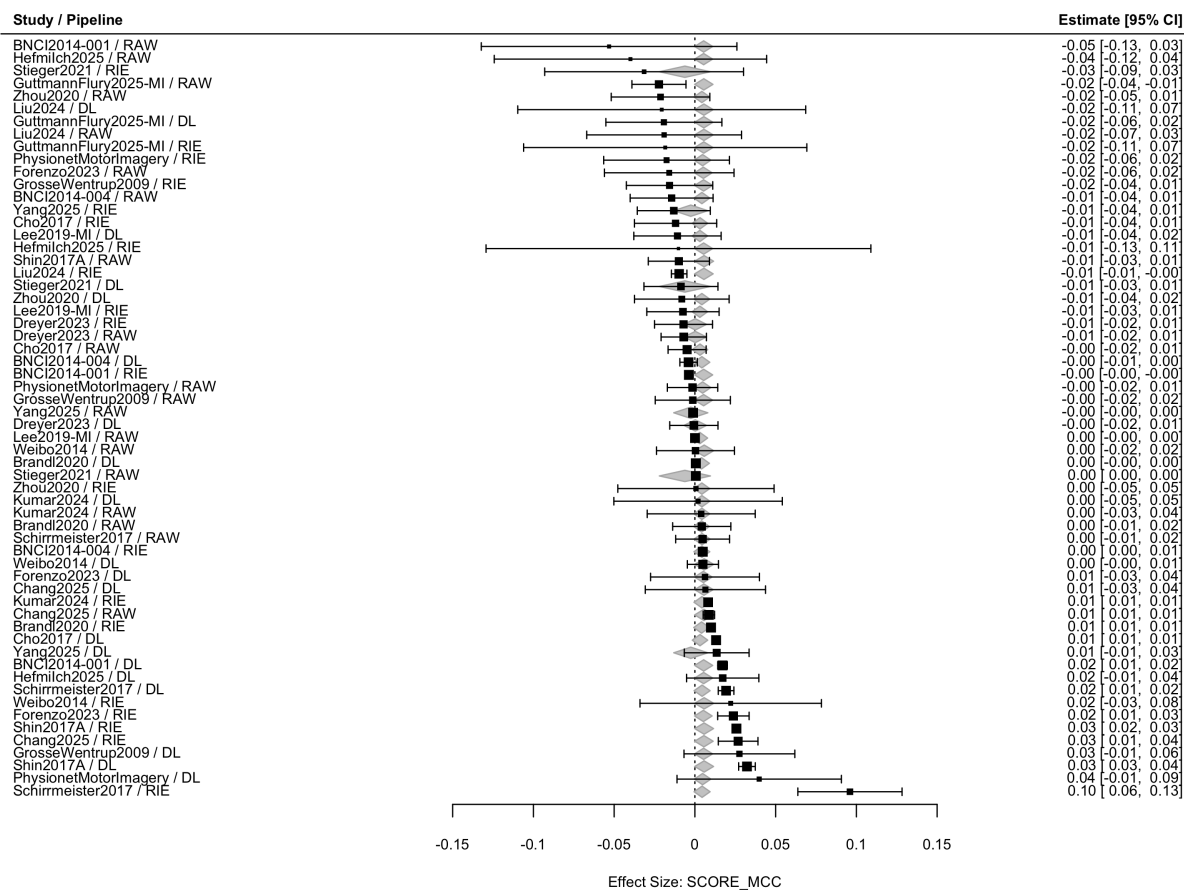


Figure F.15: The MCC meta-regression forest plot.

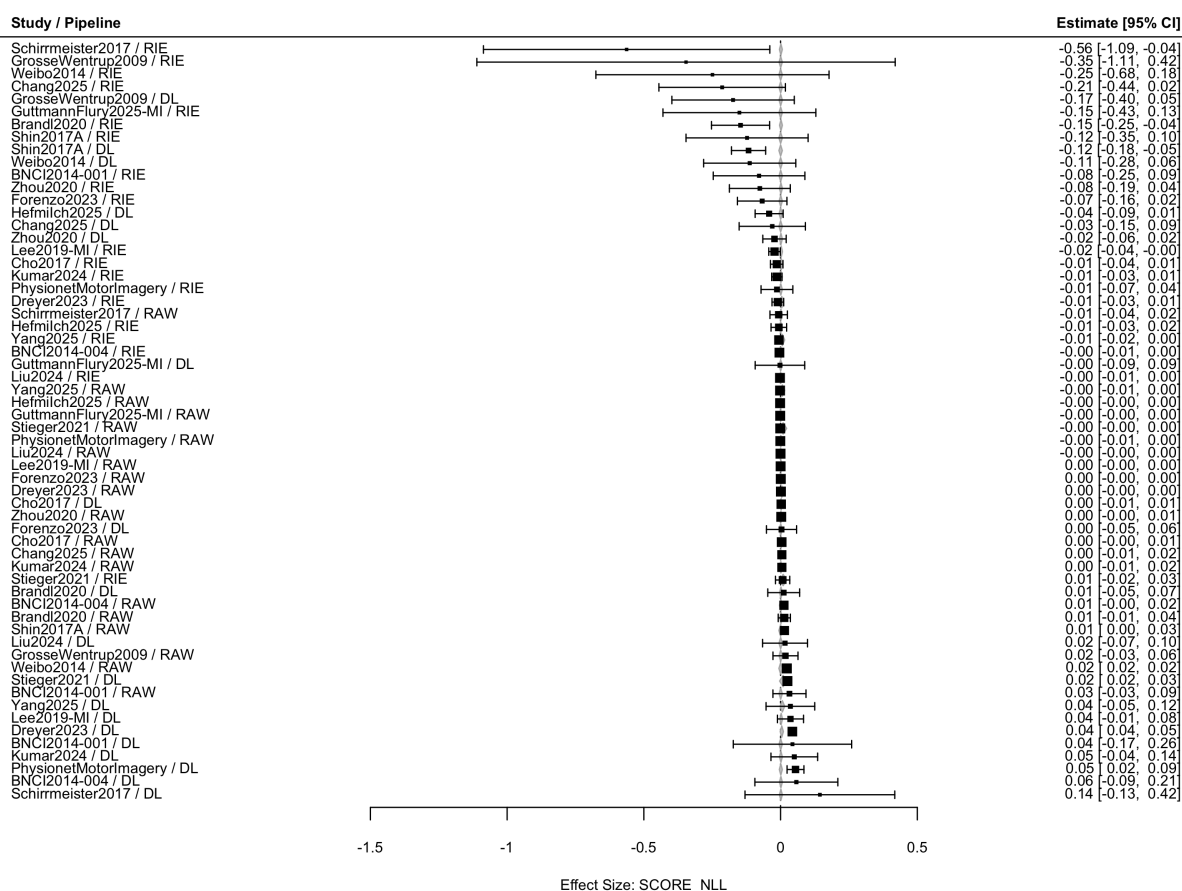


Figure F.16: The NLL meta-regression forest plot.

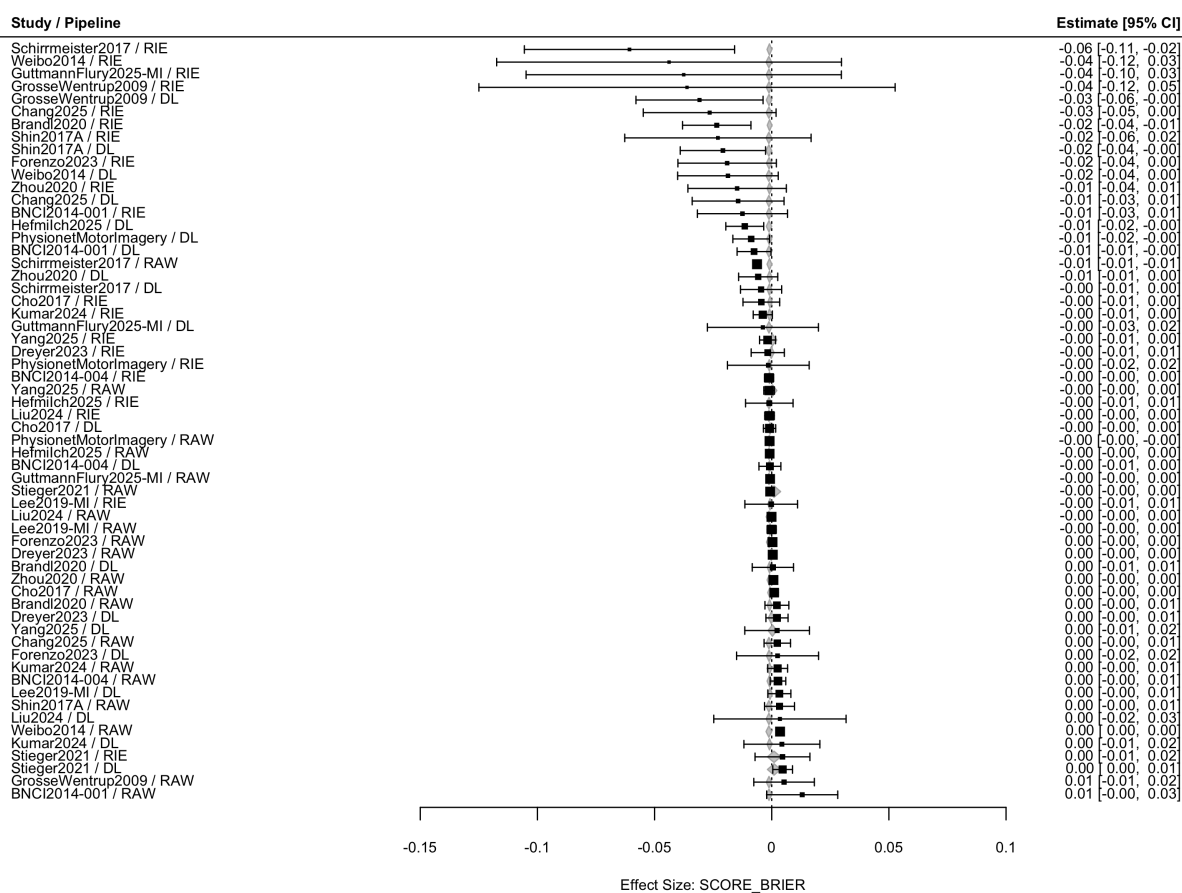


Figure F.17: The Brier score meta-regression forest plot.

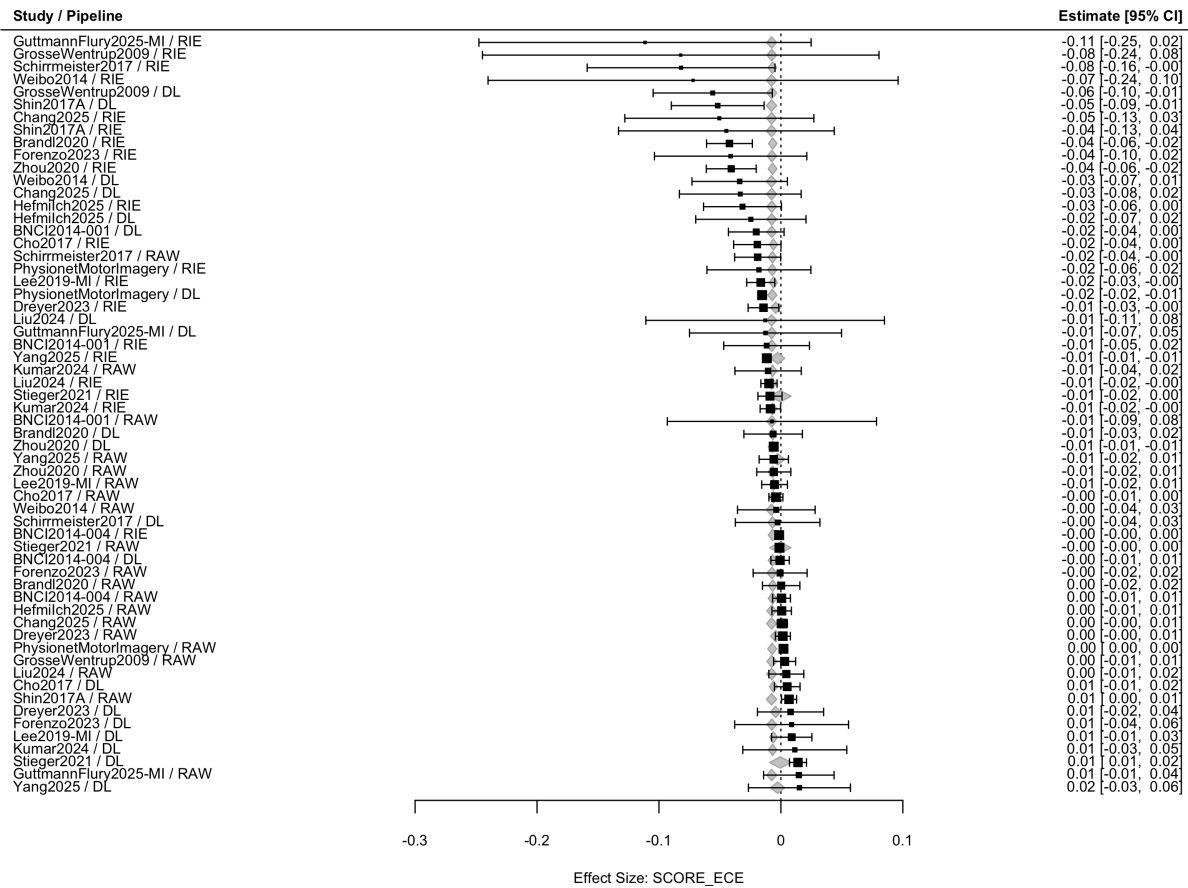


Figure F.18: The ECE meta-regression forest plot.

F.4 Computational Costs

Table F.41: The average carbon emissions across k -fold cross-validation.

Carbon Emissions: BNCI2014_001				
Dataset	Pipeline	Duration (s)	Emissions (g CO ₂ eq)	Emissions Rate (g CO ₂ eq / s)
BNCI2014_001	BDCNN	90.253	1.176	0.013
BNCI2014_001	BSCNN	110.295	1.601	0.015
BNCI2014_001	CSPBLDA	263.816	2.823	0.011
BNCI2014_001	CSPGP	233.555	3.609	0.015
BNCI2014_001	CSPLDA	7.074	0.057	0.008
BNCI2014_001	CSPSVM	7.656	0.064	0.008
BNCI2014_001	DCNN	41.488	0.494	0.012
BNCI2014_001	SCNN	30.115	0.357	0.012
BNCI2014_001	TSBLR	45.425	0.612	0.013
BNCI2014_001	TSGP	108.140	1.589	0.015
BNCI2014_001	TSLR	7.855	0.066	0.008
BNCI2014_001	TSSVM	16.061	0.133	0.008

Table F.42: The average carbon emissions across k -fold cross-validation.

Carbon Emissions: BNCI2014_004				
Dataset	Pipeline	Duration (s)	Emissions (g CO ₂ eq)	Emissions Rate (g CO ₂ eq / s)
BNCI2014_004	BDCNN	150.862	2.226	0.015
BNCI2014_004	BSCNN	210.469	3.664	0.017
BNCI2014_004	CSPBLDA	240.238	2.166	0.009
BNCI2014_004	CSPGP	333.916	5.330	0.016
BNCI2014_004	CSPLDA	7.726	0.065	0.008
BNCI2014_004	CSPSVM	11.158	0.093	0.008
BNCI2014_004	DCNN	63.742	0.808	0.013
BNCI2014_004	SCNN	61.977	0.769	0.012
BNCI2014_004	TSBLR	26.979	0.320	0.012
BNCI2014_004	TSGP	298.931	4.464	0.015
BNCI2014_004	TSLR	7.845	0.066	0.008
BNCI2014_004	TSSVM	10.588	0.088	0.008

Table F.43: The average carbon emissions across k -fold cross-validation.

Carbon Emissions: Brandl2020				
Dataset	Pipeline	Duration (s)	Emissions (g CO ₂ eq)	Emissions Rate (g CO ₂ eq / s)
Brandl2020	BDCNN	200.301	2.898	0.014
Brandl2020	BSCNN	299.041	5.263	0.018
Brandl2020	CSPBLDA	511.590	4.470	0.009
Brandl2020	CSPGP	328.077	5.317	0.016
Brandl2020	CSPLDA	8.609	0.072	0.008
Brandl2020	CSPSVM	14.016	0.116	0.008
Brandl2020	DCNN	98.956	1.233	0.012
Brandl2020	SCNN	133.890	1.938	0.014
Brandl2020	TSBLR	66.771	1.000	0.015
Brandl2020	TSGP	123.988	1.813	0.015
Brandl2020	TSLR	14.318	0.131	0.009
Brandl2020	TSSVM	89.668	0.741	0.008

Table F.44: The average carbon emissions across k -fold cross-validation.

Carbon Emissions: Chang2025				
Dataset	Pipeline	Duration (s)	Emissions (g CO ₂ eq)	Emissions Rate (g CO ₂ eq / s)
Chang2025	BDCNN	79.622	1.038	0.013
Chang2025	BSCNN	98.551	1.405	0.014
Chang2025	CSPBLDA	180.097	1.813	0.010
Chang2025	CSPGP	172.539	2.604	0.015
Chang2025	CSPLDA	6.932	0.059	0.008
Chang2025	CSPSVM	7.288	0.061	0.008
Chang2025	DCNN	31.185	0.360	0.012
Chang2025	SCNN	23.565	0.271	0.011
Chang2025	TSBLR	38.758	0.517	0.013
Chang2025	TSGP	103.334	1.504	0.015
Chang2025	TSLR	7.693	0.064	0.008
Chang2025	TSSVM	22.655	0.188	0.008

Table F.45: The average carbon emissions across k -fold cross-validation.

Carbon Emissions: Cho2017				
Dataset	Pipeline	Duration (s)	Emissions (g CO ₂ eq)	Emissions Rate (g CO ₂ eq / s)
Cho2017	BDCNN	638.232	13.252	0.021
Cho2017	BSCNN	303.608	5.388	0.018
Cho2017	CSPBLDA	718.428	6.163	0.009
Cho2017	CSPGP	336.065	5.334	0.016
Cho2017	CSPLDA	9.514	0.079	0.008
Cho2017	CSPSVM	22.722	0.188	0.008
Cho2017	DCNN	114.153	1.469	0.013
Cho2017	SCNN	124.982	1.821	0.015
Cho2017	TSBLR	71.353	1.095	0.015
Cho2017	TSGP	134.394	1.959	0.015
Cho2017	TSLR	14.837	0.129	0.009
Cho2017	TSSVM	223.952	1.853	0.008

Table F.46: The average carbon emissions across k -fold cross-validation.

Carbon Emissions: Dreyer2023				
Dataset	Pipeline	Duration (s)	Emissions (g CO ₂ eq)	Emissions Rate (g CO ₂ eq / s)
Dreyer2023	BDCNN	485.683	7.884	0.016
Dreyer2023	BSCNN	1034.410	19.343	0.019
Dreyer2023	CSPBLDA	1391.400	11.844	0.009
Dreyer2023	CSPGP	423.534	6.807	0.016
Dreyer2023	CSPLDA	13.252	0.110	0.008
Dreyer2023	CSPSVM	69.372	0.573	0.008
Dreyer2023	DCNN	293.425	3.762	0.013
Dreyer2023	SCNN	560.179	8.345	0.015
Dreyer2023	TSBLR	101.656	1.647	0.016
Dreyer2023	TSGP	187.682	2.596	0.014
Dreyer2023	TSLR	23.987	0.206	0.009
Dreyer2023	TSSVM	2097.422	17.492	0.008

Table F.47: The average carbon emissions across k -fold cross-validation.

Carbon Emissions: Forenzo2023				
Dataset	Pipeline	Duration (s)	Emissions (g CO ₂ eq)	Emissions Rate (g CO ₂ eq / s)
Forenzo2023	BDCNN	117.381	1.571	0.013
Forenzo2023	BSCNN	158.231	2.455	0.016
Forenzo2023	CSPBLDA	296.450	3.033	0.010
Forenzo2023	CSPGP	200.486	3.013	0.015
Forenzo2023	CSPLDA	7.300	0.061	0.008
Forenzo2023	CSPSVM	8.374	0.070	0.008
Forenzo2023	DCNN	60.378	0.738	0.012
Forenzo2023	SCNN	47.171	0.587	0.012
Forenzo2023	TSBLR	50.165	0.700	0.014
Forenzo2023	TSGP	113.499	1.670	0.015
Forenzo2023	TSLR	10.335	0.097	0.009
Forenzo2023	TSSVM	25.772	0.213	0.008

Table F.48: The average carbon emissions across k -fold cross-validation.

Carbon Emissions: GrosseWentrup2009				
Dataset	Pipeline	Duration (s)	Emissions (g CO ₂ eq)	Emissions Rate (g CO ₂ eq / s)
GrosseWentrup2009	BDCNN	135.620	1.909	0.014
GrosseWentrup2009	BSCNN	191.521	2.784	0.015
GrosseWentrup2009	CSPBLDA	241.155	2.405	0.010
GrosseWentrup2009	CSPGP	247.812	3.875	0.016
GrosseWentrup2009	CSPLDA	7.583	0.064	0.008
GrosseWentrup2009	CSPSVM	8.174	0.069	0.008
GrosseWentrup2009	DCNN	56.452	0.666	0.012
GrosseWentrup2009	SCNN	70.937	0.949	0.013
GrosseWentrup2009	TSBLR	63.404	0.933	0.015
GrosseWentrup2009	TSGP	120.361	1.755	0.015
GrosseWentrup2009	TSLR	14.760	0.146	0.010
GrosseWentrup2009	TSSVM	69.740	0.573	0.008

Table F.49: The average carbon emissions across k -fold cross-validation.

Carbon Emissions: GuttmanFlury2025_MI				
Dataset	Pipeline	Duration (s)	Emissions (g CO ₂ eq)	Emissions Rate (g CO ₂ eq / s)
GuttmanFlury2025_MI	BDCNN	91.415	1.212	0.013
GuttmanFlury2025_MI	BSCNN	128.629	1.908	0.015
GuttmanFlury2025_MI	CSPBLDA	199.649	1.982	0.010
GuttmanFlury2025_MI	CSPGP	140.433	2.064	0.015
GuttmanFlury2025_MI	CSPLDA	7.047	0.059	0.008
GuttmanFlury2025_MI	CSPSVM	7.622	0.064	0.008
GuttmanFlury2025_MI	DCNN	41.090	0.488	0.012
GuttmanFlury2025_MI	SCNN	38.705	0.514	0.013
GuttmanFlury2025_MI	TSBLR	51.291	0.722	0.014
GuttmanFlury2025_MI	TSGP	93.805	1.326	0.014
GuttmanFlury2025_MI	TSLR	10.019	0.095	0.009
GuttmanFlury2025_MI	TSSVM	18.626	0.155	0.008

Table F.50: The average carbon emissions across k -fold cross-validation.

Carbon Emissions: HefmiIch2025				
Dataset	Pipeline	Duration (s)	Emissions (g CO ₂ eq)	Emissions Rate (g CO ₂ eq / s)
HefmiIch2025	BDCNN	206.660	3.286	0.016
HefmiIch2025	BSCNN	238.749	3.734	0.016
HefmiIch2025	CSPBLDA	237.846	2.408	0.010
HefmiIch2025	CSPGP	135.279	1.985	0.015
HefmiIch2025	CSPLDA	7.070	0.059	0.008
HefmiIch2025	CSPSVM	7.889	0.067	0.008
HefmiIch2025	DCNN	48.596	0.620	0.013
HefmiIch2025	SCNN	36.113	0.451	0.012
HefmiIch2025	TSBLR	46.381	0.628	0.014
HefmiIch2025	TSGP	113.062	1.622	0.014
HefmiIch2025	TSLR	7.414	0.062	0.008
HefmiIch2025	TSSVM	9.169	0.077	0.008

Table F.51: The average carbon emissions across k -fold cross-validation.

Carbon Emissions: Kumar2024				
Dataset	Pipeline	Duration (s)	Emissions (g CO ₂ eq)	Emissions Rate (g CO ₂ eq / s)
Kumar2024	BDCNN	180.217	2.654	0.015
Kumar2024	BSCNN	229.977	3.908	0.017
Kumar2024	CSPBLDA	446.366	3.893	0.009
Kumar2024	CSPGP	302.645	4.790	0.016
Kumar2024	CSPLDA	7.978	0.067	0.008
Kumar2024	CSPSVM	12.362	0.102	0.008
Kumar2024	DCNN	92.861	1.196	0.013
Kumar2024	SCNN	74.237	0.954	0.013
Kumar2024	TSBLR	39.498	0.521	0.013
Kumar2024	TSGP	122.173	1.808	0.015
Kumar2024	TSLR	8.794	0.074	0.008
Kumar2024	TSSVM	25.077	0.209	0.008

Table F.52: The average carbon emissions across k -fold cross-validation.

Carbon Emissions: Lee2019_MI				
Dataset	Pipeline	Duration (s)	Emissions (g CO ₂ eq)	Emissions Rate (g CO ₂ eq / s)
Lee2019_MI	BDCNN	216.814	2.995	0.014
Lee2019_MI	BSCNN	308.352	5.708	0.019
Lee2019_MI	CSPBLDA	734.331	6.304	0.009
Lee2019_MI	CSPGP	431.066	7.104	0.016
Lee2019_MI	CSPLDA	9.668	0.081	0.008
Lee2019_MI	CSPSVM	22.088	0.183	0.008
Lee2019_MI	DCNN	143.661	1.816	0.013
Lee2019_MI	SCNN	104.502	1.360	0.013
Lee2019_MI	TSBLR	71.814	1.087	0.015
Lee2019_MI	TSGP	131.658	1.921	0.015
Lee2019_MI	TSLR	15.248	0.133	0.009
Lee2019_MI	TSSVM	172.659	1.428	0.008

Table F.53: The average carbon emissions across k -fold cross-validation.

Carbon Emissions: Liu2024				
Dataset	Pipeline	Duration (s)	Emissions (g CO ₂ eq)	Emissions Rate (g CO ₂ eq / s)
Liu2024	BDCNN	85.462	1.138	0.013
Liu2024	BSCNN	118.404	1.735	0.015
Liu2024	CSPBLDA	192.694	2.002	0.010
Liu2024	CSPGP	157.702	2.350	0.015
Liu2024	CSPLDA	6.775	0.057	0.008
Liu2024	CSPSVM	7.250	0.060	0.008
Liu2024	DCNN	32.345	0.383	0.012
Liu2024	SCNN	21.416	0.242	0.011
Liu2024	TSBLR	33.088	0.427	0.013
Liu2024	TSGP	105.267	1.535	0.015
Liu2024	TSLR	6.920	0.058	0.008
Liu2024	TSSVM	7.773	0.065	0.008

Table F.54: The average carbon emissions across k -fold cross-validation.

Carbon Emissions: PhysionetMI				
Dataset	Pipeline	Duration (s)	Emissions (g CO ₂ eq)	Emissions Rate (g CO ₂ eq / s)
PhysionetMI	BDCNN	338.487	6.100	0.018
PhysionetMI	BSCNN	186.867	3.110	0.017
PhysionetMI	CSPBLDA	362.152	3.268	0.009
PhysionetMI	CSPGP	313.780	4.952	0.016
PhysionetMI	CSPLDA	7.780	0.065	0.008
PhysionetMI	CSPSVM	10.401	0.087	0.008
PhysionetMI	DCNN	59.850	0.733	0.012
PhysionetMI	SCNN	71.450	1.009	0.014
PhysionetMI	TSBLR	59.155	0.867	0.015
PhysionetMI	TSGP	116.206	1.709	0.015
PhysionetMI	TSLR	11.555	0.106	0.009
PhysionetMI	TSSVM	63.518	0.524	0.008

Table F.55: The average carbon emissions across k -fold cross-validation.

Carbon Emissions: Schirrmeister2017				
Dataset	Pipeline	Duration (s)	Emissions (g CO ₂ eq)	Emissions Rate (g CO ₂ eq / s)
Schirrmeister2017	BDCNN	156.865	2.038	0.013
Schirrmeister2017	BSCNN	207.080	3.449	0.017
Schirrmeister2017	CSPBLDA	421.971	3.689	0.009
Schirrmeister2017	CSPGP	321.049	5.138	0.016
Schirrmeister2017	CSPLDA	9.001	0.076	0.008
Schirrmeister2017	CSPSVM	12.466	0.104	0.008
Schirrmeister2017	DCNN	100.645	1.221	0.012
Schirrmeister2017	SCNN	69.168	0.876	0.013
Schirrmeister2017	TSBLR	83.022	1.322	0.016
Schirrmeister2017	TSGP	145.943	2.054	0.014
Schirrmeister2017	TSLR	20.190	0.188	0.009
Schirrmeister2017	TSSVM	216.024	1.777	0.008

Table F.56: The average carbon emissions across k -fold cross-validation.

Carbon Emissions: Shin2017A				
Dataset	Pipeline	Duration (s)	Emissions (g CO ₂ eq)	Emissions Rate (g CO ₂ eq / s)
Shin2017A	BDCNN	132.530	1.790	0.014
Shin2017A	BSCNN	169.354	2.199	0.013
Shin2017A	CSPBLDA	149.076	1.532	0.010
Shin2017A	CSPGP	202.950	3.064	0.015
Shin2017A	CSPLDA	6.703	0.056	0.008
Shin2017A	CSPSVM	7.019	0.060	0.008
Shin2017A	DCNN	29.569	0.360	0.012
Shin2017A	SCNN	23.712	0.278	0.012
Shin2017A	TSBLR	33.085	0.426	0.013
Shin2017A	TSGP	120.230	1.790	0.015
Shin2017A	TSLR	6.824	0.057	0.008
Shin2017A	TSSVM	7.379	0.062	0.008

Table F.57: The average carbon emissions across k -fold cross-validation.

Carbon Emissions V				
Dataset	Pipeline	Duration (s)	Emissions (g CO ₂ eq)	Emissions Rate (g CO ₂ eq / s)
Stieger2021	BDCNN	1858.538	39.555	0.021
Stieger2021	BSCNN	1512.804	30.690	0.020
Stieger2021	CSPBLDA	2908.654	12.242	0.004
Stieger2021	CSPGP	958.800	16.727	0.017
Stieger2021	CSPLDA	19.602	0.162	0.008
Stieger2021	CSPSVM	269.052	2.224	0.008
Stieger2021	DCNN	503.736	6.688	0.013
Stieger2021	SCNN	806.079	12.254	0.015
Stieger2021	TSBLR	152.593	2.637	0.017
Stieger2021	TSGP	359.470	4.582	0.013
Stieger2021	TSLR	39.068	0.330	0.008
Stieger2021	TSSVM	30967.842	109.434	0.004

Table F.58: The average carbon emissions across k -fold cross-validation.

Carbon Emissions: Weibo2014				
Dataset	Pipeline	Duration (s)	Emissions (g CO ₂ eq)	Emissions Rate (g CO ₂ eq / s)
Weibo2014	BDCNN	73.374	0.939	0.013
Weibo2014	BSCNN	95.042	1.327	0.014
Weibo2014	CSPBLDA	145.050	1.532	0.011
Weibo2014	CSPGP	171.538	2.573	0.015
Weibo2014	CSPLDA	6.802	0.057	0.008
Weibo2014	CSPSVM	6.923	0.058	0.008
Weibo2014	DCNN	27.665	0.316	0.011
Weibo2014	SCNN	28.814	0.367	0.013
Weibo2014	TSBLR	41.409	0.545	0.013
Weibo2014	TSGP	109.883	1.617	0.015
Weibo2014	TSLR	7.566	0.063	0.008
Weibo2014	TSSVM	11.966	0.099	0.008

Table F.59: The average carbon emissions across k -fold cross-validation.

Carbon Emissions: Yang2025				
Dataset	Pipeline	Duration (s)	Emissions (g CO ₂ eq)	Emissions Rate (g CO ₂ eq / s)
Yang2025	BDCNN	629.163	8.840	0.014
Yang2025	BSCNN	930.118	21.647	0.023
Yang2025	CSPBLDA	2048.867	17.302	0.008
Yang2025	CSPGP	633.898	10.494	0.017
Yang2025	CSPLDA	15.855	0.132	0.008
Yang2025	CSPSVM	145.765	1.203	0.008
Yang2025	DCNN	508.366	6.556	0.013
Yang2025	SCNN	242.688	3.214	0.013
Yang2025	TSBLR	118.738	2.003	0.017
Yang2025	TSGP	245.301	3.326	0.014
Yang2025	TSLR	28.427	0.241	0.008
Yang2025	TSSVM	5107.372	22.225	0.004

Table F.60: The average carbon emissions across k -fold cross-validation.

Carbon Emissions: Zhou2020				
Dataset	Pipeline	Duration (s)	Emissions (g CO ₂ eq)	Emissions Rate (g CO ₂ eq / s)
Zhou2020	BDCNN	202.243	3.175	0.016
Zhou2020	BSCNN	272.483	4.834	0.018
Zhou2020	CSPBLDA	459.260	4.009	0.009
Zhou2020	CSPGP	325.897	5.218	0.016
Zhou2020	CSPLDA	8.500	0.071	0.008
Zhou2020	CSPSVM	13.229	0.110	0.008
Zhou2020	DCNN	90.958	1.126	0.012
Zhou2020	SCNN	107.192	1.531	0.014
Zhou2020	TSBLR	65.800	0.978	0.015
Zhou2020	TSGP	122.312	1.791	0.015
Zhou2020	TSLR	13.336	0.120	0.009
Zhou2020	TSSVM	79.670	0.660	0.008

Table F.61: The average power and energy usage across k -fold cross-validation.

Power & Energy Usage: BNCI2014_001							
Dataset	Pipeline	CPU (W)	GPU (W)	CPU (Wh)	GPU (Wh)	RAM (Wh)	Total (Wh)
BNCI2014_001	BDCNN	2.966	214.492	6.746	5.377	1.755	13.878
BNCI2014_001	BSCNN	2.273	295.061	7.705	9.040	2.145	18.890
BNCI2014_001	CSPBLDA	0.835	163.841	16.132	12.048	5.129	33.309
BNCI2014_001	CSPGP	1.050	341.674	15.828	22.212	4.541	42.580
BNCI2014_001	CSPLDA	27.698	54.476	0.429	0.109	0.138	0.675
BNCI2014_001	CSPSVM	27.019	70.026	0.460	0.149	0.149	0.757
BNCI2014_001	DCNN	7.422	125.230	3.575	1.443	0.807	5.825
BNCI2014_001	SCNN	9.940	128.773	2.545	1.078	0.586	4.208
BNCI2014_001	TSBLR	5.056	270.295	2.922	3.413	0.883	7.217
BNCI2014_001	TSGP	2.221	313.532	7.233	9.418	2.103	18.753
BNCI2014_001	TSLR	26.383	70.639	0.472	0.154	0.153	0.779
BNCI2014_001	TSSVM	12.771	70.197	0.947	0.313	0.312	1.573

Table F.62: The average power and energy usage across k -fold cross-validation.

Power & Energy Usage: BNCI2014_004							
Dataset	Pipeline	CPU (W)	GPU (W)	CPU (Wh)	GPU (Wh)	RAM (Wh)	Total (Wh)
BNCI2014_004	BDCNN	1.826	281.117	11.529	11.803	2.933	26.266
BNCI2014_004	BSCNN	1.233	410.364	15.190	23.955	4.092	43.238
BNCI2014_004	CSPBLDA	0.887	99.413	14.257	6.634	4.671	25.562
BNCI2014_004	CSPGP	0.731	364.137	22.622	33.776	6.493	62.890
BNCI2014_004	CSPLDA	26.960	69.717	0.463	0.150	0.150	0.763
BNCI2014_004	CSPSVM	17.848	69.868	0.664	0.217	0.217	1.097
BNCI2014_004	DCNN	5.110	141.464	5.790	2.505	1.239	9.534
BNCI2014_004	SCNN	5.361	129.647	5.628	2.236	1.205	9.069
BNCI2014_004	TSBLR	8.229	208.336	1.692	1.561	0.525	3.778
BNCI2014_004	TSGP	0.799	328.516	19.620	27.240	5.813	52.672
BNCI2014_004	TSLR	26.427	69.574	0.473	0.152	0.153	0.777
BNCI2014_004	TSSVM	19.486	69.385	0.630	0.204	0.206	1.040

Table F.63: The average power and energy usage across k -fold cross-validation.

Power & Energy Usage: Brandl2020							
Dataset	Pipeline	CPU (W)	GPU (W)	CPU (Wh)	GPU (Wh)	RAM (Wh)	Total (Wh)
Brandl2020	BDCNN	1.409	261.681	15.734	14.567	3.895	34.196
Brandl2020	BSCNN	0.894	410.960	22.193	34.089	5.815	62.096
Brandl2020	CSPBLDA	0.416	88.930	30.169	12.630	9.947	52.746
Brandl2020	CSPGP	0.746	373.409	22.328	34.034	6.379	62.741
Brandl2020	CSPLDA	23.748	70.460	0.511	0.168	0.167	0.847
Brandl2020	CSPSVM	14.588	70.282	0.828	0.274	0.273	1.374
Brandl2020	DCNN	3.268	133.768	8.944	3.677	1.924	14.545
Brandl2020	SCNN	2.293	241.186	11.282	8.977	2.603	22.862
Brandl2020	TSBLR	3.488	331.805	4.347	6.154	1.298	11.799
Brandl2020	TSGP	2.014	300.419	8.641	10.345	2.411	21.397
Brandl2020	TSLR	16.807	70.262	0.989	0.279	0.278	1.547
Brandl2020	TSSVM	2.344	70.141	5.247	1.748	1.744	8.738

Table F.64: The average power and energy usage across k -fold cross-validation.

Power & Energy Usage: Chang2025							
Dataset	Pipeline	CPU (W)	GPU (W)	CPU (Wh)	GPU (Wh)	RAM (Wh)	Total (Wh)
Chang2025	BDCNN	3.278	221.630	5.797	4.907	1.548	12.252
Chang2025	BSCNN	2.478	289.739	6.734	7.931	1.916	16.582
Chang2025	CSPBLDA	1.206	140.159	10.885	7.011	3.502	21.397
Chang2025	CSPGP	1.428	327.305	11.700	15.676	3.355	30.731
Chang2025	CSPLDA	30.537	71.433	0.421	0.138	0.135	0.693
Chang2025	CSPSVM	26.922	69.774	0.436	0.141	0.142	0.719
Chang2025	DCNN	9.471	120.712	2.592	1.046	0.606	4.244
Chang2025	SCNN	12.216	123.879	1.924	0.811	0.458	3.193
Chang2025	TSBLR	5.932	264.940	2.491	2.852	0.754	6.097
Chang2025	TSGP	2.309	308.529	6.885	8.857	2.009	17.751
Chang2025	TSLR	26.953	68.984	0.461	0.147	0.150	0.758
Chang2025	TSSVM	9.205	70.884	1.333	0.446	0.440	2.220

Table F.65: The average power and energy usage across k -fold cross-validation.

Power & Energy Usage: Cho2017							
Dataset	Pipeline	CPU (W)	GPU (W)	CPU (Wh)	GPU (Wh)	RAM (Wh)	Total (Wh)
Cho2017	BDCNN	0.402	557.596	44.715	99.234	12.410	156.359
Cho2017	BSCNN	0.904	410.983	23.048	34.620	5.903	63.571
Cho2017	CSPBLDA	0.295	82.431	42.306	16.449	13.969	72.724
Cho2017	CSPGP	0.729	360.179	22.704	33.700	6.534	62.939
Cho2017	CSPLDA	21.483	69.909	0.568	0.185	0.185	0.937
Cho2017	CSPSVM	9.173	69.964	1.337	0.442	0.442	2.220
Cho2017	DCNN	2.908	143.764	10.559	4.556	2.220	17.335
Cho2017	SCNN	2.593	224.237	11.277	7.785	2.430	21.492
Cho2017	TSBLR	3.280	346.363	4.667	6.865	1.387	12.919
Cho2017	TSGP	1.905	292.419	9.584	10.915	2.613	23.112
Cho2017	TSLR	14.874	69.206	0.944	0.285	0.288	1.517
Cho2017	TSSVM	0.946	69.611	13.177	4.330	4.355	21.861

Table F.66: The average power and energy usage across k -fold cross-validation.

Power & Energy Usage: Dreyer2023							
Dataset	Pipeline	CPU (W)	GPU (W)	CPU (Wh)	GPU (Wh)	RAM (Wh)	Total (Wh)
Dreyer2023	BDCNN	0.613	322.762	40.053	43.524	9.444	93.020
Dreyer2023	BSCNN	0.268	448.977	79.180	128.943	20.113	228.236
Dreyer2023	CSPBLDA	0.153	78.988	82.173	30.528	27.055	139.756
Dreyer2023	CSPGP	0.589	367.301	28.363	43.717	8.235	80.315
Dreyer2023	CSPLDA	15.225	69.940	0.785	0.257	0.258	1.300
Dreyer2023	CSPSVM	3.027	69.818	4.069	1.346	1.349	6.764
Dreyer2023	DCNN	1.144	139.838	27.289	11.398	5.705	44.392
Dreyer2023	SCNN	0.551	255.937	47.760	39.815	10.892	98.467
Dreyer2023	TSBLR	2.319	381.490	6.684	10.771	1.976	19.432
Dreyer2023	TSGP	1.414	252.737	13.826	13.160	3.649	30.634
Dreyer2023	TSLR	9.207	70.619	1.497	0.471	0.466	2.434
Dreyer2023	TSSVM	0.102	69.632	125.037	40.570	40.783	206.390

Table F.67: The average power and energy usage across k -fold cross-validation.

Power & Energy Usage: Forenzo2023							
Dataset	Pipeline	CPU (W)	GPU (W)	CPU (Wh)	GPU (Wh)	RAM (Wh)	Total (Wh)
Forenzo2023	BDCNN	2.385	218.335	9.140	7.119	2.282	18.541
Forenzo2023	BSCNN	1.647	327.869	11.483	14.409	3.077	28.969
Forenzo2023	CSPBLDA	0.737	145.966	18.002	12.026	5.764	35.791
Forenzo2023	CSPGP	1.231	325.902	13.591	18.066	3.898	35.555
Forenzo2023	CSPLDA	27.271	69.218	0.442	0.140	0.142	0.725
Forenzo2023	CSPSVM	24.108	69.551	0.505	0.162	0.163	0.829
Forenzo2023	DCNN	5.332	127.357	5.391	2.137	1.174	8.702
Forenzo2023	SCNN	6.924	135.728	4.232	1.781	0.917	6.931
Forenzo2023	TSBLR	4.603	289.549	3.243	4.038	0.975	8.256
Forenzo2023	TSGP	2.123	313.609	7.616	9.888	2.207	19.710
Forenzo2023	TSLR	24.516	69.622	0.746	0.200	0.201	1.147
Forenzo2023	TSSVM	8.055	69.832	1.518	0.500	0.501	2.519

Table F.68: The average power and energy usage across k -fold cross-validation.

Power & Energy Usage: GrosseWentrup2009							
Dataset	Pipeline	CPU (W)	GPU (W)	CPU (Wh)	GPU (Wh)	RAM (Wh)	Total (Wh)
GrosseWentrup2009	BDCNN	1.890	270.734	9.691	10.197	2.637	22.525
GrosseWentrup2009	BSCNN	1.267	304.886	12.903	16.225	3.724	32.852
GrosseWentrup2009	CSPBLDA	0.901	136.042	14.577	9.110	4.689	28.376
GrosseWentrup2009	CSPGP	1.001	348.120	16.883	24.017	4.818	45.719
GrosseWentrup2009	CSPLDA	27.426	69.678	0.462	0.147	0.147	0.756
GrosseWentrup2009	CSPSVM	24.182	69.124	0.494	0.157	0.159	0.810
GrosseWentrup2009	DCNN	5.147	138.709	4.584	2.176	1.098	7.857
GrosseWentrup2009	SCNN	3.785	228.931	5.305	4.509	1.379	11.193
GrosseWentrup2009	TSBLR	3.639	323.087	4.089	5.691	1.233	11.012
GrosseWentrup2009	TSGP	2.039	303.339	8.221	10.142	2.340	20.703
GrosseWentrup2009	TSLR	18.528	68.909	1.152	0.283	0.287	1.722
GrosseWentrup2009	TSSVM	3.046	68.682	4.077	1.331	1.356	6.764

Table F.69: The average power and energy usage across k -fold cross-validation.

Power & Energy Usage GuttmanFlury2025_MI							
Dataset	Pipeline	CPU (W)	GPU (W)	CPU (Wh)	GPU (Wh)	RAM (Wh)	Total (Wh)
GuttmanFlury2025_MI	BDCNN	2.914	225.493	6.753	5.765	1.778	14.296
GuttmanFlury2025_MI	BSCNN	1.944	310.111	8.928	11.084	2.501	22.513
GuttmanFlury2025_MI	CSPBLDA	1.086	134.679	12.032	7.468	3.882	23.382
GuttmanFlury2025_MI	CSPGP	1.771	309.189	9.527	12.091	2.730	24.348
GuttmanFlury2025_MI	CSPLDA	29.376	69.224	0.424	0.136	0.137	0.696
GuttmanFlury2025_MI	CSPSVM	27.188	69.868	0.461	0.148	0.148	0.757
GuttmanFlury2025_MI	DCNN	7.469	125.000	3.536	1.427	0.799	5.762
GuttmanFlury2025_MI	SCNN	7.335	206.979	3.083	2.227	0.753	6.063
GuttmanFlury2025_MI	TSBLR	4.514	293.861	3.332	4.187	0.997	8.516
GuttmanFlury2025_MI	TSGP	2.537	291.902	6.210	7.608	1.824	15.642
GuttmanFlury2025_MI	TSLR	24.692	69.621	0.727	0.194	0.195	1.115
GuttmanFlury2025_MI	TSSVM	11.217	70.274	1.101	0.364	0.362	1.827

Table F.70: The average power and energy usage across k -fold cross-validation.

Power & Energy Usage: HefmiIch2025							
Dataset	Pipeline	CPU (W)	GPU (W)	CPU (Wh)	GPU (Wh)	RAM (Wh)	Total (Wh)
HefmiIch2025	BDCNN	1.200	356.947	14.274	20.478	4.018	38.770
HefmiIch2025	BSCNN	1.003	354.354	15.916	23.501	4.642	44.058
HefmiIch2025	CSPBLDA	0.913	142.426	14.379	9.413	4.625	28.416
HefmiIch2025	CSPGP	1.821	307.369	9.230	11.561	2.630	23.422
HefmiIch2025	CSPLDA	26.987	68.789	0.424	0.135	0.137	0.697
HefmiIch2025	CSPSVM	26.911	69.867	0.483	0.153	0.153	0.790
HefmiIch2025	DCNN	6.324	170.421	4.068	2.307	0.945	7.320
HefmiIch2025	SCNN	8.411	154.137	3.070	1.546	0.702	5.318
HefmiIch2025	TSBLR	4.992	271.647	3.009	3.500	0.902	7.411
HefmiIch2025	TSGP	2.124	304.238	7.408	9.533	2.198	19.140
HefmiIch2025	TSLR	27.247	69.918	0.449	0.144	0.144	0.737
HefmiIch2025	TSSVM	21.593	70.844	0.550	0.180	0.178	0.909

Table F.71: The average power and energy usage across k -fold cross-validation.

Power & Energy Usage: Kumar2024							
Dataset	Pipeline	CPU (W)	GPU (W)	CPU (Wh)	GPU (Wh)	RAM (Wh)	Total (Wh)
Kumar2024	BDCNN	1.610	264.883	14.555	13.252	3.504	31.312
Kumar2024	BSCNN	1.176	380.953	17.233	24.403	4.472	46.108
Kumar2024	CSPBLDA	0.477	87.735	26.381	10.876	8.679	45.936
Kumar2024	CSPGP	0.810	357.816	20.519	30.121	5.885	56.524
Kumar2024	CSPLDA	25.745	69.297	0.477	0.154	0.155	0.786
Kumar2024	CSPSVM	16.598	68.731	0.729	0.236	0.240	1.205
Kumar2024	DCNN	3.683	142.182	8.640	3.670	1.806	14.116
Kumar2024	SCNN	4.504	141.263	6.899	2.914	1.443	11.256
Kumar2024	TSBLR	5.796	256.873	2.556	2.818	0.768	6.143
Kumar2024	TSGP	1.974	316.530	8.213	10.742	2.376	21.331
Kumar2024	TSLR	24.104	69.999	0.530	0.171	0.171	0.872
Kumar2024	TSSVM	8.338	70.146	1.485	0.489	0.488	2.461

Table F.72: The average power and energy usage across k -fold cross-validation.

Power & Energy Usage: Lee2019_MI							
Dataset	Pipeline	CPU (W)	GPU (W)	CPU (Wh)	GPU (Wh)	RAM (Wh)	Total (Wh)
Lee2019_MI	BDCNN	1.389	215.622	18.132	12.993	4.216	35.341
Lee2019_MI	BSCNN	0.895	440.525	23.654	37.706	5.996	67.355
Lee2019_MI	CSPBLDA	0.289	82.398	43.293	16.806	14.278	74.378
Lee2019_MI	CSPGP	0.567	387.010	29.117	46.327	8.382	83.826
Lee2019_MI	CSPLDA	21.493	69.700	0.577	0.187	0.188	0.952
Lee2019_MI	CSPSVM	9.390	69.286	1.307	0.425	0.429	2.162
Lee2019_MI	DCNN	2.328	132.665	13.340	5.294	2.793	21.428
Lee2019_MI	SCNN	3.258	143.074	9.861	4.153	2.032	16.046
Lee2019_MI	TSBLR	3.262	337.636	4.698	6.735	1.396	12.829
Lee2019_MI	TSGP	1.930	294.984	9.314	10.787	2.560	22.661
Lee2019_MI	TSLR	14.358	69.965	0.973	0.296	0.296	1.566
Lee2019_MI	TSSVM	1.223	69.539	10.155	3.335	3.357	16.848

Table F.73: The average power and energy usage across k -fold cross-validation.

Power & Energy Usage: Liu2024							
Dataset	Pipeline	CPU (W)	GPU (W)	CPU (Wh)	GPU (Wh)	RAM (Wh)	Total (Wh)
Liu2024	BDCNN	3.010	238.066	6.111	5.657	1.662	13.430
Liu2024	BSCNN	2.047	309.219	7.999	10.168	2.302	20.469
Liu2024	CSPBLDA	1.132	152.532	11.705	8.165	3.747	23.617
Liu2024	CSPGP	1.569	320.567	10.636	14.020	3.066	27.723
Liu2024	CSPLDA	30.462	69.706	0.406	0.131	0.132	0.669
Liu2024	CSPSVM	26.853	69.592	0.433	0.140	0.141	0.714
Liu2024	DCNN	9.526	123.906	2.774	1.116	0.629	4.520
Liu2024	SCNN	13.322	119.755	1.727	0.713	0.416	2.856
Liu2024	TSBLR	6.903	247.757	2.119	2.277	0.643	5.040
Liu2024	TSGP	2.262	313.533	6.918	9.151	2.047	18.116
Liu2024	TSLR	30.376	69.844	0.414	0.134	0.135	0.683
Liu2024	TSSVM	27.072	69.702	0.468	0.150	0.151	0.769

Table F.74: The average power and energy usage across k -fold cross-validation.

Power & Energy Usage: PhysionetMI							
Dataset	Pipeline	CPU (W)	GPU (W)	CPU (Wh)	GPU (Wh)	RAM (Wh)	Total (Wh)
PhysionetMI	BDCNN	0.729	448.484	23.215	42.173	6.582	71.970
PhysionetMI	BSCNN	1.420	371.010	13.806	19.255	3.633	36.694
PhysionetMI	CSPBLDA	0.589	99.838	21.475	10.044	7.042	38.560
PhysionetMI	CSPGP	0.797	358.414	21.262	31.071	6.101	58.434
PhysionetMI	CSPLDA	27.090	69.845	0.468	0.151	0.151	0.770
PhysionetMI	CSPSVM	19.548	69.370	0.621	0.200	0.202	1.024
PhysionetMI	DCNN	5.275	132.726	5.283	2.206	1.164	8.652
PhysionetMI	SCNN	4.402	215.066	6.249	4.270	1.389	11.908
PhysionetMI	TSBLR	3.922	318.096	3.847	5.227	1.150	10.224
PhysionetMI	TSGP	2.108	308.571	7.942	9.961	2.260	20.162
PhysionetMI	TSLR	20.822	69.085	0.802	0.222	0.225	1.248
PhysionetMI	TSSVM	3.303	69.503	3.721	1.226	1.235	6.183

Table F.75: The average power and energy usage across k -fold cross-validation.

Power & Energy Usage: Schirrmeister2017							
Dataset	Pipeline	CPU (W)	GPU (W)	CPU (Wh)	GPU (Wh)	RAM (Wh)	Total (Wh)
Schirrmeister2017	BDCNN	1.846	193.085	12.589	8.411	3.050	24.050
Schirrmeister2017	BSCNN	1.279	371.985	15.266	21.406	4.026	40.698
Schirrmeister2017	CSPBLDA	0.504	88.743	24.924	10.399	8.205	43.528
Schirrmeister2017	CSPGP	0.785	363.393	21.797	32.584	6.242	60.623
Schirrmeister2017	CSPLDA	23.151	69.722	0.542	0.174	0.175	0.892
Schirrmeister2017	CSPSVM	16.471	69.773	0.741	0.242	0.242	1.225
Schirrmeister2017	DCNN	3.127	130.319	8.810	3.644	1.957	14.411
Schirrmeister2017	SCNN	4.677	143.804	6.233	2.762	1.345	10.340
Schirrmeister2017	TSBLR	2.781	373.934	5.362	8.623	1.614	15.600
Schirrmeister2017	TSGP	1.780	267.584	10.549	10.847	2.838	24.233
Schirrmeister2017	TSLR	12.384	69.612	1.431	0.390	0.393	2.214
Schirrmeister2017	TSSVM	0.980	69.532	12.591	4.172	4.200	20.963

Table F.76: The average power and energy usage across k -fold cross-validation.

Power & Energy Usage: Shin2017A							
Dataset	Pipeline	CPU (W)	GPU (W)	CPU (Wh)	GPU (Wh)	RAM (Wh)	Total (Wh)
Shin2017A	BDCNN	1.802	264.120	8.819	9.722	2.577	21.119
Shin2017A	BSCNN	1.368	249.186	10.930	11.723	3.293	25.946
Shin2017A	CSPBLDA	1.459	148.300	9.037	6.141	2.898	18.077
Shin2017A	CSPGP	1.192	330.243	13.561	18.650	3.946	36.157
Shin2017A	CSPLDA	30.685	70.103	0.400	0.131	0.130	0.661
Shin2017A	CSPSVM	29.868	70.003	0.430	0.136	0.136	0.703
Shin2017A	DCNN	9.481	161.372	2.350	1.326	0.575	4.251
Shin2017A	SCNN	11.841	139.970	1.900	0.920	0.461	3.281
Shin2017A	TSBLR	6.868	246.610	2.121	2.267	0.643	5.031
Shin2017A	TSGP	1.976	324.062	7.955	10.823	2.338	21.116
Shin2017A	TSLR	30.785	69.662	0.408	0.132	0.133	0.673
Shin2017A	TSSVM	26.914	69.288	0.441	0.142	0.143	0.727

Table F.77: The average power and energy usage across k -fold cross-validation.

Power & Energy Usage Stieger2021							
Dataset	Pipeline	CPU (W)	GPU (W)	CPU (Wh)	GPU (Wh)	RAM (Wh)	Total (Wh)
Stieger2021	BDCNN	0.137	579.206	131.568	299.016	36.138	466.722
Stieger2021	BSCNN	0.194	498.965	123.023	209.685	29.416	362.123
Stieger2021	CSPBLDA	0.011	76.648	25.960	61.926	56.557	144.443
Stieger2021	CSPGP	0.252	430.100	64.147	114.580	18.643	197.370
Stieger2021	CSPLDA	10.594	69.946	1.154	0.381	0.381	1.916
Stieger2021	CSPSVM	0.785	69.579	15.804	5.200	5.232	26.236
Stieger2021	DCNN	0.684	150.611	48.059	21.063	9.795	78.916
Stieger2021	SCNN	0.418	239.221	75.352	53.564	15.674	144.590
Stieger2021	TSBLR	1.563	425.095	10.135	18.018	2.967	31.120
Stieger2021	TSGP	0.784	189.923	28.134	18.945	6.989	54.068
Stieger2021	TSLR	5.577	69.431	2.384	0.753	0.760	3.897
Stieger2021	TSSVM	0.000	69.262	93.285	595.803	602.152	1291.240

Table F.78: The average power and energy usage across k -fold cross-validation.

Power & Energy Usage Weibo2014							
Dataset	Pipeline	CPU (W)	GPU (W)	CPU (Wh)	GPU (Wh)	RAM (Wh)	Total (Wh)
Weibo2014	BDCNN	3.479	217.369	5.227	4.430	1.427	11.084
Weibo2014	BSCNN	2.545	279.618	6.422	7.382	1.848	15.652
Weibo2014	CSPBLDA	1.503	159.715	8.812	6.442	2.820	18.074
Weibo2014	CSPGP	1.417	326.753	11.480	15.548	3.335	30.363
Weibo2014	CSPLDA	30.136	70.653	0.409	0.133	0.132	0.675
Weibo2014	CSPSVM	30.214	69.816	0.418	0.134	0.135	0.687
Weibo2014	DCNN	10.469	121.174	2.261	0.931	0.538	3.730
Weibo2014	SCNN	9.342	196.663	2.190	1.575	0.560	4.326
Weibo2014	TSBLR	5.489	259.694	2.633	2.989	0.805	6.428
Weibo2014	TSGP	2.168	316.328	7.287	9.655	2.137	19.079
Weibo2014	TSLR	26.795	69.983	0.450	0.147	0.147	0.745
Weibo2014	TSSVM	17.008	69.159	0.707	0.230	0.233	1.170

Table F.79: The average power and energy usage across k -fold cross-validation.

Power & Energy Usage Yang2025							
Dataset	Pipeline	CPU (W)	GPU (W)	CPU (Wh)	GPU (Wh)	RAM (Wh)	Total (Wh)
Yang2025	BDCNN	0.514	203.687	56.451	35.626	12.234	104.310
Yang2025	BSCNN	0.288	651.259	69.057	168.282	18.086	255.425
Yang2025	CSPBLDA	0.103	77.018	120.479	43.834	39.839	204.151
Yang2025	CSPGP	0.380	392.448	42.142	69.352	12.326	123.820
Yang2025	CSPLDA	13.211	70.283	0.936	0.310	0.308	1.554
Yang2025	CSPSVM	1.443	69.830	8.538	2.828	2.834	14.200
Yang2025	DCNN	0.673	135.585	48.320	19.145	9.885	77.351
Yang2025	SCNN	1.484	145.409	23.398	9.804	4.719	37.921
Yang2025	TSBLR	2.004	407.586	7.884	13.442	2.309	23.634
Yang2025	TSGP	1.132	228.878	18.908	15.567	4.769	39.245
Yang2025	TSLR	7.617	70.369	1.738	0.556	0.553	2.846
Yang2025	TSSVM	0.009	69.510	64.320	98.614	99.310	262.244

Table F.80: The average power and energy usage across k -fold cross-validation.

Power & Energy Usage Zhou2020							
Dataset	Pipeline	CPU (W)	GPU (W)	CPU (Wh)	GPU (Wh)	RAM (Wh)	Total (Wh)
Zhou2020	BDCNN	1.359	321.516	15.469	18.067	3.932	37.468
Zhou2020	BSCNN	0.959	421.875	19.807	31.933	5.298	57.038
Zhou2020	CSPBLDA	0.464	87.703	27.183	11.188	8.930	47.301
Zhou2020	CSPGP	0.790	361.079	22.182	33.051	6.337	61.570
Zhou2020	CSPLDA	23.957	69.307	0.509	0.164	0.165	0.838
Zhou2020	CSPSVM	15.424	69.350	0.786	0.255	0.257	1.298
Zhou2020	DCNN	3.542	133.888	8.132	3.383	1.769	13.283
Zhou2020	SCNN	2.766	239.338	8.855	7.124	2.084	18.064
Zhou2020	TSBLR	3.537	326.638	4.292	5.970	1.279	11.542
Zhou2020	TSGP	2.027	302.924	8.462	10.292	2.378	21.132
Zhou2020	TSLR	17.591	68.983	0.904	0.256	0.259	1.419
Zhou2020	TSSVM	2.638	70.064	4.688	1.550	1.549	7.788