Modeling the Feasibility of Nanopore-Based Protein Identification in the Human Proteome

Melissa Sofia Queen

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

Doctor of Philosophy

University of Washington

2025

Reading Committee:

Jeffrey Nivala, Chair

Luis H Ceze

Hari Sadasivan

Program Authorized to Offer Degree:
Paul G. Allen School of Computer Science and Engineering

University of Washington

Abstract

Modeling the Feasibility of Nanopore-Based Protein Identification in the Human Proteome

Melissa Sofia Queen

Chair of the Supervisory Committee:
Jeffrey Nivala
Paul G. Allen School of Computer Science and Engineering

The gap between what can be read from the genome and what is functionally realized at the protein level remains a central challenge in molecular biology. Because proteins are shaped by splicing, translation, and post-translational modifications, and often exist as diverse full-length proteoforms, proteomics demands new approaches beyond conventional mass spectrometry.

This thesis explores the feasibility of identifying gene-encoded human proteins directly from nanopore-generated electrical signals ("squiggles"). Using AminoScribe, a simulation framework grounded in empirical nanopore data generated via an unfoldase-based translocation approach, simulated squiggles were produced from full-length human protein sequences. In this method, proteins are pulled through the nanopore by a molecular motor, enabling continuous signal acquisition along the entire length of each molecule. Each simulated signal was aligned to a reference library of canonical squiggle patterns using dynamic time warping (DTW), allowing for comparisons invariant to temporal distortions. Classification systems were evaluated using fifty simulated versions per protein to assess proteome-wide coverage.

Results indicate that human protein squiggles contain distinct, classifiable signal features. A protein was considered "covered" if correctly identified in at least one

of ten trials, and "robustly covered" if correctly identified in all ten. Under ideal simulation conditions, a nearest-neighbor classifier achieved nearly 100% coverage and 92% robust coverage. In more realistic, high-noise conditions, overall coverage remained high (99%), but robust coverage dropped to 10%. A multilayer perceptron (MLP) classifier improved performance in these conditions, increasing robust coverage to 33% using a reduced feature set.

These findings establish a baseline for protein identification from nanopore squiggles and outline both the potential and current limitations of nanopore-based proteomics under simulated noise models. Future work will explore the extension of this approach to detecting post-translational modifications and other forms of protein-level variation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

This work was made possible by the support of many people and communities.

First, my research group, the Molecular Information Systems Lab (MISL), and especially the nanopore protein research team. Science is collaborative by nature, and everything here builds on the creativity and effort of the entire group.

I owe deep thanks to the women who welcomed me into graduate school and guided me through my first projects: Lee Organick, Katie Doroshak, and Callie Bee. You are phenomenal researchers and (unofficial) mentors, and your example shaped my path more than you know.

I am also grateful to my advisors and committee members: Luis Ceze, Jeff Nivala, Hari Sadasivan, and Yoshi Kohno.

Luis—your presentation of a DNA-based implementation of Simulation Search during the prospective student visit day deeply inspired me and, with real joy, set me on the research path that became the foundation of this dissertation. Your support during my first projects and throughout my early years in graduate school prepared me for success, and your continued enthusiasm and endorsement have been profoundly meaningful.

Yoshi—though we never worked on a project directly, I learned a great deal from following the CyBio group and exploring the intersection of molecular computing and data security, and I have always appreciated your enthusiasm as a committee member.

Hari—the later stages of this dissertation hinged on GPU development, and I could not have navigated the challenges and rewards of that work without your guidance.

Jeff—you welcomed me back into research after medical leave and helped me

transition from DNA to proteins, from data storage to nanopore sensing. I left every meeting calmer, more inspired, and more confident. You have always believed in this project, which I take as an honor, but more importantly you have always supported me as a whole person, through dead ends, rabbit holes, crises of confidence, health problems, delays, and the messiness of being human while doing research. I could not have finished without you.

The proteome-scale simulations in this dissertation were made possible by computational resources and support provided by Advanced Micro Devices, Inc., under the AMD AI & HPC Cluster Program. I am especially grateful to Gina Satariman, Gagan Singh, and the rest of the AMD team who supported me in developing, running, and profiling my software.

My PhD was also supported in part by the Department of Energy's Computational Science Graduate Fellowship. This fellowship not only funded my work but also connected me to a remarkable community of scholars, and I am honored to count myself among its alumni.

Finally, graduate school is just one part of life (though at times it felt overwhelming) and I am grateful to my family and friends for grounding me. There were moments when the finish line seemed impossibly far, but knowing that *you* believed I could reach it often made all the difference.

Chapter 1
INTRODUCTION

DNA, the code of life, is present in every cell of an organism. However, it can only offer a partial glimpse into the intricate and ever-changing landscape of a cell. When a gene encoded in DNA is activated, it is first transcribed into messenger RNA (mRNA), which is in turn translated into a chain of amino acids, forming a protein. Proteins have myriad functions: they make up the structure of a cell, they digest food, they defend against pathogens, and much more. They are the actors that have taken the script, the genome, and have brought it to life on the stage.

DNA sequencing has become a well-established practice, with next-generation sequencing capable of generating over a billion sequencing reads per run. This has unlocked an unprecedented view of the genome—the complete collection of genes that a cell can express. It is the proteome—the profile of proteins present in a cell—that truly reflects cellular state and processes. We can sequence the mRNA present in a cell to understand what genes are currently active, but this alone cannot tell us what proteins are actually present. The translation of mRNAs into proteins can happen at different rates, and those proteins can have varying lifespans. Moreover, the proteins themselves can be modified after they have been created, both enzymatically (e.g., phosphorylation) and spontaneously (e.g., deamination) [33].

Proteome snapshots hold the promise of unlocking new insights into cell states and functions. This understanding is an invaluable tool for not only scientific research, but also in clinical applications and diagnostics. Many cancers can be diagnosed and monitored through protein biomarkers, such as the Human Epidermal Growth Factor Receptor 2 (HER2/neu) in breast cancer [18, 20], CA-125 in ovarian cancer [40],

and CA 19-9 in pancreatic cancer [2]. In the case of Alzheimer's disease—infamously difficult to detect early—the presence of Amyloid-beta and Tau proteins, especially the ratio of total to phosphorylated Tau, are key markers for tracking disease progression [34].

The protein-coding information of a single gene can be rearranged and spliced together in mRNA form, resulting in a variety of protein sequences, each with varying function. These spliced proteins are called spliceoforms, and it has been proposed that the more than 20,000 genes in the human genome may give rise to millions of distinct proteins [33]. This highlights the importance of being able to directly measure proteins, and distinguish between spliceoforms. For example, the protein Bcl-2 has a number of different spliceoforms, the ratios of which control cell death [14]. If we could reliably distinguish between Bcl-2 spliceoforms, we would be able to predict the fate of cells.

Protein sequencing technologies have lagged significantly behind their DNA sequencing counterparts. There are several factors contributing to this gap. DNA is constructed from four nucleic acids, while proteins are made of a much more complex alphabet of 20 different amino acids. DNA has a well-behaved double-helix structure with predictable base pairings, while proteins form complex secondary and tertiary structures that are difficult to predict and manipulate. Moreover, a key part of many DNA sequencing techniques involves amplification, allowing a small sample of DNA to be duplicated into a much larger sample—a capability not currently available for proteins. This makes detecting proteins that are present in trace amounts a formidable challenge. Overcoming these challenges in protein sensing would mark a significant step forward in our ability to probe the inner workings of cells.

Chapter 2

BACKGROUND

2.1 Current Protein Identification Technologies

Mass spectrometry (MS), first devised in the early 1900s to analyze atomic and isotopic composition, has undergone decades of refinement to become the state-of-the-art method for protein identification [19]. To identify proteins, the MS protocol enzymatically digests a sample into constituent peptides, ionizes them, and transmits them through a mass analyzer [1]. The mass analyzer precisely measures each peptide's mass-to-charge ratio. By matching these measurements to a reference database, peptides can be identified and, in turn, used to infer the identity of the original protein [1].

While MS is a powerful and indispensable technique, it is not without drawbacks. MS instruments are costly and complex, and need expert operators. The ionization and fragmentation process is inherently destructive [10]: proteins in the sample cannot be used for further analysis. MS also typically needs a substantial pool of proteins for bulk processing. Consequently, proteins present in trace amounts can be drowned out by more dominant signals [1]. Additionally, when proteins are digested into constituent peptides we lose information about the complete functional form of the protein. For example, splice isoforms may yield the same constituent peptides, and thus be indistinguishable from one another in the MS framework.

Mass spectrometry methodologies have significantly advanced in the quest for single-cell proteomics. In 2016, a study assessing MS capabilities found that only 10% of single nucleotide variants observed at the DNA or RNA level were detectable by MS, and even fewer proteoforms were measurable [26]. By 2020, Kelly et al.

[15] used MS to detect approximately 1,000 distinct proteins within a single cell—an impressive improvement over conventional MS, though still far from capturing the complete proteome. Despite these advances, increased sensitivity often comes at the cost of additional time and resources devoted to sample preparation.

As the scientific community has continued to refine MS protocols, it has also been exploring alternate strategies for protein sensing. Two such technologies are fluorosequencing [35] and FRET X [8]. Swaminathan et al. [35] propose a fluorosequencing method, which labels peptides with fluorescent markers and observes fluorescence changes as amino acids are iteratively cleaved using Edman degradation. This approach allows for parallel sequencing by anchoring peptides on a plate and observing the fluorescence changes at each peptide location. Not all amino acids are labeled, which prevents explicit identification; however, knowing the positions of a subset of amino acids can suffice to match sequences to a human protein database. Their Monte Carlo simulations demonstrate robust protein identification coverage, albeit sensitive to certain errors like photobleaching. The method achieves proteome coverage comparable to mass spectrometry with the advantages of single-molecule sensitivity and parallelization.

Similarly, de Lannoy et al. [8] introduced FRET X, a single-molecule fingerprinting technique that uses Förster Resonance Energy Transfer (FRET) to measure distances between labeled amino acids within proteins while they are in their native, folded structures. Through both simulations and experimental work, they showed they were able to differentiate spliceoforms and identify proteins using a reference database with high accuracy, even when simulating errors.

Despite these advances, current protein identification technologies retain significant limitations. Mass spectrometry is inherently destructive, requires bulk samples, and cannot detect proteins present in very low abundance. Alternative single-molecule methods, while promising, remain under development and struggle with incomplete labeling. It remains difficult to resolve spliceoforms and post-translational modifica-

tions. In general, many techniques cannot interrogate proteins in their native states or detect molecules that fail to ionize or fluoresce. These persistent challenges highlight the need for approaches like nanopore sensing that can quantitatively probe individual, full-length proteins.

2.2 Nanopore Sensing Technology

Nanopore sensors operate by analyzing the characteristic changes in current as molecules pass through nanoscale pores. The resulting current trace (often referred to as a squiggle) can be analyzed directly using signal processing algorithms, and in the case of DNA and RNA molecules it can be basecalled into a sequence [36]. A key benefit of nanopore sequencing strategies is the ability to read an entire, full-length molecule [23][21]. Unlike the many other strategies for both DNA and protein sequencing that involve fragmenting molecules into constituent pieces, nanopore sequencing allows for the analysis of unmodified and complete molecules. These full-length reads allow us to distinguish between the spliceoforms that are otherwise occluded when proteins are fragmented.

Nanopore sensors are inherently quantitative: each squiggle represents a single observed molecule [6, 37]. This allows for the observation of minute amounts of proteins that are otherwise undetectable or drowned out in bulk analysis methods. In the absence of a protein amplification method, this single molecule detection strategy is particularly compelling. Previous research with nanopores has shown that they can detect post-translational changes [25, 38, 23], and discriminate between single amino acid changes in a predictable way [21]. The unique squiggles generated by nanopores hold the potential to serve as distinct fingerprints for database lookup strategies, as well as the potential for *de novo* sequencing.

Chapter 3

RESEARCH QUESTION AND OBJECTIVES

Our project investigates whether nanopore-generated squiggles are distinct enough to detect and identify human proteins. In general, sequencing determines the complete order of a molecule's building blocks, while sensing identifies a molecule by comparing measured features to a reference database without reconstructing the full sequence. Rather than directly predicting protein sequences from these squiggles, we use a sensing, or fingerprinting, approach where squiggles are matched against a reference database.

To assess the viability of this approach, we simulate realistic nanopore squiggles for each known protein in the human proteome, incorporating stochastic noise to mimic experimental variability. We then match these squiggles against a canonical reference database. By performing Monte Carlo simulations and calculating precision, recall, and F1 scores, we can quantify how reliably we can expect to identify human proteins using only their nanopore squiggle. Quantifying identification performance provides a reference point to evaluate whether nanopore sensing can complement or surpass current approaches like mass spectrometry and fluorosequencing, establishing a benchmark for using nanopores as a novel tool for human proteomics.

Chapter 4

METHODS

4.1 Proteome Coverage Estimation Strategy

To estimate proteome coverage we simulate nanopore squiggles for each gene-encoded human protein and attempt to identify them by comparing signals against canonical reference database. The reference squiggles are generated using a deterministic, non-stochastic baseline model. Two types of coverage are defined: *any coverage*, the fraction of proteins identified at least once across ten trials, and *robust coverage*, the fraction identified in all ten trials. These initial simulations were restricted to gene-encoded protein sequences, excluding proteoforms and post-translational modifications.

4.2 Core Tools

4.2.1 AminoScribe: Digital Twin for Nanopore Protein Squiggles

Motivation

Several nanopore signal simulators have been developed to test, improve, and explore nanopore sequencing. These simulators focus exclusively on modeling DNA squiggles, as this is the current use case for nanopores, and no such models exist for simulating protein squiggles. NanoSim generates nanopore sequencing data based on statistical modeling. It does not simulate the raw signal data directly, but rather simulates the downstream sequencing results [39]. DeepSimulator, another major tool, uses deep learning techniques to produce realistic signal data, although it tends to be computationally intensive and produces some systematic errors affecting downstream

analysis [16].

Squigulator [11] is a model that generates realistic DNA sequencing squiggles. Rather than taking a deep learning approach, they build up their model explicitly applying the behavior and noise observed in experimental nanopore data. This approach is more interpretable than deep learning, and significantly less computationally intensive. Squigulator has a modular design that allows users to define their own pore model, and also models signal noise. This allows the model to be used to explore optimal conditions for basecalling, by adjusting parameters and using Monte Carlo simulations to assess their impact on basecalling accuracy.

To run proteome-scale simulations we need an in silico model capable of generating millions of different realistic protein squiggles. Existing nanopore simulators, like like NanoSim [39], DeepSimulator [16], and Squigulator [11], focus exclusively on DNA squiggles and are not designed for protein simulations. This motivated the development of a novel nanopore protein signal simulator called AminoScribe.

AminoScribe is built using the experimental data collected by Motone et al. [21] and Cardozo et al. [5]. These experiments informed a base model that generates smooth, deterministic squiggles by convolving a 20-amino-acid reading window across a protein sequence. Stochastic noise is applied on top of this base model to generate more realistic squiggles.

Proteins don't move through the nanopore at a fixed speed—-translocation is controlled by a motor protein with distinct and varying stepping behavior. To mimic these variations in translocation speed we take dynamic time warping (DTW) alignments between real experimental squiggles and the base model to learn stepping behavior, which was then reverse engineered and incorporated into the model.

The experimental squiggles also have high frequency background noise, which is typically smoothed with a low pass filter. AminoScribe uses the high frequency noise residuals from these experimental squiggles to apply noise to simulated squiggles. These squiggles are reproducible when using a seeded random number generator

(configurable), and the model safe for multiprocessing.

Usage and Licensing

AminoScribe takes a protein sequence or UniProt ID and uses configurable noise levels, optional tags, and post-processing options (e.g., normalization, filtering). It outputs a reproducible, stochastic simulated squiggle. AminoScribe source code is publicly available at `https://github.com/uwmisl/Amino-Scribe` under the Apache License, Version 2.0. It is available on the Python Package Index (PyPI) at `https://pypi.org/project/aminoscribe` and can be imported into Python environments: `pip install aminoscribe`. Full implementation details and model verification results are provided as a Supplement (Chapter A).

4.2.2 DTWhiz: GPU-Accelerated Global Signal Alignment

Motivation

Dynamic time warping (DTW) is a signal processing algorithm that can compare two signals with inconsistent time domain cadences. DTW is an invaluable tool in processing nanopore signals, since the translocating molecules have variable speeds. When comparing squiggles we don't want to penalize any speed-ups or stalls—the signal comparison must allow for insertions and deletions of signal values. DTW gives us a principled way to compare these signals, and returns a score that represents the similarity of two signals, with a lower score indicating the two signals are more similar in shape. Algorithm and implementation details are provided as a Supplement (Chapter B).

A full-depth Monte Carlo simulated test of SquiggleMatch requires over 10 billion DTW alignments. CPU-based DTW implementations are intractably slow at this scale—we benchmarked an optimized C implementation (dtaidistance[9]) on a dual-socket AMD EPYC system with 128 physical CPU cores and found a full simulation

would take 7 weeks to complete.

DTWhiz is able to run a full simulation on a single AMD Instinct™ MI210 Accelerator in total runtime of around 25 days, which can be distributed across compute nodes and makes our initial simulations and well as future iterations more computationally tractable.

We benchmarked DTWhiz against `dtaidistance` to quantify both end-to-end runtime and raw kernel execution time. The kernel refers to the core GPU computation of the DTW matrix itself, which executes in under 0.20 seconds per workload. However, repeated GPU setup calls (`hipSetDevice`) dominate the overall runtime, bringing the end-to-end figure closer to 2.2 seconds. Reporting both runtimes clarifies performance: the kernel-only runtime defines the theoretical lower bound, while the end-to-end runtime reflects the cost of the current usage model. This distinction demonstrates the efficiency of the core implementation and points to an easy opportunity for future optimization, such as batching workloads to amortize remaining overhead.

Usage and Licensing

DTWhiz takes as input a query squiggle and a set of reference squiggles. It outputs a vector of DTW alignment scores. DTWhiz source code is publicly available at `https://github.com/uwmisl/dtwhiz` under the Apache License, Version 2.0. Implementation and benchmarking details are available as a Supplement (Chapter B).

Method	Per Protein ID Time	Full Simulation Time
fastdtw (single core)	35 minutes	66 years
dtw-python (1 core)	24 minutes	46 years
dtw-python (128 cores)	27 seconds	11 weeks
dtaidistance	5 seconds	7 weeks
DTWhiz (end-to-end)	2.2 seconds	25 days
DTWhiz (kernel execution)	0.2 seconds	2.2 days

4.3 Simulation Workflow

4.3.1 SquiggleMatch Engine

The SquiggleMatch engine uses AminoScribe to generate batches of stochastic protein squiggles. It then uses DTWhiz to execute the protein identification step, where each squiggle is compared using DTW alignment to all squiggles in a canonical reference set. SquiggleMatch takes as input a UniProt protein ID and a desired simulation depth, and returns a vector of DTW alignment scores against the canonical reference set.

4.3.2 Monte Carlo Framework

Overview: The Monte Carlo simulation uses the SquiggleMatch engine to classify a batch of 50 squiggles for each gene-encoded human protein (this currently excludes proteoforms and post-translational modifications). This gives over a million simulated protein identifications. The simulation runs in configurable noise regimes, determining the level of noise applied to simulated protein squiggles. Metadata, like the random seed used by AminoScribe, squiggle length, and normalization factors are recorded along with final results. All data was saved in Apache Parquet format, partitioned by protein id. This partitioning gives every protein id a separate directory, and since each protein id is processed individually with separate SquiggleMatch processes, the

results can safely be written in parallel. Partitioning also lets downstream analysis avoid pulling all data into memory by streaming data with pyarrow or by choosing specific protein records to investigate.

The full simulation ran on the AMD AI & HPC Cluster [3], utilizing nodes with a dual-socket AMD EPYC system with 128 physical CPU cores and four AMD Instinct™ MI210 Accelerators (MI210).

Further details about the simulation pipeline, implementation details, and scaling strategies are provided as a Supplement (Chapter C).

We ran two full simulations of proteome coverage, one with minimal modeled noise and one that included all modeled sources of stochasticity.

In the low noise regime, AminoScribe generates baseline squiggles without time warping or amplitude noise. The only stochasticity, and the only difference from the canonical squiggles, is the inclusion of variance in the signal impact at the amino acid level. This configuration tests the best case scenario for squiggle identification by including only intrinsic signal fluctuations. This gives an upper bound on proteome coverage.

In the high noise regime all of AminoScribe's stochastic features were enabled: time warping, amplitude noise, and dynamic z-score normalization. The resulting squiggles are processed with a low-pass filter and downsampled by a factor of 20 to mimic the expected data preparation pipeline for experimental squiggles. This regime captures proteome coverage under the full set of variability modeled to date, giving the most comprehensive approximation currently available.

4.4 Analysis Methodology

4.5 Avoiding Direct Squiggle to Protein ID Classification

There have been a number of successful approaches to classifying nanopore squiggles using raw signal data [4, 22, 24, 29, 22, 31, 17]. However, our early tests showed that

a network trained directly on simulated squiggles could memorize simulation artifacts and achieve unrealistically high accuracy compared to the squiggles coming out of the wet lab. AminoScribe is tuned to mimic the DTW alignment behavior of experimental squiggles. By converting every squiggle into DTW alignment scores against canonical references we obtain features that better reflect behavior of experimental squiggles. Therefore, rather than learning directly from raw simulated signals, we strictly use DTW scores as feature vectors to give a more realistic approximation of experimental performance. As we collect and process more experimental protein squiggle data, the AminoScribe model will improve and in the future we expect to be able to develop direct signal-to-label classifiers.

4.6 Nearest Neighbor Evaluation

Nearest Neighbor (NN) classification identifies a protein strictly based on its DTW alignment scores against a reference set. NN evaluations assess whether DTW alignment preserves signal shape under noise. Metrics used include precision, recall and top-k accuracy. Proteome coverage is measured as the fraction of identifiable proteins.

4.6.1 Intermediate Results Aggregation

The full dataset of results is dense, with almost 20 billion different DTW alignment scores. Intermediate processing breaks this down into condensed statistics, which are then saved and can be loaded and reused without having to scan the full dataset again. This pipeline takes advantage of Parquet partitioning to extract relevant information about each protein without needing to load all data into memory. Parquet files can be read and processed in parallel, speeding up the dataset scan. We collected both per-trial and per-label statistics. In this setup, a trial is the attempted identification of a single protein signal. A label is one of the proteins represented in the canonical squiggle set. Per-trial statistics yield true-positive rates and per-label statistics capture false positive rates. These statistics are collated and used to give an overview of

per-protein recall and precision.

4.6.2 Per-Trial Metrics

For each protein We load the partitioned results and collect the 'correct match rank' for each of the 50 trials. This is the ranked position of the true label. We then compute the recall (true positive rate) for each protein, taking the fraction of trials for which the true match was ranked first.

4.6.3 Per-Label Aggregates

To avoid repeated scans of the dataset, the per-protein intermediate processing step also collects a summary of observed false positives. This gives us a false positive rate for each label in the reference database.

We also compute reliability, a heuristic of a label's predictive power. Labels that match rarely or randomly provide little information and have a low reliability score. Labels that regularly show up as matches to the same set of proteins have high reliability.

4.6.4 Combined Metrics

Merging the per-trial and per-label summaries lets us calculate the F1 score of each protein, a metric that balances over- and under-prediction.

4.7 Multilayer Perceptron Classifier

4.7.1 Motivation: Do we need all DTW alignment scores?

We investigated whether classification is possible when using a subset of DTW alignment scores as features, rather than relying only on the top ranking match. Using only a subset of DTW alignment scores would eliminate the need for a complete scan of the proteome reference set. This reduces the number of DTW alignments that need

to be performed and would dramatically reduce the computational cost of protein identifications.

4.7.2 Anchor Selection via Label Reliability

Choosing a set of discriminating anchors from a 20k-dimension space is non-trivial. Instead we start by choosing anchors based on how canonical squiggle labels performed under nearest neighbor classification. Traditional feature selection and dimensionality reduction techniques, like mutual information (MI) or principal component analysis (PCA), break down or become intractable with our dataset of 20k labels and only 50 trials per label. With this many labels and so few relative trials, MI is both computationally intensive (many labels) and less statistically reliable (too few trials per label). PCA is similarly likely to fit noise without enough trials to show clear, distinct protein clusters in the full high dimensional space.

Instead, we choose labels based on the sum of pointwise mutual information calculations.

For each label in the canonical reference set, we record the proteins that were incorrectly assigned that label. For each label L and protein i, we compute

$$\begin{aligned} p(L) &= \text{how often } L \text{ is predicted,} \\ p(i) &= \text{how often } i \text{ is the true label,} \\ p(L,i) &= \text{how often } L \text{ is predicted while } i \text{ is the true label.} \end{aligned}$$

Pointwise mutual information between the label L and a particular protein i:

$$\text{PMI}(L,i) = \log\left(\frac{p(L,i)}{p(L)\cdot p(i)}\right)$$

measures the joint probability between label L and protein i—are they completely independent variables, or does label L correlate with protein i? The pointwise mutual

information is 0 if L matches protein i only by chance, and is > 0 if they occur together more often than just randomly.

For our reliability measure, this is weighted by how often L and i occur together: $p(L, i)$, so that informative but rare pairings do not dominate scores, and then summed over all proteins:

$$\text{Reliability}(L) = \sum_i \left[p(L, i) \cdot \log \left(\frac{p(L, i)}{p(L) \cdot p(i)} \right) \right]$$

This rewards labels that are predicted often, and that have a consistent set of proteins they match to. Labels with high reliability might have many false positives, but even a false positive can help narrow down the pool.

Labels with high reliability are chosen as anchor features. Since a protein's true label may not be among these anchors, classifiers infer protein identity by evaluating the protein's DTW alignment scores relative to the anchor set in feature space.

This is used as a first-pass heuristic for feature selection and reduces the massive dimensionality of the original 19,772 element feature vectors, allowing for easier classification train and testing.

4.7.3 Model Architecture

To see if this feature reduction strategy is effective, we trained a Multilayer Perceptron (MLP) using PyTorch [7] and tested classification accuracy.

The full dataset comprised 19,772 unique protein labels, with 50 trials per label, resulting in nearly one million labeled examples. The dataset was split into training and test sets using an 80/20 stratified split, ensuring that each protein label was proportionally represented in both sets. Since each protein has only 50 trials, the test set contains roughly 10 examples per label; thus, evaluation metrics per protein may be noisy, and results should be interpreted as a proof of concept rather than definitive performance.

Input features to the MLP are DTW alignment scores between the unknown squiggle the selected subset of squiggles in the reference database. Input features were standardized using `StandardScaler` to prevent anchor points with consistently large DTW alignment scores from dominating—for example, if the canonical squiggle is especially long or the signal sits at an uncommonly high or low amplitude range, it would consistently have poor alignments and high alignment scores.

Our two-layer MLP consists of an input layer (1000 or 2589 DTW alignment scores, depending on noise regime), one hidden layer (1024 neurons, ReLU), a dropout layer ($p = 0.2$), and an output layer (19,772 neurons, softmax).

This architecture was selected as a lightweight deep learning baseline, using mostly default hyperparameters (e.g., ReLU activation, dropout rate, and hidden layer size). No systematic hyperparameter tuning was performed at this stage, as the goal was to establish baseline feasibility and assess if a small number of anchor distances can preserve signal space structure.

The model was evaluated with a 20% holdout set, giving us ten simulated squiggle identifications per protein. We collected top-k accuracy results for each protein, and the proteome coverage percent (fraction of proteins with successful identifications).

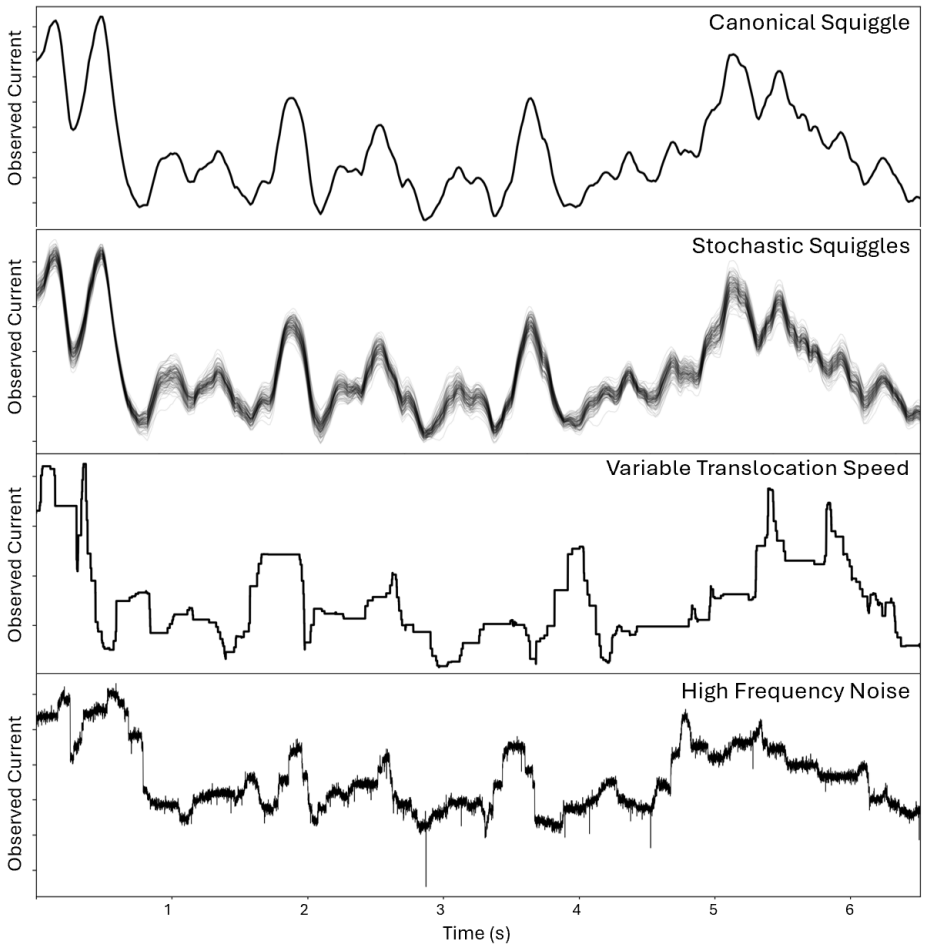

Figure 4.1: AminoScribe can apply different sources of stochastic behavior. On top we see the canonical squiggle shape. In the middle we see a sampling of simulated squiggles when including per-amino-acid variance behavior and an example of modeling varying translocation speeds. The squiggle no longer matches point by point with the canonical shape, highlighting the need for dynamic time warping alignments. On bottom there's an example of a squiggle generated with per-amino-acid variance, variable translocation speed, and amplitude noise.

Chapter 5

RESULTS

5.1 Experiment Overview

The Monte Carlo protein identification simulation was run in two different noise regimes. The low noise regime modeled only inherent amino-acid-level variance in the protein squiggle, leaving the time domain consistent (removing motor protein behavior artifacts) and adding no signal noise. The high noise regime includes all stochastic squiggle behavior modeled thus far: amino-acid-level variance, time warping, signal noise, and post-processing normalization using a C-terminal tag. In each regime we simulated 50 squiggles for each of 19772 human protein sequences. Each simulated squiggle was aligned to all 19772 canonical squiggles, giving a DTW alignment score for each. These alignment scores are used directly by a nearest neighbor classifier.

A subset of canonical squiggles was chosen to act as anchors in signal space, reducing the dimensions from 19,772 to just a few thousand. The alignment scores between an unknown squiggle and these anchors form the feature vectors that are used to train and test a Multilayer Perceptron (MLP) classifier. The MLP classifier was benchmarked with a 20% holdout set of simulated squiggles.

5.2 Classification Statistics

In top-k accuracy, a match is considered accurate as long as the correct label is in the top-k ranked labels. In top-1 accuracy the correct label must be chosen exactly, while top-100 loosens the requirement to being anywhere in the top 100 ranked options (The 99.5% percentile amongst 19,772 labels).

In the low noise regime, nearest neighbor (NN) classification performs near-perfectly:

top-1 accuracy is 97.91%, rising to 99.96% at top-100 (Table 5.1). The MLP classifier, though using only a small fraction of the full feature space (1000 of 19,772 DTW distances), tracks closely—reaching 100% top-100 accuracy. In the more realistic high noise regime, performance drops sharply: NN top-1 accuracy falls to 65.17%. However, the MLP classifier regains much of this ground, achieving 80.05% at top-1 and 98.81% at top-100, again using a reduced feature vector (Figure 5.1).

Classification performance also varies with sequence length. Shorter proteins are harder to identify due to limited distinguishing signal content. In both regimes, accuracy increases with sequence length (Figures 5.4). In the low noise setting, sequences longer than 250 amino acids achieve high reliability. Under high noise, the length threshold for stabilization shifts higher—proteins shorter than about 1000 amino acids exhibit noticeably lower accuracy and greater variance.

Additionally, classification accuracy is strongly affected by signal normalization. Each squiggle is normalized using a common C-terminal tag, but noise and time warping cause variation in the observed mean and standard deviation. As shown in Figure 5.5, correctly matched squiggles cluster in a diagonal band in normalization space, suggesting that some skewed normalizations (e.g., upshifted mean with downscaled variance) are still compatible with accurate identification. This goes against our the assumption that optimal normalization is centrally located, and points to underlying interactions between this normalization technique and DTW-based matching.

Together, these results show the limits of simple NN classification under realistic noise and the potential of feature-space compression and learned models to recover accuracy.

Squiggles are normalized before comparison using identical C-terminal tag sequence. Because the squiggle has noise and timewarping added before performing z-score normalization the detected mean and standard deviation is different between

	Low Noise NN	Low Noise MLP	High Noise NN	High Noise MLP
Top-1	97.91%	96.31%	65.17%	80.05%
Top-5	99.64%	99.61%	73.25%	92.09%
Top-20	99.92%	99.99%	78.92%	96.51%
Top-100	99.96%	100%	85.28%	98.81%

Table 5.1: Top-k Accuracy Comparison Between Nearest Neighbor and MLP Classifiers

trials. The coordinates on this heatmap show the mean and standard deviation detected in C-terminal squiggle, and are colored depending on the rank of the correct match. A dark purple indicates that the best match was ranked very close to the top, while yellow indicates ranking in the middle of the pack, far below any useful cutoff. High ranking squiggles cluster in a diagonal band rather than a concentrated center point.

5.2.1 Feature Selection and Label Reliability

The vector of DTW alignment scores between a query squiggle and canonical squiggles is used as a feature vector for more advanced classifiers. Rather than using all DTW alignment scores, we selected a subset to use as anchor points. An unknown squiggle is DTW aligned to only these anchor squiggles, and the resulting scores help to triangulate its identity. Anchors are chosen based on a label reliability heuristic discussed in Methods. Labels that are highly reliable match a very consistent set of proteins, discriminating between protein clusters and reflecting local structure in the proteome. This is exactly what we want: to encode structure beyond just self-matching.

In the low noise regime, we chose the top 0.5% of labels ranked by reliability as the anchor vectors. In the high noise regime we did the same, while also including proteins with perfect precision. In the low noise regime almost all proteins had perfect precision, but the addition of realistic noise dropped this down to just 2589 proteins.

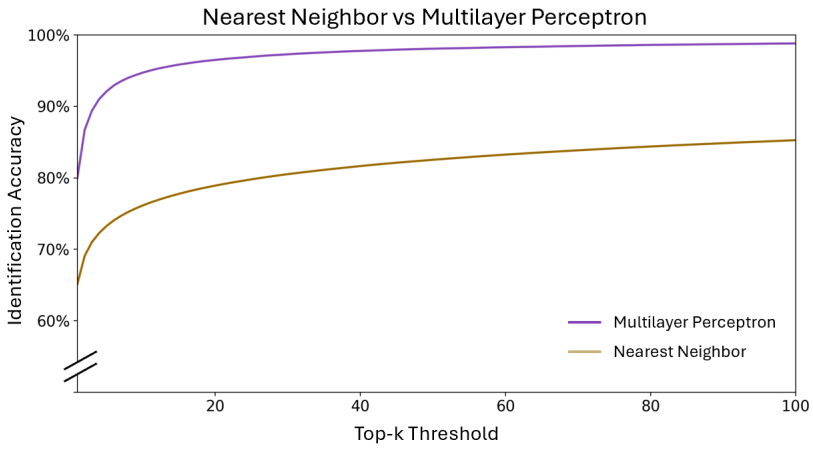

Figure 5.1: This plot shows the probability of recovering the true match within the top k ranked candidates for nearest neighbor (NN) and for the multilayer perceptron (MLP) in the high noise regime. The NN classifier starts with 65% accuracy at k=1, and rises as we loosen the definition of success to include top-20 (79%), but reaches diminishing returns and improves to only 85% when taking top-100. Using the MLP classifier improves accuracy significantly, with 80% of trials successfully identified with just k=1 (comparable to k=20 for NN). The MLP gains accuracy very quickly as we relax to higher k values. With top-5 it has 90% accuracy, growing to 96% accuracy at k=20 before plateauing. This shows that the MLP classifier almost always has the correct answer ranked among the top 20 candidates.

These protein squiggles remained distinct enough with the addition of noise that they are not easily confused with other proteins. We chose to include these proteins to act as distinct outliers—the highly reliable anchors can define clustering locations, while the outliers can take a zoomed out look from a different perspective.

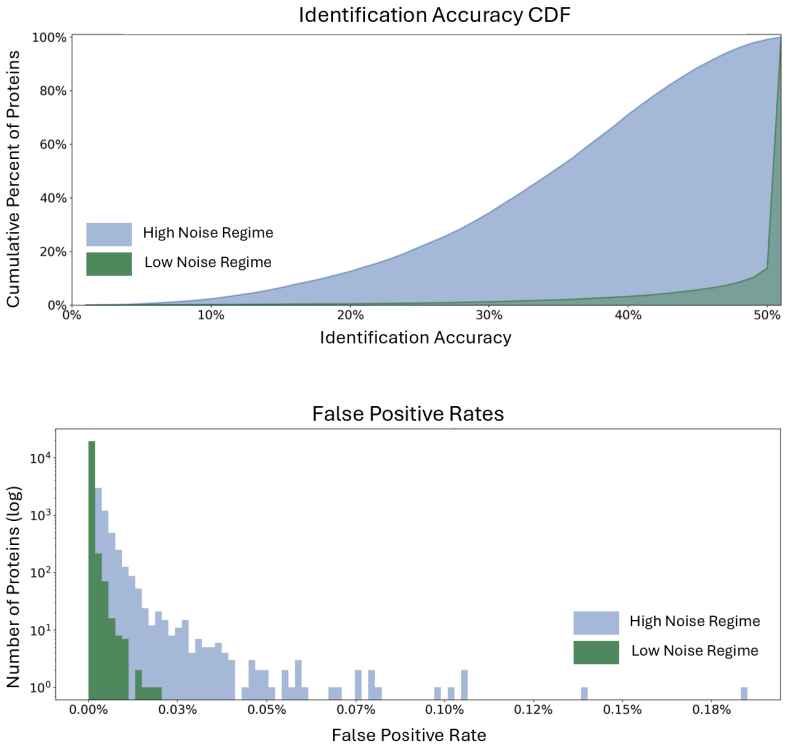

Figure 5.2: The first plot shows the per-protein identification accuracy as a cumulative distribution function (CDF), summarizing 50 trials per protein. In the high noise regime most proteins achieve 70–80% accuracy, indicating that correct matches are ranked first the majority of the time. The second plot shows the distribution of false positive (FP) rates per protein, expressed as a percentage of all trials ($50 \times 19{,}772$). Only a small fraction of proteins contribute disproportionately to false positives: in the canonical set, a few proteins account for the majority of FP events, while most proteins have rates near 0%. The pattern is consistent across noise regimes, although the high-noise regime produces generally higher FP counts. This highlights that FP occurrences are concentrated in a small subset of proteins rather than spread uniformly.

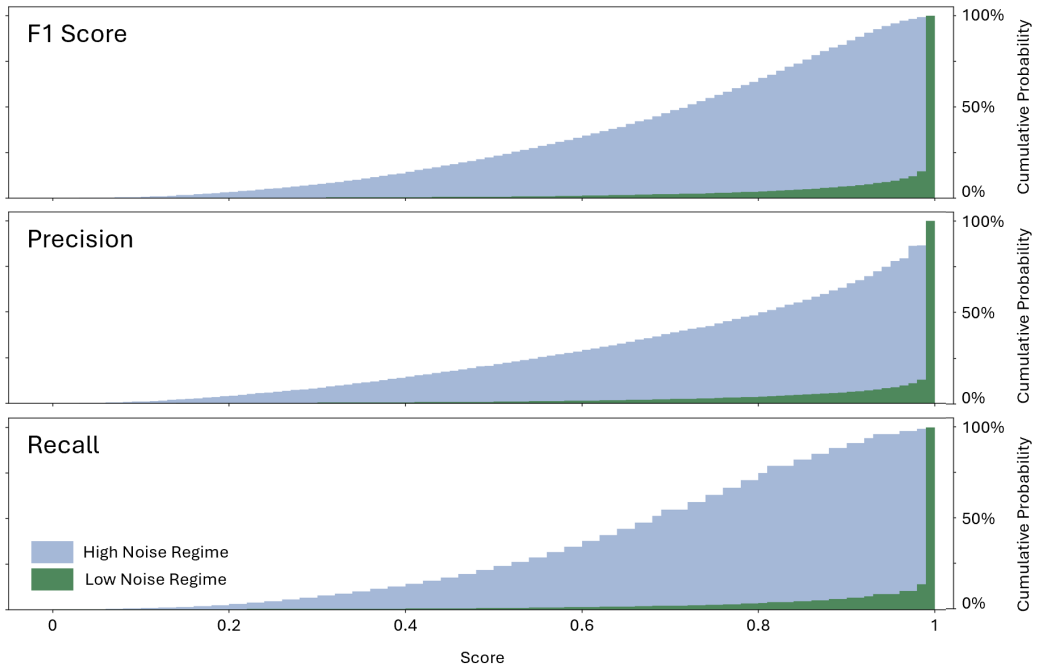

Figure 5.3: In the low noise regime, proteins had very high mean precision (97.95%), recall (97.91%) and resultant F1 score (97.82%). In the high noise regime, protein precision remains fairly high but recall drops significantly, pulling the F1 scores away from 1.

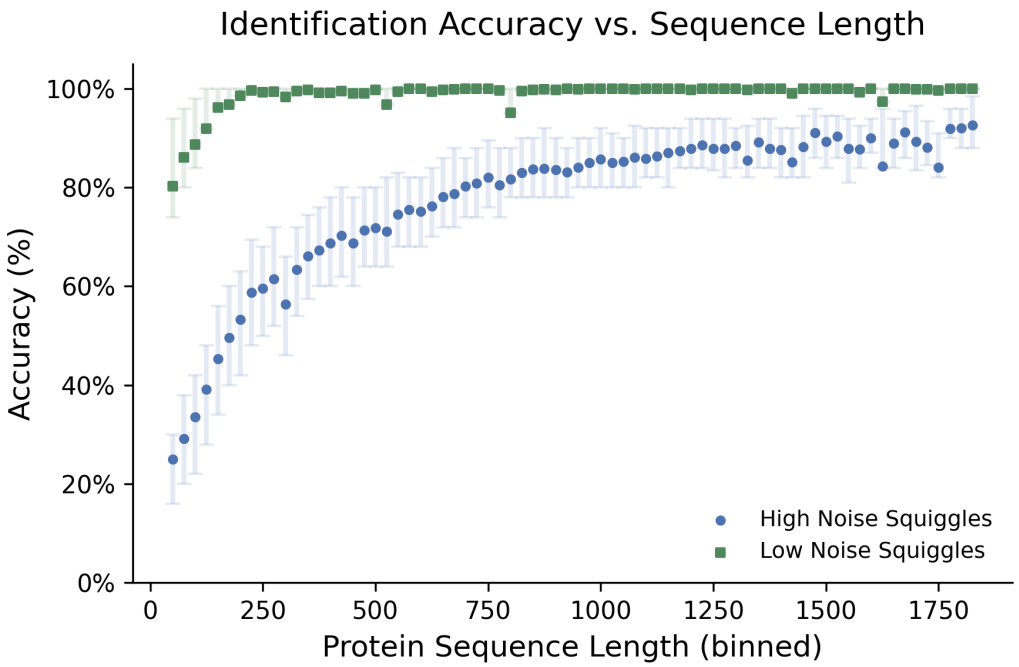

Figure 5.4: Shorter sequences have lower accuracy and higher variance, but in the low noise regime the typical high accuracy is recovered for sequences longer than 250 amino acids. In the high noise regime there is still a distinct correlation between sequence length and accuracy. Accuracy improves quickly up to sequence length 250, as seen in the low noise regime, and improves slowly but consistently beyond that. Even in the ideal case, short sequences are harder to identify. They are even harder to detect in the realistic high-noise regime, and sequence length overall has a more dominant effect on accuracy, since it doesn't stabilize until sequences are near 1000 amino acids long, many times longer than most protein sequences.

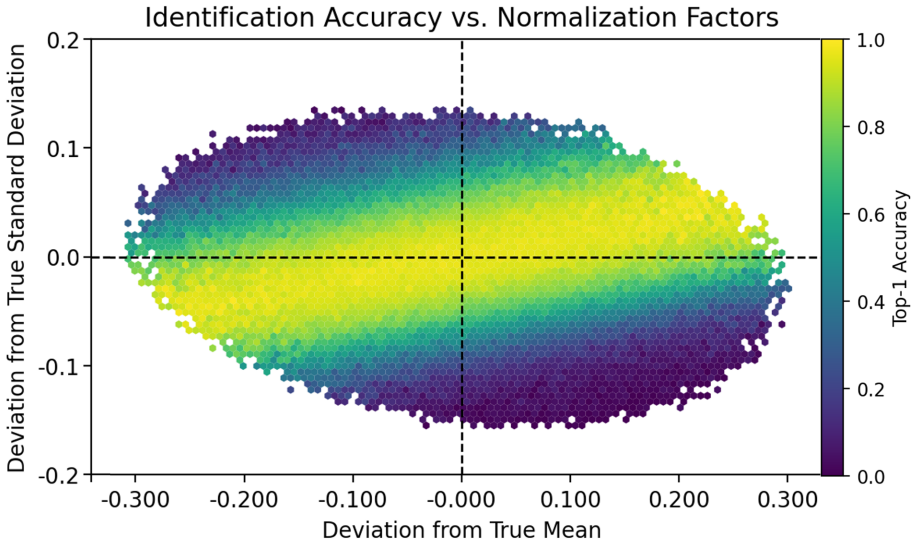

Figure 5.5: Impact of normalization on classification accuracy. Identification is highest near ideal normalization, but a diagonal band of shifted mean–variance pairs also yields high accuracy, while opposing shifts in mean and variance quickly reduce performance.

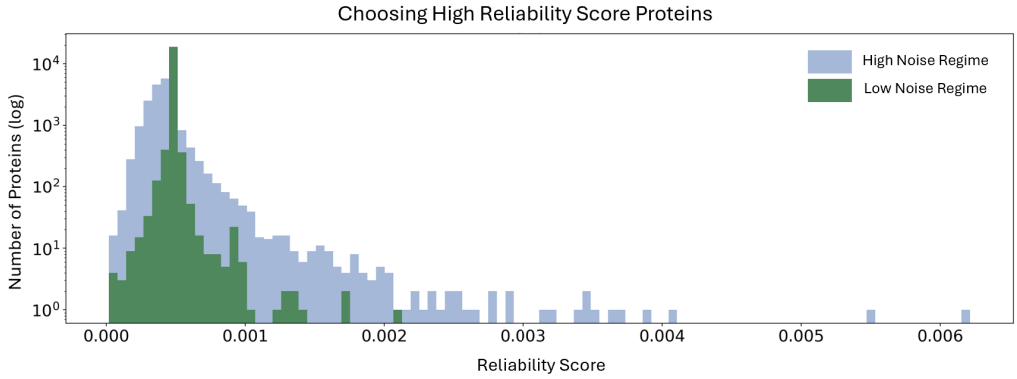

Figure 5.6: In the low noise regime, many proteins achieve perfect precision and recall, producing a sharp peak in reliability around 5×10^{-4} (the expected value for a label with exactly 50 correct matches). Reliability decreases when a label has fewer matches or false positives, and increases when it consistently matches to more than one protein. For example, evenly split matches between two similar proteins yield a small secondary peak near 10^{-3}. In the high noise regime, the sharp peak at 5×10^{-4} smooths out as errors accumulate, but this also reveals relationships between proteins: more labels show elevated reliability, indicating clusters of proteins that are frequently confounded.

Chapter 6

DISCUSSION

6.1 Proteome Coverage

To address our central research question—*How much of the human proteome can be identified using nanopore squiggles?*—we define two coverage metrics: **Any Coverage**, at least one correct identification across ten trials; and **Robust Coverage**, correct identification in all ten trials. For fair comparison with the MLP classifier—which uses a 20% test split—all methods were evaluated on a 20% holdout set, yielding ten classification trials per protein.

Method	**Any Coverage**	**Robust Coverage**
Low Noise Regime, Nearest Neighbor	99.87%	91.96%
Low Noise Regime, MLP Classifier	99.21%	86.85%
High Noise Regime, Nearest Neighbor	98.89%	10.44%
High Noise Regime, MLP Classifier	98.36%	33.34%

Table 6.1: Coverage Metrics Across Different Classifiers and Noise Regimes

6.1.1 Low Noise Regime: Upper Bound

In the low noise regime, nearly all proteins have at least one correct identification and 92% were identified across all ten trials. Protein squiggles tightly cluster around their canonical forms in signal space. For some identifications, modeled amino acid variance still disrupts alignment rankings. The MLP classifier, currently untuned, did not improve on this already high coverage.

6.1.2 High Noise Regime: Realistic Conditions

In the high noise regime, where we include all modeled stochasticity, nearly all proteins ($\approx 99\%$) had at least one correct identification. However, robustness drops, with only 10% of proteins achieve 10/10 accuracy with nearest neighbor. The MLP classifier recovers some ground, identifying a third of the proteome with 10/10 accuracy.

6.2 Sequence Length and Normalization

Because DTW is sensitive to amplitude differences, our pipeline uses a C-terminal tag to normalize dynamic range across signals. However, the mean and standard deviation of the tag—parameters used in z-score normalization—are not always estimated accurately. When the resulting normalization deviates from expectation, squiggle accuracy often declines, though not uniformly: some skewed normalizations still yield confident identifications, while others lead to steep accuracy losses.

We also found that shorter proteins are more difficult to identify. Short signals contain fewer distinctive features, particularly given the large (20–amino acid) reading window currently modeled. For short proteins, the C-terminal normalization tag dominates the signal, potentially obscuring informative regions. Refining the design and usage of a normalization tag could reduce normalization artifacts and improve the accuracy of identifying shorter proteins.

6.3 Dimensionality Reduction and Advanced Classifiers

Anchor-based dimensionality reduction proved highly effective: selecting 3,575 squiggles reduced dimensionality by 82% and decreased DTW computation time from 200–300ms to 35–55ms. The MLP classifier maintained or exceeded the performance of a naive nearest neighbor approach, particularly in high-noise conditions, increasing robust coverage from 10.44% to 33.34%. These results demonstrate that careful selection of anchor features can both accelerate and stabilize identification, with further

optimization likely to improve performance even more.

6.4 Comparison to Other Identification Methods

Our in silico results demonstrate that nanopore fingerprinting can be competitive with existing protein identification technologies while offering distinct advantages in resolution and full-length context. Compared to mass spectrometry, which remains the gold standard for depth and precision [1], nanopore fingerprinting provides single-molecule resolution without requiring destructive digestion or bulk sample preparation. While mass spectrometry benefits from extensive reference libraries, it struggles with incomplete coverage, low-abundance proteins, and differentiation of spliceoforms [26] [15]. Nanopore fingerprinting, by contrast, preserves the full-length sequence context, allowing for the potential resolution of proteoforms and post-translational modifications that are otherwise occluded in conventional MS workflows [23] [21].

Relative to emerging single-molecule approaches such as FRET X [8] and fluorosequencing [35], nanopore fingerprinting offers complementary strengths. Like these techniques, nanopore methods enable single-molecule detection, but they also provide direct, full-length signal information without requiring residue-specific labeling or complex chemistries. Whereas FRET X depends on measuring distances between labeled amino acids within folded proteins [8] and fluorosequencing relies on partial labeling and iterative cleavage [35], nanopore fingerprinting generates a continuous, quantitative signal from the native molecule.

Our work on feature reduction and multilayer perceptron classification highlights the potential for scalable protein identification. The use of anchor-based dimensionality reduction demonstrated that high-accuracy identification can be achieved with substantially fewer features, reducing DTW computation time by an order of magnitude while maintaining or improving classification performance. This suggests that nanopore fingerprinting could be extended to high-throughput or real-time applications, a domain where traditional MS and other single-molecule methods are either

slower or more resource-intensive.

These results place nanopore fingerprinting as a promising alternative in the proteomics landscape. It bridges the gap between high-resolution, bulk techniques like mass spectrometry and emerging single-molecule approaches, offering a combination of full-length context, quantitative precision, and computationally tractable analysis. While the technology remains in development and faces challenges such as noise sensitivity and algorithmic optimization, the findings presented here suggest that careful signal processing and machine learning strategies can make nanopore fingerprinting a viable tool for comprehensive human protein identification.

While the current MLP classifier and anchor-based dimensionality reduction demonstrate promising performance, future work could explore systematic hyperparameter optimization, more sophisticated feature representations, and advanced model architectures to further improve protein identification accuracy. Additionally, integrating these approaches with emerging wet-lab nanopore protocols could help guide experimental design and enhance the overall robustness and scalability of nanopore fingerprinting.

Technology	Strengths	Limitations	Development Stage
FRET X	Sub-nanometer precision; resilient to noise (e.g., photobleaching); single-molecule sensitivity	Low throughput; depends on 3D conformation; requires residue-specific labeling	Coverage analysis done in simulation; model builds on experimental results
Fluoro-sequencing	Encodes sequence, not structure; works on disordered peptides; potentially scalable; single-molecule resolution	Fragile chemistry; shallow read depth; heavy reliance on preprocessing and classifiers; limited to short peptides	Most analysis done in simulation; very early experimental stage
Mass Spectrometry	Depth; precision; extensive reference libraries	Incomplete coverage; bulk-only; splice-blind	Mature and well analyzed. Recent advancements include single cell proteomics and top-down approaches
Nanopore Fingerprinting	Single-molecule reads; full-length context; high potential for proteoform resolution	Emerging tech; identification-based (not sequencing); noise-sensitive	Coverage analysis done in simulation; model is built on experimental results

Table 6.2: Comparison of Protein Identification Technologies

Chapter 7

CONCLUSIONS

7.1 The Feasibility of a Nanopore Protein Identifier

Our proteome coverage simulation results show that protein squiggles have distinct, detectable signatures. Dynamic time warping alignment scores capture shape similarity and can be used to identify proteins based solely on their simulated nanopore squiggle. Under ideal conditions, these alignment scores carry enough identifying information that a naïve nearest neighbor classifier provides excellent coverage. Under more realistic noise conditions, nearest neighbor performance declines, but more advanced classifiers, like the multilayer perceptron tested here, can recover accuracy. Notably, our multilayer perceptron outperformed nearest neighbor while using only a small subset of alignment scores, highlighting the potential for computational efficiency without sacrificing performance.

Nanopores offer unique advantages for protein identification. They scan whole proteins at once, producing full-length reads that can capture splice isoforms and information about sequence length, which is not possible when proteins are fragmented for mass spectrometry or fluorosequencing. Our simulations also indicate that longer proteins are identified more accurately, making long reads especially valuable. The discrete, single-molecule nature of nanopore data allows direct counting of populations and detection of low-abundance proteins, while also revealing intra- and inter-molecular features that are typically obscured in bulk measurements.

Another advantage is the non-destructive nature of nanopore sensing. Proteins remain intact as they pass through the pore, opening the possibility of re-reading the same molecule to improve identification. This contrasts with top-down mass

spectrometry or FRET X, which require destructive processing or labeling and cannot preserve native molecules.

Current limitations of our modeled setup include the requirement for fully unfolded proteins and the reliance on a predefined set of labels and canonical squiggles. Proteoforms and post-translational modifications would dramatically expand the labeling space, which is not yet fully addressed. Nonetheless, our use of anchor-based dimensionality reduction demonstrates that it is possible to maintain distinct signal representations while keeping computation tractable, providing a pathway toward scaling to larger and more complex protein sets.

Overall, our *in silico* results demonstrate that nanopore fingerprinting is competitive with existing protein identification technologies. It bridges the gap between high-resolution, bulk approaches like mass spectrometry and emerging single-molecule methods, offering full-length context, quantitative precision, and computationally tractable analysis. Although the technology remains under development and challenges like noise sensitivity and algorithmic optimization persist, our results indicate that nanopore fingerprinting, in combination with careful signal processing and machine learning, could enable comprehensive protein identification, especially in applications requiring minimal sample input and high proteoform resolution.

Looking forward, this approach could be applied in several real-world contexts. Clinical proteomics could benefit from rapid, low-input identification of biomarkers directly from patient samples, enabling early disease detection or monitoring. Structural biology studies could leverage full-length nanopore reads to observe isoforms and post-translational modifications without fragmentation. Additionally, the single-molecule resolution of nanopore data opens the possibility of protein "base calling," analogous to how nanopore sequencing enabled DNA base calling. The challenge is considerably higher (we must resolve 20 amino acids across a large reading window) but with sufficiently advanced signal processing and machine learning, direct inference of primary protein sequences may be achievable. As wet-lab nanopore protocols

continue to develop, these *in silico* findings could help guide experimental design, for example in pore selection and engineering, tagging strategies, and rereading protocols. By combining experimental nanopore sensing with scalable machine learning frameworks, future research could further improve identification accuracy, extend coverage to larger proteomes, and open new avenues in proteomics and molecular diagnostics.

Together, these advances suggest that nanopore protein fingerprinting could transform proteomics, enabling rapid, high-resolution, and single-molecule insights previously out of reach.

BIBLIOGRAPHY

[1] Ruedi Aebersold and Matthias Mann. Mass spectrometry-based proteomics. *Nature*, 422(6928):198–207, March 2003. Publisher: Nature Publishing Group.

[2] Maria João Amaral, Rui Caetano Oliveira, Paulo Donato, and José Guilherme Tralhão. Pancreatic Cancer Biomarkers: Oncogenic Mutations, Tissue and Liquid Biopsies, and Radiomics—A Review. *Digestive Diseases and Sciences*, 68(7):2811–2823, 2023.

[3] AMD. AMD University Program AI & HPC Cluster.

[4] Yuwei Bao, Jack Wadden, John R. Erb-Downward, Piyush Ranjan, Weichen Zhou, Torrin L. McDonald, Ryan E. Mills, Alan P. Boyle, Robert P. Dickson, David Blaauw, and Joshua D. Welch. SquiggleNet: real-time, direct classification of nanopore signals. *Genome Biology*, 22(1):298, October 2021.

[5] Nicolas Cardozo, Karen Zhang, Kathryn Doroschak, Aerilynn Nguyen, Zoheb Siddiqui, Nicholas Bogard, Karin Strauss, Luis Ceze, and Jeff Nivala. Multiplexed direct detection of barcoded protein reporters on a nanopore array. *Nature Biotechnology*, 40(1):42–46, January 2022. Number: 1 Publisher: Nature Publishing Group.

[6] Xiaohan Chen, Shuo Zhou, Yunjiao Wang, Ling Zheng, Sarah Guan, Deqiang Wang, Liang Wang, and Xiyun Guan. Nanopore Single-molecule Analysis of Biomarkers: Providing Possible Clues to Disease Diagnosis. *Trends in analytical chemistry : TRAC*, 162:117060, May 2023.

[7] PyTorch Contributors. PyTorch, 2025.

[8] Carlos Victor de Lannoy, Mike Filius, Raman van Wee, Chirlmin Joo, and Dick de Ridder. Evaluation of FRET X for single-molecule protein fingerprinting. *iScience*, 24(11):103239, November 2021.

[9] dtai. dtaidistance: Distance measures for time series (Dynamic Time Warping, fast C implementation), 2025.

[10] John B. Fenn. Electrospray Wings for Molecular Elephants (Nobel Lecture). *Angewandte Chemie International Edition*, 42(33):3871–3894, 2003. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/anie.200300605.

[11] Hasindu Gamaarachchi, James M. Ferguson, Hiruna Samarakoon, Kisaru Liyanage, and Ira W. Deveson. Simulation of nanopore sequencing signal data with tunable parameters. *Genome Research*, 34(5):778–783, June 2024.

[12] Toni Giorgino. Computing and Visualizing Dynamic Time Warping Alignments in R: The dtw Package. *Journal of Statistical Software*, 31:1–24, August 2009.

[13] Toni Giorgino. dtw-python: A comprehensive implementation of dynamic time warping (DTW) algorithms., 2025.

[14] Justin Kale, Elizabeth J. Osterlund, and David W. Andrews. BCL-2 family proteins: changing partners in the dance towards death. *Cell Death and Differentiation*, 25(1):65–80, January 2018.

[15] Ryan T. Kelly. Single-cell Proteomics: Progress and Prospects. *Molecular & cellular proteomics: MCP*, 19(11):1739–1748, November 2020.

[16] Yu Li, Sheng Wang, Chongwei Bi, Zhaowen Qiu, Mo Li, and Xin Gao. DeepSimulator1.5: a more powerful, quicker and lighter simulator for Nanopore sequencing. *Bioinformatics (Oxford, England)*, 36(8):2578–2580, April 2020.

[17] Yusen Lin, Yongjun Zhang, Hang Sun, Hang Jiang, Xing Zhao, Xiaojuan Teng, Jingxia Lin, Bowen Shu, Hao Sun, Yuhui Liao, and Jiajian Zhou. NanoDeep: a deep learning framework for nanopore adaptive sampling on microbial sequencing. *Briefings in Bioinformatics*, 25(1):bbad499, January 2024.

[18] Laura Lopez-Gonzalez, Alicia Sanchez Cendra, Cristina Sanchez Cendra, Eduardo David Roberts Cervantes, Javier Cassinello Espinosa, Tatiana Pekarek, Oscar Fraile-Martinez, Cielo García-Montero, Ana María Rodriguez-Slocker, Laura Jiménez-Álvarez, Luis G. Guijarro, Soledad Aguado-Henche, Jorge Monserrat, Melchor Alvarez-Mon, Leonel Pekarek, Miguel A. Ortega, and Raul Diaz-Pedrero. Exploring Biomarkers in Breast Cancer: Hallmarks of Diagnosis, Treatment, and Follow-Up in Clinical Practice. *Medicina*, 60(1):168, January 2024.

[19] Fred W. McLafferty. A Century of Progress in Molecular Mass Spectrometry. *Annual Review of Analytical Chemistry*, 4(Volume 4, 2011):1–22, July 2011. Publisher: Annual Reviews.

[20] Essraa Metwali and Stephen Pennington. Mass Spectrometry-Based Proteomics for Classification and Treatment Optimisation of Triple Negative Breast Cancer. *Journal of Personalized Medicine*, 14(9):944, September 2024.

[21] Keisuke Motone, Daphne Kontogiorgos-Heintz, Jasmine Wee, Kyoko Kurihara, Sangbeom Yang, Gwendolin Roote, Oren E. Fox, Yishu Fang, Melissa Queen, Mattias Tolhurst, Nicolas Cardozo, Miten Jain, and Jeff Nivala. Multi-pass, single-molecule nanopore reading of long protein strands. *Nature*, 633(8030):662–669, September 2024. Publisher: Nature Publishing Group.

[22] Ben Noordijk, Reindert Nijland, Victor J Carrion, Jos M Raaijmakers, Dick de Ridder, and Carlos de Lannoy. baseLess: lightweight detection of sequences in raw MinION data. *Bioinformatics Advances*, 3(1):vbad017, January 2023.

[23] Ian C. Nova, Justas Ritmejeris, Henry Brinkerhoff, Theo J. R. Koenig, Jens H. Gundlach, and Cees Dekker. Detection of phosphorylation post-translational modifications along single peptides with nanopores. *Nature Biotechnology*, 42(5):710–714, May 2024. Publisher: Nature Publishing Group.

[24] Marketa Nykrynova, Roman Jakubicek, Vojtech Barton, Matej Bezdicek, Martina Lengerova, and Helena Skutkova. Using deep learning for gene detection and classification in raw nanopore signals. *Frontiers in Microbiology*, 13, September 2022. Publisher: Frontiers.

[25] Christian B. Rosen, David Rodriguez-Larrea, and Hagan Bayley. Single-molecule site-specific detection of protein phosphorylation with a nanopore. *Nature Biotechnology*, 32(2):179–181, February 2014.

[26] Kelly V. Ruggles, Karsten Krug, Xiaojing Wang, Karl R. Clauser, Jing Wang, Samuel H. Payne, David Fenyö, Bing Zhang, and D. R. Mani. Methods, Tools and Current Perspectives in Proteogenomics*. *Molecular & Cellular Proteomics*, 16(6):959–981, June 2017.

[27] Hari Sadasivan. harisankarsadasivan/DTWax, May 2025. original-date: 2021-12-19T21:09:03Z.

[28] Harisankar Sadasivan, Daniel Stiffler, Ajay Tirumala, Johnny Israeli, and Satish Narayanasamy. Accelerated Dynamic Time Warping on GPU for Selective Nanopore Sequencing. *Journal of Biotechnology and Biomedicine*, 7(1):137–148, February 2024. Publisher: Fortune Journals.

[29] Harisankar Sadasivan, Jack Wadden, Kush Goliya, Piyush Ranjan, Robert P. Dickson, David Blaauw, Reetuparna Das, and Satish Narayanasamy. Rapid Real-time Squiggle Classification for Read until using RawMap. *Archives of clinical and biomedical research*, 7(1):45–57, 2023.

[30] Bertil Schmidt and Christian Hundt. cuDTW++: Ultra-Fast Dynamic Time Warping on CUDA-Enabled GPUs. In *Euro-Par 2020: Parallel Processing: 26th International Conference on Parallel and Distributed Computing, Warsaw, Poland, August 24–28, 2020, Proceedings*, pages 597–612, Berlin, Heidelberg, August 2020. Springer-Verlag.

[31] Anjana Senanayake, Hasindu Gamaarachchi, Damayanthi Herath, and Roshan Ragel. DeepSelectNet: deep neural network based selective sequencing for oxford nanopore sequencing. *BMC Bioinformatics*, 24(1):31, January 2023.

[32] slaypni. fastdtw: Dynamic Time Warping (DTW) algorithm with an O(N) time and memory complexity., 2025.

[33] Lloyd M. Smith and Neil L. Kelleher. Proteoform: a single term describing protein complexity. *Nature Methods*, 10(3):186–187, March 2013. Number: 3 Publisher: Nature Publishing Group.

[34] Marc Suárez-Calvet, Thomas K Karikari, Nicholas J Ashton, Juan Lantero Rodríguez, Marta Milà-Alomà, Juan Domingo Gispert, Gemma Salvadó, Carolina Minguillon, Karine Fauria, Mahnaz Shekari, Oriol Grau-Rivera, Eider M Arenaza-Urquijo, Aleix Sala-Vila, Gonzalo Sánchez-Benavides, José Maria González-de-Echávarri, Gwendlyn Kollmorgen, Erik Stoops, Eugeen Vanmechelen, Henrik Zetterberg, Kaj Blennow, and José Luis Molinuevo. Novel tau biomarkers phosphorylated at T181, T217 or T231 rise in the initial stages of the preclinical Alzheimer's continuum when only subtle changes in Alpha-Beta pathology are detected. *EMBO Molecular Medicine*, 12(12):e12921, December 2020.

[35] Jagannath Swaminathan, Alexander A. Boulgakov, and Edward M. Marcotte. A Theoretical Justification for Single Molecule Peptide Sequencing. *PLoS Computational Biology*, 11(2):e1004080, February 2015.

[36] Oxford Nanopore Technologies. Data analysis | Oxford Nanopore Technologies.

[37] Nitinun Varongchayakul, Jiaxi Song, Amit Meller, and Mark W. Grinstaff. Single-molecule protein sensing in a nanopore: a tutorial. *Chemical Society reviews*, 47(23):8512–8524, November 2018.

[38] Roderick Corstiaan Abraham Versloot, Florian Leonardus Rudolfus Lucas, Liubov Yakovlieva, Matthijs Jonathan Tadema, Yurui Zhang, Thomas M. Wood, Nathaniel I. Martin, Siewert J. Marrink, Marthe T. C. Walvoort, and Giovanni Maglia. Quantification of Protein Glycosylation Using Nanopores. *Nano Letters*, 22(13):5357–5364, July 2022. Publisher: American Chemical Society.

[39] Chen Yang, Justin Chu, René L Warren, and Inanç Birol. NanoSim: nanopore sequence read simulator based on statistical characterization. *GigaScience*, 6(4):gix010, April 2017.

[40] Beatrice W. T. Yin and Kenneth O. Lloyd. Molecular Cloning of the CA125 Ovarian Cancer Antigen: IDENTIFICATION AS A NEW MUCIN, MUC16 *. *Journal of Biological Chemistry*, 276(29):27371–27375, July 2001. Publisher: Elsevier.

Appendix A

AMINOSCRIBE

A.1 Overview

AminoScribe is a Python module that can realistically simulate protein nanopore squiggles. It generates squiggles from either a sequence or UniProt accession ID, and optionally adds empirical time and amplitude noise. After squiggle generation it can optionally apply dynamic normalization using a C-terminal sequence or apply linear interpolation downsampling. All randomness can be made reproducible by passing in a seed value.

AminoScribe source code is publicly available at `https://github.com/uwmisl/Amino-Scribe` under the Apache License, Version 2.0. It is available on the Python Package Index (PyPI) at `https://pypi.org/project/aminoscribe` and can be imported into Python environments: `pip install aminoscribe`.

A.2 Synthetic Protein Dataset and Experimental Context

In the current experimental setup, proteins are engineered with a negatively charged tail at the N-terminus and a folded domain plus an ssrA tag at the C-terminus. This allows the protein to be initially pulled through the nanopore via electrophoretic force, up until the folded domain, which is too large to pass through the pore and halts the translocation. Once the proteins have been captured by the pores in this way, ClpX unfoldase is added. The ClpX motor protein recognizes the ssrA tag and ratchets the protein backward through the pore. As the protein is pulled back out through the pore we are able to capture the current signal, which varies based on what amino acids are in the pore at the time, providing a current level signature, or squiggle.

When the protein strand is initially pulled through the pore by electrophoresis forces, it translocates too quickly for the signal to be captured. The motor protein, in contrast, slows down the rate that the protein is passing through the pore, enough that many data points can be collected for each step of the ratchet.

Previous experimental work designed synthetic proteins made up of repeating modular segments. Each segment is bookended by tyrosine residues, which are large amino acids that cause a noticeable current trough. In between the troughs there's a spacer sequence that passes more easily through the pore and causes the current to rise. In the very middle of the segment there's a single variable residue, giving us 20 different possible segments, one for each amino acid. This design lets us compare the effect that different amino acids have on the signal—some amino acids will pull the peak of the segment even higher, and some will lower it into a trough.

Eight different synthetic proteins were designed, each composed of five different segments so that each of the 20 amino acids is represented twice as the variable residue. The lab collected translocation data for these proteins using an ONT MinION device and a dataset of 673 traces. The segments can be isolated using the distinctive double tyrosine troughs. Squiggle behavior was learned by analyzing confidently isolated segment squiggles.

A.3 Canonical model

The AminoScribe toolchain begins with a noiseless canonical squiggle model. This portion of the model takes a protein sequence as input and outputs a predicted nanopore signal. It takes the protein sequence and converts it into an array of impact scores. An impact score is a unitless number, unique to each amino acid, that represents the effect that that amino acid will have on the current. These impact scores are integrated over a 20-residue parabolic window[5] and convolved across the protein sequence to predict current levels. This generates a smooth, idealized squiggle shape.

A.3.1 Choosing Amino Acid Impact Scores

The amino acid impact scores were originally derived based on known amino acid features—each amino acid's charge and volume were combined linearly to predict current effects. The coefficients of the linear combination were chosen to fit the model to the observed data. This model worked well, but didn't capture the inherent variability in the impact that even the same amino acids might have. A second pass of tuning was done, this time allowing impact scores to be represented as Gaussian distributions, rather than a single deterministic point. Using the biophysically principled model as a starting point, Gaussian means and standard deviations were estimated for each amino acid, choosing parameters to fit the model to the observed data. Using Gaussian distributions allows for stochastic jittering in the impact scores for each amino acid. This can be optionally disabled, in which case AminoScribe will use the mean value deterministically.

A.4 Variable Translocation Speed

To simulate realistic variations in translocation speed through the nanopore, AminoScribe introduces time-domain noise using empirical step size and duration distributions. These distributions were extracted from dynamic time warping (DTW) alignments between real experimental squiggles and their corresponding noiseless templates using dtw-python[13, 12]. The result is a pair of frequency-weighted distributions:

- Step Size (in amino acids): How far to jump in the sequence for the next reading frame—how far the window should slide along the input sequence.

- Step Duration (in data points): How many samples the current step should be stretched to.

These distributions are later sampled to generate realistic time domain stretching.

A.4.1 Learning Stepping Behavior

To see what sort of time domain noise the signals experience I took the noiseless model and used DTW to align it to each of the experimental traces. This showed me how the experimental traces varied in speed – sometimes skipping ahead of the modeled signal, and sometimes pausing at a particular data point. Empirical distributions of step sizes and dwell times are extracted from these alignments.

This technique mimics stepping behavior, but doesn't extract step information directly. ClpX has discrete steps that skip over multiple residues, and significant dwell time between steps. The dtw alignments are not incentivized to have this behavior, and instead takes steps of size 1 most of the time, and then captures all the variability in the step duration parameter. The DTW alignment has such fine granularity that it prefers to mimic a smoothly sliding reading window, rather than abrupt jumps. This could be improved by adding constraints to the DTW algorithm, or by detecting and extracting steps directly.

A.4.2 Applying Stepping Behavior in AminoScribe

The observed distributions of step sizes and dwell times are used to apply realistic looking time warping to the noiseless model.

Time warping is applied in an independent and iterative fashion. For each simulated squiggle a list of step sizes and durations is drawn i.i.d. from the extracted empirical distributions. Starting at the beginning of the noiseless template, we scan forward, taking steps forward as determined by sampled step sizes. After each step, the corresponding value in the noiseless template is repeated for the sampled duration. This is repeated until the entire noiseless template has been scanned. The resulting signal is a time-domain distorted version of the noiseless template that mimics the natural variability in translocation timing. For performance, step sizes and durations are chosen up front instead of choosing a new sample in every iteration of the loop.

Random behavior can be set and made deterministic with a seed.

A.5 Noise

AminoScribe models amplitude-domain noise using experimental high-frequency current fluctuations. This noise is assumed to be happening at the signal level, not the sequence level. When generating noisy signals, amplitude noise is added after time warping and before optional filtering and downsampling. This ensures realistic signal variability and prevents the time warping of noise residuals.

A.5.1 Learning the Noise Profile

An experimental squiggle contains both valuable signal information as well as high frequency noise components. Often, in a nanopore analysis pipeline, a squiggle will be low pass filtered down to a cleaner signal of only low frequency components. In my case, noise values were extracted by using high-pass filtering with subtraction (residual analysis). A high pass Bessel filter was chosen to mirror the lab's filtering pipeline.

A.5.2 Applying Noise in AminoScribe

Each data point in the warped squiggle is perturbed by sampling from a weighted distribution of residuals. This noise model assumes independence across timepoints, which is a simplification, but offers a performant first-order approximation. This could be improved by coloring the noise (passing the sampled white noise back through the high pass filter to ensure that the applied noise has the intended frequency range).

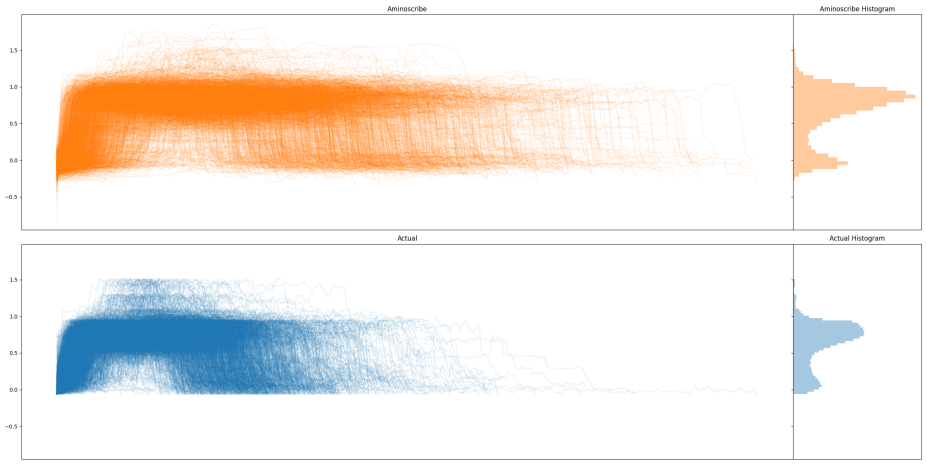

Figure A.1: A set of experimental squiggle snippets is compared to a simulated version of the set. Plot shows an overlay of all simulated (orange) or experimental (blue) squiggles used in this validation.

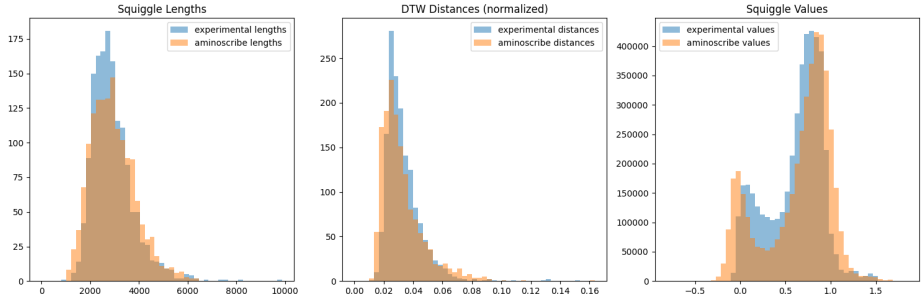

Figure A.2: The AminoScribe model was verified by comparing three squiggle features: Length, DTW distances, and current level values.

A.6 Model Verification

A.6.1 Squiggle Length

To check the accuracy of the time warping step of AminoScribe I compared histograms of the lengths of the experimental dataset and the lengths of a simulated version of the same sequences. The canonical model produces a signal that's always the length of the sequence minus 19 (due to the size 20 window), but the time warping stage of AminoScribe is stochastic and depends on the sampled step sizes and durations, yielding a variety of lengths for even the same input sequence.

The length histograms correlate very well, indicating that the time warping approach generates realistic squiggle lengths, giving us confidence in our algorithm of picking independent random samples from empirical observations.

A.6.2 Current Levels

To assess how well the canonical model as well as the applied amplitude noise are performing I took every data point from every experimental squiggle and plotted them as a histogram. I took a simulated version of each experimental trace and did the same, with the distributions showing very similar shapes.

This means that purely in terms of the current values that the simulation is reporting, it's behaving realistically. This tests both the time warping, which increases the number of data points observed during particularly long plateaus, as well as the noise model, which softens the harsh boundaries realistically.

A.6.3 DTW Alignment Scores

When you use DTW to align two signals, it reports a normalized distance metric that indicates how similar the two signals are after time warping. In the context of squiggles, a low distance indicates that the squiggles have similar underlying shapes.

The squiggles produced by the simulations would ideally have a similar distance score when compared back to the idealized template as an experimental trace would. In the later Monte Carlo simulation workflow we take a squiggle and look up a closest match based on this distance metric, so the closer these distributions are the more confidently we can draw conclusions when using simulated squiggles to represent experimental results.

Distributions match quite well, with the model performing very slightly better. This means the simulated squiggles appear to match their canonical shape at the same level as experimental squiggles. It's important to match and not dramatically improve

the distribution here–AminoScribe seeks to represent reality rather than improve on metrics.

Appendix B

DTWHIZ

DTWhiz is a GPU-accelerated implementation of the dynamic time warping (DTW) program designed for global alignments. DTWhiz takes a single query signal and performs global DTW alignments against a set of reference signals, returning alignment scores.

Dynamic time warping is a signal processing algorithm that can compare two signals with inconsistent time domain cadences. DTW is an invaluable tool in processing nanopore signals, since the translocating molecules have variable speeds. When comparing squiggles we don't want to penalize any speed-ups or stalls—the signal comparison must allow for insertions and deletions of signal values. DTW gives us a principled way to compare these signals, and returns a score that represents the similarity of two signals, with a lower score indicating the two signals are more similar in shape.

DTWhiz source code is publicly available at `https://github.com/uwmisl/dtwhiz` under the Apache License, Version 2.0.

B.1 Motivation

The Monte Carlo simulated test of SquiggleMatch requires DTW alignments between all simulated trial squiggles and all canonical squiggles. With nearly one million comparisons required for a full proteome coverage simulation, existing CPU-based DTW implementations become a major bottleneck, so we turned to an accelerated, GPU-based approach.

To benchmark performance we tested a SquiggleMatch run, which compares a

length 2048 query to 19780 references, each of length 2048. This represents the time to identify a single protein squiggles against our reference database. CPU performance benchmarks were run on a dual-socket AMD EPYC system with 128 physical CPU cores.

Running on a single core, Python implementations `dtw-python`[13] and `fastdtw`[32] took 0.41–0.58 hours to process our benchmarking workload, extrapolating to an estimated runtime of over 46–66 years for the full Monte Carlo simulation. Parallelizing `dtw-python` across 128 cores brings the runtime of a single SquiggleMatch down to a much more tractable 27.6 seconds, but running the full depth simulation would still take an estimated 0.87 years. Using `dtaidistance`[9] with its optimized C backend and parallel execution, a SquiggleMatch takes 4.9 seconds, projecting to 7 weeks for a full simulation.

Using our GPU implementation, the same workload completes in approximately 2.2 seconds, with the core kernel itself accounting for just under 0.20 seconds. When scaled to the full Monte Carlo simulation, this projects to a total runtime of roughly 25 days (a 49% speedup compared to `dtaidistance`), while the kernel execution alone corresponds to only 2.2 days (a 95.5% speedup). The gap highlights the impact of repeated `hipSetDevice` calls, which dominate the non-kernel overhead in the current setup. By batching multiple workloads per device and thereby amortizing GPU context initialization, the effective runtime could be brought much closer to the kernel-level performance limit.

Method	Per Protein ID Time	Full Simulation Runtime
fastdtw	35 minutes	66 years
dtw-python (single core)	24 minutes	46 years
dtw-python (128 cores)	27 seconds	11 weeks
dtaidistance (128 cores)	5 seconds	7 weeks
DTWhiz (end-to-end)	2.2 seconds	25 days
DTWhiz (kernel execution)	0.2 seconds	2.2 days

B.2 Parallelizing DTW with Wavefront Propagation

Dynamic Time Warping (DTW) is an algorithm for measuring similarity between two temporal sequences which may vary in speed or length. Unlike standard distance metrics, DTW allows flexible alignment by warping the time axis, effectively stretching or compressing sections of sequences to find an optimal match. Given a query $Q = [q_1, q_2, \ldots, q_n]$ and reference $R = [r_1, r_2, \ldots, r_m]$, DTW computes a $n \times m$ cost matrix $DTWScore$, where each element $DTWScore(i, j)$ represents the cumulative cost of aligning $q_1 \ldots q_i$ with $r_1 \ldots r_j$. The optimal alignment is obtained by minimizing the total cumulative distance along a warping path from $DTWScore(1, 1)$ to $DTWScore(n, m)$ and requires $O(nm)$ time complexity.

Matrix computations like this—filling a matrix along diagonals—can be parallelized using a wavefront propagation algorithm. Each thread is assigned a column (or a range of columns) and waits until the diagonal wavefront reaches its assigned position. When a thread computes a cell $DTWScore(i, j)$, the neighboring cells $DTWScore(i-1, j)$, $DTWScore(i, j-1)$, and $DTWScore(i-1, j-1)$ are already available. The top and top-left cells are maintained locally from previous computations, while the left cell can be retrieved from the preceding thread using a `shuffle up` intrinsic, minimizing inter-thread communication.

Since we work with a finite number of threads (32 on NVIDIA, 64 on AMD), this

Algorithm 1 Dynamic Time Warping

Initialize $DTWScore(0,0) = 0$, and $DTWScore(i,0) = DTWScore(0,j) = \infty$ for $i, j > 0$
for $i = 1$ to n **do**
    for $j = 1$ to m **do**
        cost $= \|x_i - y_j\|^2$
        $DTWScore(i,j) =$ cost $+ \min\big($
            $DTWScore(i-1,j)$, // insertion
            $DTWScore(i,j-1)$, // deletion
            $DTWScore(i-1,j-1)\big)$ // match
    end for
end for
return $DTWScore(n,m)$

can only fill in a submatrix with width equal to the number of threads. This algorithm can be extended to larger matrices with two strategies. Instead of giving exactly one column to each thread, threads can be in charge of a contiguous range of columns, computing a subsection of a row at a time. Alternately, we can tile the matrix into submatrices that are computed sequentially, with boundary results saved and stored for subsequent submatrix calculations. DTWhiz implements both strategies, with configurable values to allow for performance tuning.

B.3 DTWhiz Features

DTWhiz is developed as an extension of DTWax[27, 28], itself an extension of cuDTW[30]. DTWax was designed for local alignment and is coupled to NVIDIA GPUs via a CUDA dependency. DTWhiz introduces the following improvements.

B.3.1 Global Alignment

DTWhiz performs full-sequence dynamic time warping, enabling end-to-end comparisons rather than local matching. This is uncommon in the sequence or squiggle alignment space, where many applications focus on subsequence alignments. Our

goal is to align an entire squiggle to a potential match shape—rather than asking "does this shape appear anywhere in the squiggle?" we ask "does this shape match the overall shape of the squiggle?"

Global alignment is enforced by treating the top and left boundaries as containing ∞. This prevents cells in the top row from aligning from above or diagonally as the only valid move is from the left neighbor. Similarly, infinity values on the left enforce that paths must come from above, not from the left or top-left. To ensure alignment at the ends of the signals, we take the score in the bottom-right cell, which corresponds to the final points of each signal.

B.3.2 Platform Portability

The codebase was HIPified, allowing DTWhiz to run on NVIDIA and AMD GPUs, increasing accessibility and hardware flexibility.

B.3.3 Test Suite

DTWhiz includes a comprehensive test suite that generates input data and verifies results against `dtw-python` as a ground truth. A large set of tests uses only the alignment scores produced by production DTWhiz builds, while a smaller, more detailed subset runs in debug mode to recover and verify the full alignment score matrix. The test harness is implemented in Python and can be configured to run custom tests or a selected subset of the provided cases.

B.3.4 Alignment Score Matrix Recovery and Visualization

DTWhiz returns only the final alignment score, $DTWScore(n, m)$. The alignment score matrix is intentionally not kept in memory for performance and tractability (large workloads cannot fit the alignment matrices in memory at all). But while debugging during development, knowing only that the final answer is wrong doesn't

point to what is going wrong. To see where implementation errors are happening we must recover the full alignment score matrix and compare element-wise to the expected values.

To that end, we added `DEBUG` directives that allow the code to optionally run in a mode which collects the full score matrix in memory and transfers the results back to the host at the end of the computation. The `#ifdef DEBUG` construct gates a separate debug pathway that allocates memory for the full score matrix, without affecting the behavior of production builds.

Once the alignment score matrix is recovered it is compared to a ground truth score matrix, computed with the `dtw-python` [13] module. Differences between the DTWhiz reported alignment matrix and the ground truth matrix can be plotted visually, highlighting error locations and propagation.

We observed occasional alignment score discrepancies when running on large float arrays. These differences were intermittent, and true/false plots revealed no systemic patterns. The DTWhiz kernel uses 32-bit floats, while our ground truth calculations employ a Python library with 64-bit floats. We hypothesized that the discrepancies arose from floating-point precision differences. To investigate, we developed a matrix visualization tool that generates a heatmap of the cell-wise value differences rather than a binary true/false plot. This revealed how precision differences propagate across the matrix. For our use case, input arrays are small and noisy enough that the precision loss is negligible, but users should be aware of this effect if 64-bit precision is required.

The alignment matrix is divided into submatrices that are filled iteratively, with careful saving and reading of boundary results. We discovered a subtle bug affecting the top-left diagonal element at the border: this value was not being saved like other boundary data. As a result, interior submatrices occasionally treated it as if it were infinite, producing errors when the optimal alignment passes through this missing diagonal.

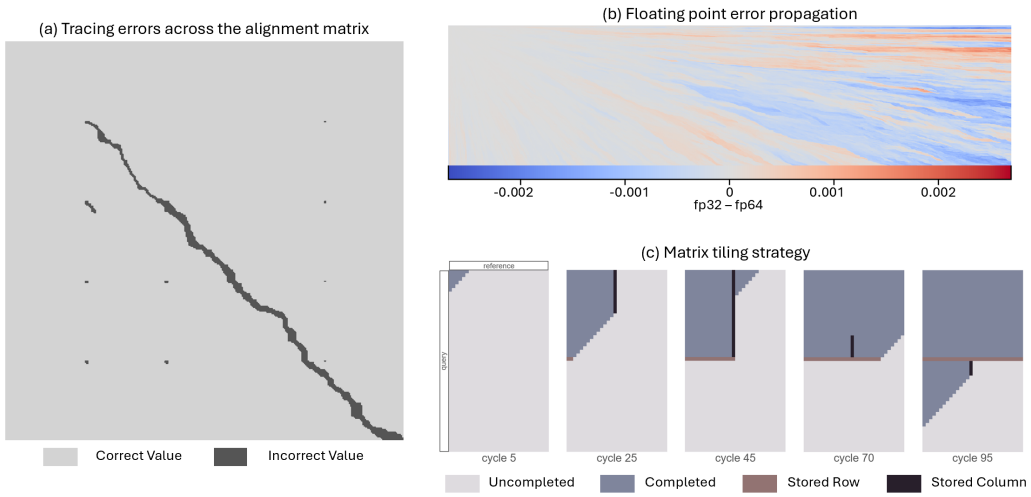

Figure B.1: (a) Differences between the DTWhiz reported alignment matrix and the ground truth matrix can be visualized, highlighting error locations and propagation. In this case, errors were caused by a diagonal border element. Small errors in the top left of matrix tiles appear like a dot grid. For most tiles the error corrected itself because the algorithm found better alternate paths. However, in this example, an error propagated through the entire computation and resulted in an incorrect final alignment score. (b) DTWhiz uses 32-bit floats, while most Python and many C implementations use 64-bit floats. Floating point errors propagate across the alignment matrix. (c) Wavefront propagation computes the DTW alignment matrix in parallel along moving anti-diagonals. Large matrices are divided into tiles, and boundary information is stored between tiles. To conserve memory, the alignment score matrix is not stored—only the moving wavefront is retained and prior calculations are discarded. Similarly, the stored boundary row and column results are kept only until needed, as shown by the elements returning to the completed state.

To correctly allow alignment paths across tiling diagonals, the top-left diagonal value must be explicitly saved. Since the alignment matrix is not otherwise recorded, we must be intentional about when and where this value is stored. During execution, the value is used by the previous submatrix as the "top" for its last column. The last thread in the submatrix writes this value to memory, and because it will immediately be consumed by the next submatrix, only a single `float32` is needed to track it. Memory for the full distance score is already allocated once the matrix computation is complete, so we could reuse that variable—effectively recording the final DTW alignment score for each submatrix without additional memory overhead.

B.4 Performance Profiling

To evaluate and optimize the performance of DTWhiz, we profiled a sample workload that aligns a set of 19780 canonical squiggles padded to length 2048 to a noisy squiggle that was padded to length 2048. This is the lower end of typical workloads, which vary between lengths 1024 and 4096.

Performance tests ran on AMD Instinct™ MI210 Accelerators. These GPUs have up to 22.6 TFLOPs of peak single-precision performance and 1.6 TB/s of memory bandwidth. We were able to reach 1.87 TFLOPs and found we have a ∼91 FLOPs/byte arithmetic intensity, indicating DTWhiz is compute bound.

B.4.1 TFLOPs

To benchmark the TFLOPs we estimated the FLOP count analytically and then cross-checked it using ROCm's `rocprof` profiler.

At the heart of DTWhiz is a fused multiply-add (FMA)-based score function, used to populate each cell of the alignment matrix:

```
fma(fma(r - q, r - q, 0), 1, min(left, min(top, diag)))
```

Here, `fma(a, b, c)` computes $(a \times b) + c$ as a single fused operation, reducing rounding error and improving performance.

At first glance, this expression appears to involve six floating-point operations per matrix cell: two subtractions, two multiplications, and two additions (following the industry standard of counting a `fma` as two operations, and not including the `min()` operations). For a typical workload involving 19,780 arrays of length 2048 each being aligned against a length 2048 array, DTWhiz must compute approximately 83 billion matrix cells. Based on the naïve 6-ops-per-cell assumption, we initially estimated a workload of nearly 498 billion floating-point operations, and a throughput of approximately 2.55 TFLOPs given a 195 ms runtime.

However, when we validated this using AMD's `rocprof` profiler, we encountered a surprising discrepancy: the profiler reported only 365 billion operations. Digging deeper, we realized the compiler was further optimizing this expression. The outer `fma(a, 1, b)` simplifies to a single addition, and the subtraction `(r - q)` is reused within the inner FMA. As a result, the actual operation count drops to roughly four FLOPs per cell, reducing the expected total to about 332 billion—nearly matching the profiler's report. This brought the sustained throughput to 1.87 TFLOPs for our workload of 19780 signals of length 2048 compared to a signal of length 2048, $19780 \times 2048 \times 2048 =$ almost 83 billion matrix elements.

B.4.2 Arithmetic Intensity

We also computed the arithmetic intensity — the number of FLOPs per byte of data moved — as a way to understand the bottlenecks. For this workload, arithmetic intensity was estimated at around 91 FLOPs/byte, vastly exceeding the hardware's roofline of 14.1 FLOPs/byte. This indicates that DTWhiz is strongly compute-bound, not memory-bound. This is expected as the number of operations scales quadratically with the signal lengths.

B.4.3 VALU and Occupancy

Additional profiling uncovered high vector arithmetic logic unit (VALU) utilization (95.6%) indicating compute saturation. But occupancy — the proportion of compute units (CUs) actively engaged — was moderate, around 68%. When threads are engaged they are fully utilizing the VALU, but not all available threads are being used.

Wavefront propagation has to ramp up to full occupancy by filling in the initial corner triangle of the matrix–in cycle 1 only thread 0 is doing work, in cycle 2 there are two threads active, etc. The same is true of the final bottom right corner triangle. When filling in these regions we get only about 50% occupancy. In the data setup

we're using, the matrix is divided into two tiles, with the startup and cooldown triangles occupying a significant portion of the total computation. All threads are active for a central diagonal band across the matrix tiles, bringing expected occupancy to (50%+100%+50%)/3 = 67%, as observed. The impact of the wavefront propagation triangle corners can be mitigated with narrower tiles, which directly reduce the size of the triangles and push to full occupancy faster. The tradeoff is that more tiles means more data reads and writes as tile boundaries are stored and retrieved.

B.4.4 The Kernel in Context: The Cost of `hipSetDevice`

The DTWhiz kernel itself executes extremely quickly once running on the GPU. However, the end-to-end runtime of a single workload is dominated not by the kernel, but by GPU setup overhead. In our measurements, the total runtime per workload is about 2.2 seconds, with the majority of that time attributable to the call to `hipSetDevice`. This call initializes the GPU context and effectively "wakes up" the device. Since each DTWhiz workload is dispatched independently, each incurs this startup penalty.

This distinction between kernel execution time and program runtime is important: while the kernel demonstrates high efficiency, the apparent performance is limited by GPU context initialization. In practice, batching multiple workloads into a single invocation would amortize the cost of `hipSetDevice` across the batch, substantially reducing the per-workload runtime. For this reason, we report both kernel performance (to highlight algorithmic efficiency) and end-to-end runtime (to reflect current usage patterns), and note that the latter can be improved with standard batching techniques.

B.5 Future Work

While DTWhiz already provides substantial GPU acceleration, future work could explore systematic tuning of kernel launch parameters, wavefront tile sizes, and batching

strategies to further reduce end-to-end runtime. Such refinements would make large-scale Monte Carlo proteome simulations more tractable and enable rapid nanopore squiggle analysis in high-throughput applications.

Appendix C

SIMULATION PIPELINE

C.1 SquiggleMatch: Efficient Matching Engine

The SquiggleMatch pipeline integrates the AminoScribe signal simulator and the DTWhiz GPU-accelerated kernel to simulate protein identification at the squiggle level. The core idea is to compare a squiggle simulated by AminoScribe to a set of canonical squiggle shapes and identify the most likely match using global DTW alignment scores.

C.1.1 Inputs

The Canonical Squiggle Set

A set of reference squiggle shapes was built using AminoScribe in canonical mode. This set contains a squiggle for each protein sequence retrieved from UniProt Homo sapiens proteome reference dataset UP000005640, provided the sequences didn't contain an X (unknown amino acid) or U (selenocysteine), and weren't too long to fit into our padded length. This reduces the original 20,647 reference sequences to 19,772. An N-terminal tag and a C-terminal normalization tag were appended. From this sequence a canonical squiggle was generated and used when trying to match unknown squiggles.

Simulated Unknown Squiggle

To emulate experimental squiggles, AminoScribe can be configured to apply stochastic behaviors, including amino acid impact score variance, time warping, and signal noise.

These protein sequences have the same leader and normalization tags appended to match reference conditions.

C.1.2 Output

Given a single noisy squiggle, SquiggleMatch uses DTWhiz to compute a global DTW distance between the query and each reference squiggle. Each alignment produces a distance score, and the reference protein with the lowest score is taken as the predicted identity. All scores are retained for downstream analysis.

C.1.3 Implementation Details

The pipeline is implemented in Python, using AminoScribe and calling DTWhiz via the subprocess module. Reference squiggles are stored in binary format for efficient I/O.

C.2 Simulation Methodology

A Monte Carlo simulation framework was developed to evaluate SquiggleMatch under simulated experimental conditions. Each protein was simulated 50 times using AminoScribe with stochastic variation, and aligned using DTWhiz to the canonical database.

C.2.1 High Throughput Framework

A script, `process_protein.py`, uses SquiggleMatch to generate 50 squiggles for a particular protein. It then uses DTWhiz to align each squiggle against the reference proteome. Instances of this script launched via `process_many_proteins.sh`, capped at 16 concurrent jobs across four GPUs. A top level Python script submits one SLURM job per batch of 1024 proteins. The proteome is split into 20 files, each running for 3-4 hours.

C.2.2 Data Batching

All squiggles are padded to a uniform length (e.g., 2048), using high constant values to avoid score bias. Padding is also applied to noisy squiggles. Each squiggle must include at least one padding value.

C.2.3 Distributed, Safe, and Efficient Building

DTWhiz requires query and reference lengths to be specified at compile time. To avoid pre-building numerous executables and to prevent corrupted builds when multiple processes compile concurrently, we developed a build wrapper. The wrapper first checks for an existing executable; if none is found, it waits to acquire a lockfile. Once the lock is obtained, it writes `data_dims.h` and builds DTWhiz. The resulting executable is memoized in a shared directory, after which the lock is released. This approach ensures safe, concurrent building while minimizing redundant compilation.

C.3 Results Storage and Format

Each lookup produces 19780 DTW alignment scores, each a `float32`. Results are stored in Apache Parquet format, partitioned by `protein_id`. Each job writes to its own subfolder, avoiding the need for file locking. Results can also be streamed and queried without fully loading into memory. Raw squiggles are discarded, but seeds and metadata are retained.

C.4 Low Noise Regime

AminoScribe was run with no time warping or noise. Only sequence-level differences contributed. The reference database used two batches (padded to 1024 and 2048) with `pad_value=5`. A few proteins had squiggle values above 5, but post-experiment checks confirmed no impact on alignments.

C.5 High Noise Regime

All stochastic features were enabled. Squiggles were filtered and downsampled by 20x. DTWhiz was run with a single reference batch (padded to 2048 with `pad_value=50`). Proteins with long sequences were excluded. This configuration captured the full modeled variability and serves as the benchmark dataset for classifier evaluation.