Computational design and sensing algorithms for nanopore-based molecular tagging and peptide detection

Kathryn Doroschak

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

Doctor of Philosophy

University of Washington

2021

Reading Committee:

Luis Ceze, Chair

Karin Strauss

Jeff Nivala

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Computational design and sensing algorithms for nanopore-based molecular tagging and peptide detection

Kathryn Doroschak

Chair of the Supervisory Committee:
Professor Luis Ceze
Computer Science and Engineering

Molecular sensing provides a window into the complex world of otherwise invisible molecules, allowing us to measure protein abundance or sequence DNA, for example. Commercially available nanopore arrays have already made DNA sequencing less expensive and more portable than existing platforms, and they have recently emerged as potential tools for general purpose molecular sensing. Nanopore arrays record a time series of ionic current observations and do not intrinsically detect any particular types of molecules; any molecule that can physically flow through the pore will partially block the ionic current flow in unique ways depending on its physical properties, producing a characteristic current trace. Since only DNA and RNA sequencing are officially supported, any applications beyond straightforward DNA sequencing require developing novel computational pipelines and algorithms to extract biologically relevant information.

Here I present computational methods for three novel uses of commercial nanopore devices: (1) Porcupine, a molecular tagging system using custom designed nanopore-orthogonal DNA molecular bits (molbits); (2) Big Bits, a DNA data storage implementation using sequentially encoded molbits; and (3) Poretitioner, a pipeline for identifying NanoporeTERs (NTERs, Nanopore-addressable protein Tags Engineered as Reporters) and other engineered molecules. In each chapter, I present my contributions to novel computational analysis of

nanopore data for these applications. Briefly, Porcupine labels physical objects using molecular tags. These tags encode digital information via the presence and absence of molbits, which I algorithmically designed to produce visually unique nanopore signals. The tags are later read back and decoded directly from the nanopore ionic current trace using a convolutional neural network (CNN). Big Bits extends upon this, using design principles from Porcupine to encode even more information for DNA data storage. Instead of using presence or absence to encode information, molbits in Big Bits are encoded sequentially. In the Poretitioner pipeline, I extract ionic current for captured peptides, then filter, classify, and quantify them using components built by both myself and others that can be tuned for various molecules.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Throughout my graduate school career and preparing this thesis, I received a great deal of support and assistance, for which I am very grateful.

I would first like to thank my partner, Sven Dorkenwald, for his love and support throughout my graduate school experience. Your support means more to me than I can convey.

I am extremely grateful to my advisors, Professors Luis Ceze, Karin Strauss, and Jeff Nivala, whose expertise was invaluable in defining project ideas and research questions, and providing insightful feedback and encouragement. You have all been wonderful to work with and helped me grow as an independent researcher and leader, and encouraged me to do more than I thought I could. Luis, your genuine excitement is contagious and helped me be even more enthusiastic about my work. Thank you for encouraging me to join this lab and for funding me throughout these years.

Thank you also to my previous advisor, Professor Larry Ruzzo, who was beyond patient with me during my early years in graduate school. I enjoyed our deep discussions about research problems and getting to work with you as your teaching assistant for so many classes.

I would like to thank my colleagues in the Molecular Information Systems Lab for their wonderful collaboration. In particular, thank you to Karen Zhang (who has prepared countless nanopore experiments and pipetted hundreds of molbits – thank you for your undying patience with me), Lee Organick, Aishwarya Mandyam, Jessica Dunstan, and Melissa Queen for all your support, encouragement, honest feedback, and friendship throughout the years.

I could not have completed this degree without the support of my friends and family. Thank you to my mom and dad for doing their best to proudly explain my research (well,

close enough anyway), and thank you to both of you and Maya for always being there no matter what. An enormous thanks goes to Lindsey Harris, Amelia Holm, Emily Seitz, Jacob Schreiber, Naozumi Hiranuma, Jiechen Chen, and other friends – I don't know how I would have gotten through this without you.

I am grateful to my colleagues/labmates during my internship with Microsoft, Bichlien Nguyen, Yuan Jyue-Chen, Gabe Reder, and Karin Strauss, for a great experience during a very unusual summer.

Thank you to the CSE advising staff, in particular Elise Dorough, who manages to tirelessly orchestrate seemingly everything, and yet is always present and helpful.

Finally, this is in memory of my undergraduate advisor Professor John Carlis, who made it his personal mission to make me more assertive, spirited, and self-assured. Even though you kicked me out of your office for saying sorry too many times, and feigned astonishment during our Scrabble lab meetings that I could not both win at Scrabble and discuss research at the same time, I needed the tough love and would not be studying computer science today without your early support and understanding.

Thank you!

Chapter 1
INTRODUCTION

Molecular sensing enables a wide range of exploration into molecules and how they behave in biological and synthetic experiments. Nanopore arrays are just one type of molecular sensor, which measure ionic current across nano-scale pores and are typically used for DNA sequencing. Access to nanopore technology is widening due to growing commercial availability, lower costs than comparable methods, and increased portability. Since access to sequencing technology has historically been limited to established, well-funded biology labs, new applications have been increasingly varied, including DNA sequencing in the antarctic dry valleys [32] and rapid detection of SARS-CoV-2 [44, 11]. Official support is currently limited to DNA and RNA sequencing, but nanopore arrays are capable of sensing a much broader variety of analytes, as demonstrated by their lab-made counterparts [7, 8, 15, 38].

In this thesis, I explore three novel applications of nanopore arrays: molecular tagging, DNA data storage using nanopore-orthogonal bits, and a pipeline for general purpose use of commercial nanopore devices.

1.1 Background

1.1.1 DNA sequencing

The currently dominant method for DNA sequencing is sequencing-by-synthesis, often referred to as 2nd-generation sequencing or Illumina sequencing (Illumina is the main manufacturer) [22]. As a brief, abstract overview, this method sequences DNA by making copies of a cluster of identical DNA strands, adding one nucleotide at a time to each strand. Each added nucleotide contains a uniquely colored fluorophore tag. A camera and strobe then identifies the nucleotide after each addition using the color of the fluorphore. This sequencing method

is appealing because it has high accuracy and high throughput, however, the machines have a high upfront cost (∼$250k) in addition to consumables, and require careful calibration to the extent that a machine cannot be moved without professional assistance. Other drawbacks include a relatively short sequencing length (up to 300 nt, or 150 nt if "paired-end", which includes 150 nt from each end of a DNA strand) and the nature of copying DNA removes chemical modifications present in the original DNA strand that may be biologically relevant.

1.1.2 Nanopore sequencing

Nanopore arrays record ionic current as it flows through a nano-scale pore embedded in a membrane. The pores can be made of either protein or a solid-state material, and are usually shaped like a barrel or tube with a restricted neck. A few manufacturing challenges needed to be overcome before they could be commercially viable. One physical challenge of manufacturing them is controlling the size and shape of the nanopore, since the size and shape cause the ionic current to vary. Another challenge is stably embedding the pores in the membrane, especially in desired locations.

Nanopore sequencing became commercially available for early access in 2015 when Oxford Nanopore Technologies (ONT) released the MinION. The MinION is a granola bar-sized device that plugs into a laptop, bringing portability to the sequencing field. In contrast to 2nd-generation sequencing, nanopore sequencing operates at a single molecule level, that is, one DNA fragment fits in the pore at a time, and therefore only that molecule blocks ionic current as it is flowing through the pore [30]. DNA, which is negatively charged, is added to the *cis* (top) side of the flowcell and attracted to the positively charged *trans* (bottom) side of the pore. Since the DNA is small enough to fit through the pore, if no sequencing adapter is added, it flows through very quickly and is undetectable. So, a sequencing adapter is bound to the end of each fragment, which unwinds the DNA double helix at a stochastic rate and slowly feeds it through the pore. This unwinding creates distinct levels in the recorded ionic current trace. For the MinION R9.4.1 pore, the sensitive region of the pore (the narrowest restriction in the neck of the pore) fits 5-6 nucleotides, and thus measures ionic current from

5-6 nucleotides at once. Hence, the ionic current must be computationally deconvolved in order to recover the nucleotide sequence.

The MinION flowcell has 512 individually controlled wells, each of which contains four nanopores [30]. The device can read from one of the four pores in each of the 512 wells, with all 512 wells measuring ionic current at the same time. After the ionic current data is collected for a read (one DNA fragment), it is stored in the FAST5 file format, which is an HDF5 file with a specification placed on top dictating where data is stored. The reads are deconvolved from ionic current to nucleotides through a process called basecalling. Nanopore sequencing devices are inherently more prone to noise due to this ionic current sampling method, and basecalling accuracy suffers as a result. Applications that use the ionic current signal directly instead of requiring basecalling may be more tolerant of noise.

1.1.3 Other nanopore use cases

Nanopores are broadly useful beyond DNA sequencing, although commercial nanopores have not yet been used to do so. Much like for DNA sequencing, using commercial nanopores for peptide sensing could have a transformative effect on the field through portability and access. So far, only non-commercial nanopore arrays have been used to sense molecules other than DNA or RNA [7, 8, 15, 38]. These lab-made devices are appealing due to their custom configuration and flexiblity of pore chemistry. For example, non-biological pores can use a protease to unfold or stretch peptides, whereas protease would degrade the MinION's pores. However, the limited scalability and manufacturing inconsistencies in non-commercial nanopores makes them impractical for widespread use. In contrast, any methods for the MinION could easily be widely adopted, especially when released with stable tools and bioinformatic support.

1.2 Chapter outline

In each chapter, I present my contributions to novel applications of commercial nanopore devices and my computational approaches for analysis.

- **Chapter 2: Porcupine molecular tagging system** Porcupine, a molecular tagging system using custom designed nanopore-orthogonal DNA molecular bits (molbits);

- **Chapter 3: Data storage using sequentially encoded molecular bits** Big Bits, a DNA data storage implementation using sequentially encoded molecular bits;

- **Chapter 4: Peptide barcode sensing with NanoporeTERs and Poretitioner** Poretitioner, a pipeline for identifying NanoporeTERs (NTERs, Nanopore-addressable protein Tags Engineered as Reporters) and other engineered molecules.

1.3 Contributions

1.3.1 Porcupine molecular tagging system

My contributions can be divided into three sections for this project: (1) molbit design, (2) identifying molbits, and (3) demonstrating the system end-to-end.

1. **Molbit design**: I created a broadly useful evolutionary model to design unique nanopore sequences within the bounds of predetermined signal and sequence characteristics. I also produced a set of 96 sequences for the Porcupine molecular tagging system which generate highly separable ionic current traces.

2. **Identifying molbits**: I demonstrate a strategy to label molbits using nanopore sequences, a model to classify which molbit is present in each read using deep learning, and an approach to generating and balancing training and test data.

3. **End-to-end system demonstration**: My contributions include a method to binarize nanopore read counts into a bit string, encoding real data in a molecular tag with error correction (although I did not choose, design, or implement the error correction method itself), and an analysis of the time required to decode molecular tags.

1.3.2 Data storage using sequentially encoded molecular bits

Similarly to Porcupine, my contributions in this area can be divided into three sections: (1) designing molbit strands, (2) labeling molbit strands, and (3) identifying molbits.

1. **Designing molbit strands**: I produced a structural design for sequentially encoding molbits for DNA data storage (*molbit strands*), an approach to designing molbits and their adapters, and three sets of molbits to test.

2. **Labeling molbit strands**: I wrote a simple algorithm to label the highly repetitive sequences found in molbit strands. I also led the data acquisition, carried out labeling for the molbits I designed, and analyzed label correctness.

3. **Identifying molbits**: I developed and tested an initial set of machine learning models for identifying molbits. I also created a custom Pytorch FAST5 dataset module for quickly loading nanopore data to machine learning models.

1.3.3 Peptide barcode sensing with NanoporeTERs and Poretitioner

My contributions to Poretitioner and NanoporeTERs are mixed with others' contributions; I will clarify that here. Work for Poretitioner was carried out by myself and Jessica Dunstan, using aspects of the code previously designed for the NanoporeTER publication [13] by myself, Jeff Nivala, and Karen Zhang.

- **NanoporeTER publication**: I provided all of the initial infrastructure to process data. During the very beginning stages, the goal was simply to avoid segmenting data by hand. To date, nearly all of the MISL lab's nanopore data has been processed using my segmentation infrastructure. I helped with very early classification approaches that informed the current classifier but did not make it into the publication.

- **Poretitioner**: My contributions include refactoring and improving the aspects of the pipeline that produced the scientific results (i.e. the individual steps of segmenting,

filtering, classifying, and quantifying data). I also defined a new peptide FAST5 specification for general purpose usage (non-DNA molecular sensing). Jessica Dunstan provided contributions to the infrastructure of Poretitioner.

Chapter 2

PORCUPINE MOLECULAR TAGGING SYSTEM

2.1 Introduction

Portions of this chapter were previously published as a manuscript in Nature Communications [18].

Tagging physical objects has proven useful for a range of formats and scenarios like UPC barcodes in packaging, QR codes for easy association of digital information with printed material, and radio-frequency identification (RFID) tags for inventory tracking. However, these tags cannot be applied to objects that are too small, flexible, or numerous, or in scenarios where the code should be invisible to the naked eye, like anti-forgery. Molecular tags address these shortcomings via their nanoscale footprint and difficulty to forge. However, existing methods for encoding digital information in molecules; including silica-encapsulated DNA tracers [48], DNA embedded in 3D printed material [36], microbial barcodes [57], or spatially isolated marker peptides [12]; require access to specialized labs and equipment to make new tags — making them impractical in applications that require a very large number of tags. In most cases, the protocol prevents real-time use cases, for example, in the case of PCR- or SHERLOCK-based detection which takes tens of minutes in the best case [34]. An ideal molecular tagging system should be inexpensive and reliable, with fast readout and user-controlled encoding and decoding from end-to-end with minimal reliance on lab equipment.

Molecular biologists have also long been interested in tagging biomolecules, for example with multiplexing samples in a single sequencing run through the inclusion of a sample-specific short DNA tag appended to each sequenced fragment. For nanopore sequencing in particular, direct raw signal techniques have improved efficiency of dereferencing multi-

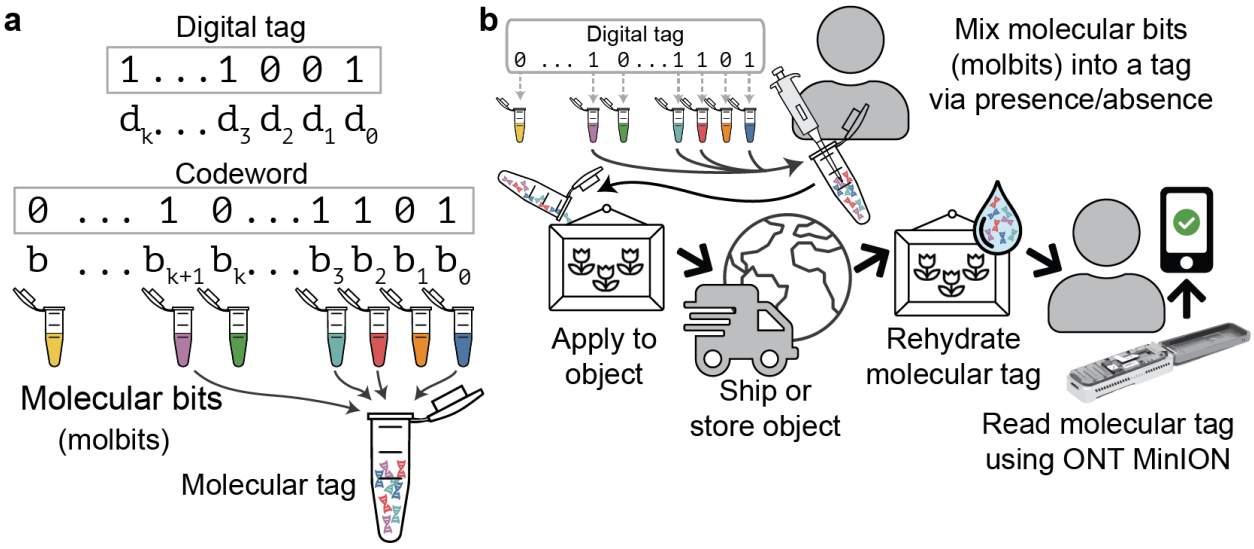

Figure 2.1: **Creating a molecular tag using the Porcupine molecular tagging system.** (a) Porcupine's encoding scheme. A digital tag is converted into a codeword to add additional bits for error correction. Each codeword bit is assigned to a unique molbit, where 1's and 0's are represented by the presence or absence of individual molbits in the molecular tag mixture. (b) A user first defines a digital tag as a binary 96-bit number, and pipettes 1-bits into the molecular tag. The tag is applied to an object, which is then shipped or stored. To read the tag, it is rehydrated and loaded directly onto Oxford Nanopore Technologies' MinION device. Software then decodes the molecular tag without knowledge of the original digital tag.

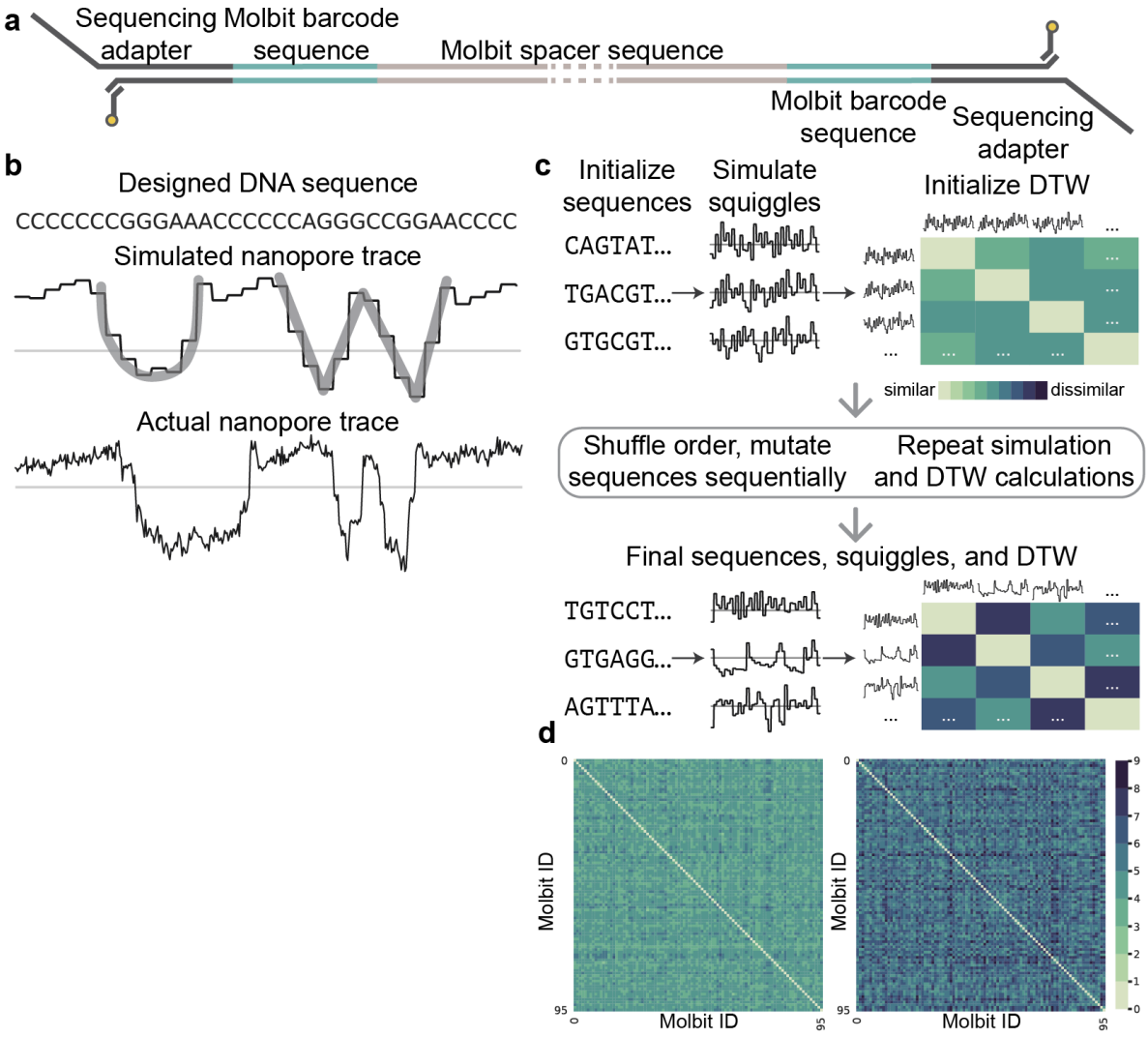

Figure 2.2: **Molbit design scheme.** (a) Molecular bit (molbit) structure. Each molbit consists of a molbit barcode sequence and a spacer sequence of a specific length. (b) The letters "UW" depicted visually as nanopore ionic current data (as opposed to encoded in the sequence contents). From top to bottom, the shown sequence was simulated using Scrappie and sequenced on the ONT MinION, demonstrating the use of simulations for designing arbitrary squiggles. (c) Evolutionary model workflow. Each round of evolution begins with a set of sequences, their simulated squiggles, and pairwise Dynamic Time Warping (DTW) distances. The order is randomized and sequences are mutated one at a time, verifying DTW improvement after each attempt. (d) Dynamic time warping (DTW) scores before (left) and after (right) 31 iterations of the evolutionary model. At the start, the minimum DTW score was 2.9 (mean 4.2 +/- 0.4), and after evolution the minimum was 4.2 (mean 5.8 +/- 0.8).

plexing barcodes. DeepBinner classifies raw signals from 12 of Oxford Nanopore Technologies' 96 commercially available barcodes, improving upon their previous sequence-based tool, Porechop [67]. Similarly, DeePlexiCon classifies four hand-designed DNA barcodes for multiplexing in direct RNA sequencing, for which barcodes were not previously commercially available [60]. While these tools enabled higher yields of demultiplexed reads, or in the case of RNA, made multiplexing possible, they would likely perform even better if the barcodes were designed to optimize raw signal separability. Other tools have also been developed for raw signal processing later in the sequencing pipeline, for example for genomic alignment [27].

Porcupine addresses limitations in prior DNA-based tagging methods by building on recent advances in nanopore-based DNA sequencing technologies and raw signal processing tools. The development of portable, real-time nanopore sequencing [30], together with new methods that simplify the modular assembly of pre-defined DNA sequences [21], creates additional opportunities for rapid writing and on-demand readout in low resource environments.

Porcupine is a molecular tagging system that uses synthetic DNA-based tags and nanopore-based readout. Porcupine encodes a 96-bit digital tag through the presence or absence of 96 predetermined DNA fragments with highly separable nanopore signals, which I call molecular bits (or molbits, previously coined by Cafferty et. al [12]), shown in Figure 2.1a. Although DNA is typically considered expensive for reading and writing, Porcupine lowers the cost by presynthesizing the DNA, which can then be mixed arbitrarily to create new molecular tags. Molecular tags are then read out quickly using a portable, low-cost sequencing device (Oxford Nanopore Technologies' MinION; Figure 2.1b). Typically, nanopore ionic current signals must first be converted back to a DNA sequence in a computationally expensive process called basecalling, but I classify molecular tags directly from the ionic current, forgoing basecalling. Error correction is also added to the tag to resolve decoding errors, similar to electronic message transmissions systems and recently used in other molecular applications [4]. Molbits are prepared for readout (sequencing) prior to tag application and can be stabilized by dehydration, an approach that extends tag shelf life, decreases decoding time, and reduces contamination from environmental DNA. Thus, creating tags requires the

set of pre-prepared molbits, sequencing adapter kit and its corresponding requirements (e.g. centrifuge, thermal cycler, and rotator mixer), and a dehydration method; and reading tags only requires the tag, nuclease-free water, and a MinION device. The result is a highly accurate real time tagging system that includes an approach to developing highly separable barcodes. These barcodes, and the methods I use to develop them, are extensible; they can be used both within Porcupine to tag physical objects and beyond this system for other molecule-level tagging needs like sample multiplexing for nanopore sequencing.

2.2 Molbit design

2.2.1 Introduction

To develop Porcupine, I first defined an individual molbit as a DNA strand that combines a unique barcode sequence (40 nt) with a longer DNA fragment selected from a set of sequence lengths that I pre-determined (Figure 2.2a). This modular approach dramatically increases the number of molbits without having to design a prohibitively large set of unique sequences. Given a set of designed molbit barcodes, using just two different fragment lengths doubles the number of available molbits without requiring additional design or synthesis. To make assembly of molbits simple and modular, I designed them to be compatible with Golden Gate Assembly, a convenient and scalable one-pot DNA assembly method that combines a TypeIIS restriction enzyme and ligase, by incorporating a short single-stranded overhang [21]. To increase classification accuracy and decrease computation time, I further optimized them to avoid basecalling.

For the molbit barcode, the objective was to produce a large set of sequences that could generate unique ionic current signatures ("squiggles") to promote unambiguous classification. I achieve this by using an evolutionary model to gradually make all squiggles more distinct from one another. For the molbit fragment lengths, the objective was to choose fragment lengths that produce uniquely identifiable signal lengths. Since the length of a DNA fragment is proportional to its nanopore signal length with some stochastic variation, I attempted to

choose lengths that are sufficiently far apart so that the distribution of signal lengths does not overlap. In this section, I describe the criteria and algorithms for designing molbits in greater detail.

2.2.2 Related work

Porcupine's overall system design was broadly inspired by a combination of punched cards, early DNA data storage, and digital tagging systems like QR codes, barcodes, and RFID tags. More specifically, the molecular bit (molbit) concept and design strategy builds upon several veins of existing research, including file identifiers for DNA data storage, barcoding for sample multiplexing, and raw signal manipulation algorithms.

Labeling DNA: Sample multiplexing and file identifiers

Sample multiplexing reduces high-throughput sequencing costs by pooling multiple samples in a single run. Unique barcode sequences are attached to the DNA (or RNA) strands in each sample in the pool so the reads belonging to each sample can be separated post-sequencing [54].

Similarly, most DNA data storage methods implement a form of barcoding, using primer sequences to distinguish files rather than biological samples. These primer sequences act as file indices for information retrieval. Bancroft et al. [6] designed DNA molecules to contain a unique primer sequence followed by data, such that the primer sequence itself does not directly contain information but instead provides an identification and retrieval method via PCR. Porcupine uses this sequence prefix approach for molbit design. It differs from both multiplexing and file identification by defining uniqueness based on nanopore signals rather than the sequence.

For nanopore sequencing specifically, ONT sells multiplexing barcode kits for DNA. Since ONT has not released any RNA barcoding kits despite supporting direct RNA sequencing on their platform, Smith et al. designed and published a set of 3 DNA barcodes that could be ligated to RNA strands and dereferenced using only the raw nanopore signal and a method

they call DeePlexiCon [60]. This project is similar to Porcupine in that it also seeks to design DNA barcodes and uses raw nanopore signals to identify them. However, it differs from Porcupine in two notable ways: (1) DeePlexiCon uses only 3 hand-designed barcodes, and (2) raw signals are classified as 2D images in pre-trained networks. (Classification is discussed in more detail in Section 2.3.)

Raw signal design

Evolutionary algorithms attempt to mimic evolutionary biology by introducing random modifications within a dataset, checking for fitness, and accepting or rejecting the changes [2]. I adopted this method to make the designed molbit sequences more unique and distinguishable.

Within this evolutionary model, Porcupine leverages ONT's Scrappie squiggler [53]. For a given DNA sequence, Scrappie simulates raw current ("squiggles") using a deep Convolutional Neural Network (CNN), predicting the mean, standard deviation, and "dwell time" (duration) for each step through the sequence. Without this, designing molbit barcodes would require extensive experimentation.

Another related method is Dynamic Time Warping (DTW). DTW aligns and measures the similarity of two time series, using dynamic programming to allow signal "stretching." The algorithm is identical to Needleman-Wunsch [49] (commonly used for sequence alignment), but instead of a cost function for matches, mismatches, and gaps, DTW uses distance (e.g., Euclidean) to compare two points. Since its complexity is $O(n^2)$, using DTW to compute similarity is often a bottleneck, and several accelerations exist including the UCR Suite [58]. I used DTW and the UCR Suite for Porcupine to compare Scrappie predictions.

2.2.3 Results

Designing highly separable molbits

To model the predicted ionic current signature for arbitrary DNA sequences, I used Scrappie squiggler, a tool that converts nucleotide sequences to ionic current via a convolutional

model. To demonstrate Scrappie's ability to accurately model real nanopore squiggles, I hand-designed a DNA sequence that appears as the letters "UW" in squiggle space (Figure 2.2b), with high visual similarity to the simulated squiggle (except for noise). Scrappie's output also let me compute the signal similarity of two sequences quantitatively using dynamic time warping (DTW) as the distance measure. I used this approach inside an evolutionary model designed to make barcodes as separable as possible (Figure 2.2c).

To produce a set of 96 orthogonal molbit barcode sequences, I initialized the evolutionary model using 96 random or pre-seeded starting sequences (see Methods). I perturbed each sequence independently in random order by mutating two adjacent nucleotides simultaneously at a random location. If the mutated sequence failed to improve the minimum and average DTW similarities between itself and all other sequences, I reversed the mutation and attempted again for the same sequence. I also restricted sequence similarity and free energy of the sequences to avoid labeling ambiguities and secondary structure (see Methods). Using this method, I began with a set of starting sequence that had a minimum DTW similarity of 2.9 and mean 4.2 +/- 0.4 and achieved a final minimum of 4.2 and mean 5.8 +/- 0.8 after 31 rounds of evolution (Figure 2.2d), representing a ∼40% improvement in both the minimum and mean. The final sequences can be found in Appendix Table A.1.

Using length as an additional encoding channel

Given the set of 96 designed molbit barcodes, I wanted to increase the number of available molbits without requiring additional barcode design or synthesis. To do this, I inserted a DNA fragment between the barcode regions as a spacer sequence, which can be set to different lengths as an additional encoding channel. Thus, since each molbit consists of the unique combination of a molbit barcode plus a specific spacer sequence length, adding another length effectively adds an additional 96 molbits. Length works as an additional encoding channel because even without basecalling, the length of nanopore signals can be easily distinguished; the signal length is roughly proportional to the DNA fragment length.

Ideally, each same-length sequence would produce a nanopore signal that is identical

in length; however, in the real world, there is some stochastic variation in signal length due to fluid dynamics, stalling, etc. The longer the sequence is, the more variation there is between resulting signal lengths. To avoid incorrect length classification, the overlap between signal length distributions must be minimized. Choosing appropriate sequence lengths is challenging without a theoretical understanding of the variability of the "dwell time," the amount of time each nucleotide spends in the sensitive region of the pore. In this implementation, I chose lengths of 400 and 1600 nucleotides as a proof of concept since they are more than sufficiently far apart, and I evaluate their performance in the next section on identifying molbits.

2.2.4 Methods

Brute force

An early attempt at designing molbits used a brute force approach, trading a shorter upfront design time for longer compute time. In this brute force approach, I generated all repetitive sequences of length 2-6, based on the idea that a CNN may be able to use the periodicity to separate barcodes. It is not computationally tractable to test all sequences of length 40; computation became unwieldy after just length 6 (with repetition to fill the full length of 40 nucleotides). Once the set of sequences was generated, each theoretical nanopore squiggle was simulated using Scrappie squiggler [53]. The DTW dissimilarity was computed between all pairs of squiggles. Hierarchical clustering [33] was applied, using an inverted version of the DTW matrix as the similarity measure and forming 96 clusters. One sequence was chosen at random from each cluster.

This approach was limited for a few reasons: it did not account for sequence similarity, causing issues for labeling training data; any benefits to discriminability potentially provided by the periodicity were outweighed by mediocre sequence-based labeling; and requiring sequences to be periodic was too restrictive and significantly limited the space of possible molbit barcodes. Instead, I moved to the evolutionary design in the following section.

Evolutionary design algorithm

To produce meaningfully separable squiggles, I used an evolutionary model. First, I initialized the 96 sequences. Fully random sequence initialization works fairly well; however, I chose to initialize using the output from the brute force approach, which produced slightly better results than random initialization after the molbits were further perturbed. A similar "warm start" could be achieved by initializing many random sequences, perhaps hundreds instead of just 96, and choosing a subset of these that are maximally distinct within this random set before beginning evolution. Starting sequences and their corresponding simulated nanopore squiggles are shown in Appendix Figure A.1.

During mutation, I placed constraints on the sequences to ensure that they can be easily synthesized, assembled, and measured using the ONT MinION. If a mutated sequence does not fulfill these constraints, the mutation is reversed and attempted again. There are two types of constraints: (1) those that affect only a single sequence (independent constraints), and (2) those that impact the relationship between one sequence and all others (dependent constraints). Independent constraints require each sequence to be within a range of allowed GC content (30-70% GC), have a maximum folding potential (-8 kcal/mol) as calculated using NUPACK's MFE utility [69], exclude the BsaI cut site sequence (GGTCTC), and have a maximum homopolymer length of five for A/T and four for C/G. Dependent constraints require a minimum sequence dissimilarity, calculated using a local variant of the Smith-Waterman (SW) algorithm [61] ($\leq$ 15 SW score; cost function +1 match, -1 mismatch, -8 gap); and a minimum squiggle dissimilarity, calculated by simulating the sequence's nanopore squiggle using the Scrappie squiggler and computing the dynamic time warping similarity [58] for all squiggles vs. the new squiggle.

At the start of each round of sequence evolution, sequence order is randomized. Each sequence is mutated sequentially in this random order. The mutation is introduced by simultaneously modifying two adjacent nucleotides, in a random location. If the new sequence fails to fulfill the preceding constraints, the algorithm undo the mutation and try again until a

maximum number of tries (100, arbitrarily), at which point it proceeds to the next sequence. Next, it recalculates sequence similarities with respect to the new candidate sequence. If any sequences are too close to the new sequence, the mutation is undone and reattempted. Next, the nanopore squiggle is simulated for the new sequence using Scrappie squiggler [53]. The algorithm recalculates the DTW similarity for all squiggles vs the new squiggle, and, if any squiggles are now too close, it undoes the mutation and tries again. If the new mutation improves both the minimum and the average DTW dissimilarity between all squiggles, it is accepted; if not, the mutation is reversed and reattempted. Evolution ends when the optimization begins bouncing between just two sequences. At this point, the process has produced a local minimum as the result of a series of random incremental improvements, so further improvements may be gained only by significantly perturbing these final sequences. I show final sequences and their corresponding simulated nanopore squiggles in Appendix Figure A.2.

2.2.5 Discussion

Porcupine successfully uses an evolutionary approach to design a set of 96 molbit barcodes. Here, DTW scores provide a theoretical measurement of nanopore signal uniqueness, but the molbits' effectiveness in a real system can only truly be measured after gathering experimental data and training a model to distinguish them. This is explored in the next sections.

The molbit design remains flexible for adding more bits simply by adding more insert lengths, increasing the barcode region length to allow even more variation between molbit barcodes, or by combining barcode regions serially.

These molbits can potentially also be used for other applications outside molecular tagging, including multiplexing, with further evaluation of their performance amidst random sequences instead of the spacer.

Contributions

My contributions include a broadly useful tool to design unique nanopore sequences, with bounds on signal and sequence characteristics, and a set of 96 unique sequences for the Porcupine molecular tagging system.

Future work

Future work for Porcupine could include developing a generative model for designing molbit barcodes. Although the evolutionary model worked well for 96 molbits, its runtime scales exponentially, creating an upper bound of a few hundred molbits in practice.

Even without moving to a generative model, incorporating noise into the evolutionary model could produce a more realistic view of the squiggle uniqueness. Scrappie predicts the mean current, which Porcupine currently uses, but it also predicts the standard deviation and a noise parameter modeled by an inverse Gaussian. Although this adds a factor of non-determinism in the model, it would be interesting to evaluate the impact of incorporating this additional information.

Finally, a frequently suggested goal for signal design is to create a tool that can design sequences to look like a target signal (e.g. UW in Figure 2.2b). This leans heavily into the visual aspects of nanopore signal design, where the objective may be to visually watermark a DNA sequence or be able to recognize signals by eye.

2.3 Identifying molbits

2.3.1 Introduction

After designing the molbits and acquiring a theoretical understanding of their separability, I tested their performance using real data, evaluating length discriminability and molbit identification separately. For length, I aimed to minimize the number of misclassified reads by finding a simple boundary between the two fragment lengths in the distribution of raw signal lengths. This is a much simpler task than identifying molbit barcodes in raw signal.

For molbit barcodes, the overall goal was to train a model to take raw signals as input and predict which molbit was present, and then measure how well the model performed on this prediction task for test data. To achieve this molbit identification, several challenges needed to be overcome including gathering training data (molbit labels) for the 96 unique DNA species, developing a model architecture and training it, and evaluating performance on reads that could not be labeled using sequences.

2.3.2 Related work

Raw signal classification

Mako [52] is a tool released by ONT that can classify arbitrary raw signals. Mako consists of a 5-layer CNN followed by a dense output layer, wrapped by software to use raw nanopore signal files (FAST5 files) natively. I used Mako directly in an early version of Porcupine, but ultimately moved on to custom networks due to accuracy and runtime concerns.

DeepBinner [67] also classifies raw signals directly through its signal-based DNA demultiplexing algorithm, since sequence-based demultiplexing often fails due to poor basecalling. DeepBinner uses a CNN-based approach to demultiplex 12 ONT barcodes chosen for their separability, and achieves a better performance overall compared to sequence-based methods. Porcupine differs from DeepBinner in the number of sequences that it can differentiate, and DeepBinner does not design any sequences.

As mentioned previously, DeePlexiCon [60] classifies DNA barcodes used for multiplexing in direct RNA sequencing. DeePlexiCon transforms 1D raw signals into 2D images using one of four different methods: Gramian Angular Field (2 versions), recurrence plots, or Markov Transition Fields. The images are then classified using a pre-trained deep residual network, ResNet V2, with 20 or 56 layers. Porcupine differs from DeePlexiCon in the number of available barcodes, and DeePlexiCon's barcodes are hand-generated.

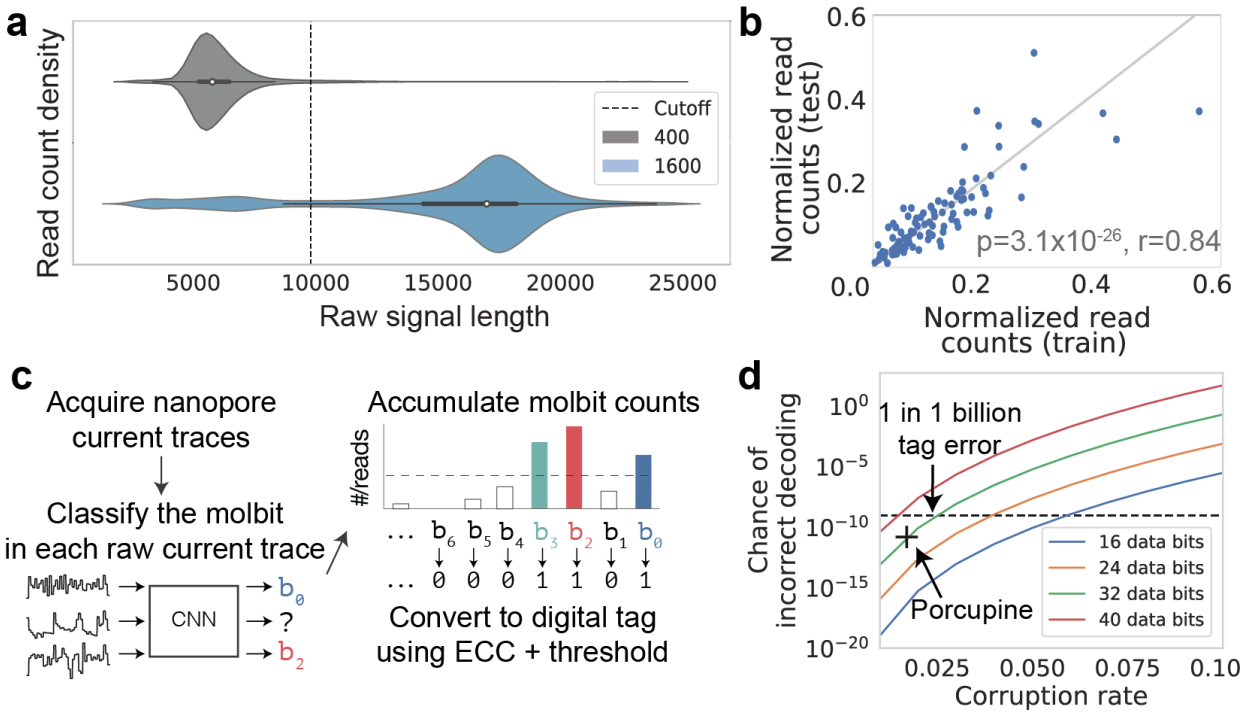

Figure 2.3: **Molbit classification and tag decoding results.** (a) Violin plots with embedded box-and-whiskers showing the distribution of raw nanopore signal lengths for DNA sequences of length 400 nt (n=354k reads) and 1600 nt (n=375k reads). The white dot represents the median, and the thick center line shows the quartiles. The dashed vertical line shows the cutoff used when calling reads as 400mers or 1600mers. (b) Correlation of read counts for each molbit, demonstrating consistency in molbit occurrences between training and testing runs (two-sided Pearson correlation, $p = 3.1 \times 10^{-26}$, $r = 0.84$). Counts were first normalized within each run and normalized again after combining runs for either training or testing. (c) Tag decoding workflow, with error correcting codes (ECC). After acquiring nanopore current traces from a standard sequencing run, the molbit in each trace is identified using the CNN (confidence ≥ 0.9). Successfully identified molbits are accumulated and converted into binary using a threshold for presence. This threshold is varied as error correction is carried out multiple times, accepting the binary digital tag that has the fewest differences from the received codeword. (d) Chance of incorrect tag decoding as a function of the bit corruption rate and number of data bits. This chance increases exponentially as the corruption rate and number of data bits increase linearly. The dashed line represents the goal of 1 in 1 billion tags incorrect, and the "+" marks Porcupine's chance of incorrect decoding.

2.3.3 Results

Classifying molbit length

I tested Porcupine's concept of varying the molbit length by using two spacer lengths (400 and 1600 nucleotides extracted from an arbitrary portion of plasmid pCDB180; see Methods) and found that simple signal length binning was sufficient for decoding. The median signal length for the 400 and 1600 nt strands was 5768 and 16968, respectively, and a cutoff at 9800 gave 91% accuracy, where most errors were caused by long strands misidentified as short strands (Figure 2.3a). The threshold for determining length was chosen by finding the point that maximized the number of correctly assigned reads when comparing separate runs containing 400 or 1600 bp long sequences.

For the 1600 bp reads that fell below this threshold, I examined their sequence contents for evidence of fragmentation. The majority of the short strands in the 1600 bp sample can be explained by fragmentation. After aligning all basecalled sequences to the insert fragment sequence using BWA-MEM [42], I examined reads that were shorter than the signal length threshold (i.e., those mislabeled as 400 bp). I found that 28% of these reads were truncated, meaning they aligned well to the beginning of the insert but terminated prematurely at various lengths. Another 59% mapped the majority of the read to a random portion of the reference, indicating other forms of fragmentation. Since these reads did not align to the beginning of the read, they do not contain the molbit barcode sequence and effectively provide no information at all. The remaining reads were simply poor quality reads that did not basecall and/or map well. To mitigate this, a length purification step could be added, increasing preparation time to provide a higher quality readout.

For remaining experiments, I focused on a single spacer length to reduce experimental complexity and devote more resources to the more conceptually interesting molbit design.

Classifying molbits directly from raw nanopore signals

Next I gathered training data and developed a convolutional neural network (CNN) model to accurately classify molbits directly from real raw nanopore data (see Appendix Figure A.3 for the model architecture). Direct signal classification is preferred over sequence-based methods primarily because classification is an inherently simpler problem than full sequence decoding, where the problem can be reduced to distinguishing between 96 distinct signals rather than reproducing the exact series of underlying nucleotides generating an arbitrary signal. This enables simpler model architectures which typically come with lower computational and training data requirements. In my model, molbits do not need to be segmented or otherwise isolated from the raw data, rather, the CNN uses only the first portion of the each molbit since the molbit barcode is located at the beginning of the strand.

I gathered training data by dividing the 96 molbit barcodes into 6 sequencing sets, with each molbit appearing once; optimizing the sets for maximum sequence separability to improve labeling (see Methods); and running each set on the MinION. I assigned labels to each molbit read using traditional basecalling methods and a modified semilocal Smith-Waterman sequence alignment [61], using only high-confidence alignments (see Methods). For test data, I divided the 96 molbits into 2 sequencing sets, with each molbit appearing once, and gathered the data as described for training.

The model was trained for 108 iterations, with a final training, validation, and test accuracy of 99.9%, 97.7% and 96.9%, respectively, compared to sequence-derived labels. However, in real world decoding, all reads are classified, not just those that pass basecalling and sequence alignment. I found that the CNN was consistently able to confidently classify a larger portion of the reads (97% of reads in the test set) than basecalling plus alignment (75% of reads in the test set), revealing a large portion of reads that could not be easily validated. I reasoned that if the occurrence of each molbit was proportional between basecalling and the CNN, that the CNN was likely not making spurious calls, but was perhaps performing better on the raw signal data. Read counts correlate extremely well between the two methods (see

Appendix Figure A.4), revealing another advantage of direct signal classification, making use of ∼30% more data than sequencing alone. As a result, "accuracy" only reflects the model accuracy and does not necessarily measure each molbit's error rate or the overall chance of decoding the tag incorrectly.

2.3.4 Methods

Experimental methods

This section was written and carried out by Karen Zhang and Jeff Nivala.

We purchased forward and reverse strands of the 40 nt unique barcode sequences from Integrated DNA Technologies (IDT). Reverse strands contained a 5' GAGT overhang and a 3' dA-tail. We annealed forward and reverse strands by mixing them equimolar in 0.5M PBS, boiling them at 94C for two minutes and then allowing them to cool at room temperature. To generate the insert spacer, we amplified an arbitrary 400 nt portion of plasmid pCDB180 [https://www.addgene.org/80677/] by PCR, using primers we designed to add BsaI cut sites to the ends of the amplified product. The primer sequences are available in Appendix Table A.2.

To assemble the molbits, 600 ng of the desired annealed barcodes (at equimolar concentrations) and 600 ng of the spacer were ligated together using NEB's Golden Gate Assembly Kit. We prepared molbits for sequencing using ONT's Ligation Sequencing Kit (SQK-LSK109) following the kit protocol (we skipped the "DNA repair and end-prep" step because molbits were already dA-tailed) and ONT's Flow Cell Priming Kit (EXP-FLP001). Molbits were sequenced on a R9.4.1 MinION flow cell with bulk FAST5 raw data collection enabled on MinKNOW.

We dehydrated tags after nanopore adapter ligation by mixing the molbits with 1% trehalose dihydrate solution and lyophilizing the sample. To sequence, we rehydrated the lyophilized sample in nuclease-free water and carried out the sequencing run as described above.

Molbit classification model

I identify individual molbits using a classification model, which takes raw nanopore signals as input and outputs the molbit ID with an associated confidence. The model consists of a 5-layer CNN, followed by two fully connected layers with 50% dropout, and a final fully connected layer with softmax as the output layer. Each of the 5 CNN layers is identically structured and includes a 1D convolutional layer with Relu activation, average pooling, and then batch normalization. I show a diagram of the model with exact parameters for each layer (e.g., kernel size) in Appendix Figure A.3.

Ideally, I would build a training dataset by sequencing each of the 96 molbits separately, guaranteeing that each read can be perfectly labeled. However, due to cost, I instead divided the 96 molbits into 6 runs of 16 molbits each. I constructed these sets to have a high predicted distance between the molbits within a set, meaning the most similar and easily confused molbits were not sequenced together for training data acquisition.

I assigned training labels using basecalling (Guppy version 3.2.2 with GPU acceleration) followed by Smith-Waterman (SW) sequence alignment (cost function: +1 match, -1 mismatch, -8 gap) against the full set of 96 molbits. I considered any SW score ≥ 15 to be a well-aligned match. As a quality measure, I also examined how many of these reads were labeled with one of the 16 possible molbits. An average of 98.7% +/- 2.1% of well-aligned reads belonged to the true set of molbits across all training runs, indicating high quality labels.

After labeling the training data, I balanced the dataset by allowing a maximum of 6000 read occurrences for each molbit, with a total of 274,667 reads used for training. To preprocess the raw signal, I rescaled the signal using a Median Absolute Deviation method modified from Oxford Nanopore Technologies' Mako classification tool, trimmed the signal to remove the variable-length stalled signal characteristic to the beginning of sequencing reads, and finally truncated the signal to the first 3000 data points. On average, the molbit barcode comprises $\sim$10-15% of these 3000 observations. Due to stochastic variation in the

dwell time of each nucleotide in the pore, there is some stretch and stalling in the signal, especially near the beginning of the read. This can cause considerable variation in both the position and length of the molbit barcode. This highlights the flexibility of the CNN, which allows me to be liberal with trimming since finding the exact end point of the barcode is not required.

I split the training data 85%/15% to produce training and validation sets and trained the model for 109 iterations, with a final maximum training accuracy of 99.94% and validation accuracy of 97.78%. Confusion matrices for training and validating the final model are shown in Appendix Figure A.5a.

Testing data was acquired and labeled in the same manner as the training data, using 2 new sequencing runs, each containing a unique half of the molbits. Performance on these test sets was 98.1% and 95.7% for labeled data (Appendix Figure A.5b). The flowcell from test set 1 was washed and reused for test set 2, which potentially contributed to a small portion of the errors present due to DNA carryover between the runs. I show read counts for these two test runs in Appendix Figure A.6, noting when a molbit was possibly present from the previous run.

2.3.5 Discussion

Porcupine's raw signal molbit classification algorithm has sufficiently high accuracy (∼97%) to consider it a reliable method for identifying molbits. Using raw signals to identify molbits had an additional, unexpected benefit: the ability to use nearly all reads, not just a set of the highest quality reads. In a typical DNA sequencing workflow, data is basecalled, then quality filtered, then aligned and quality filtered again. This was the case for the labeled data; I only trained on the highest quality reads, representing about 70% of the reads. However, this sequence-based quality filtering is unavailable when examining only raw signals. I showed that molbit identification was still successful even when using all reads, which potentially allows tags to be decoded with 30% less data or could increase confidence with the same duration of sequencing.

From a computational perspective, I note that basecalling is getting faster and more reliable; however, raw signal classification is fundamentally a simpler problem than basecalling. Such classifiers can be trained with comparatively minimal data and expanded to non-traditional sensing; for example, my molbit approach could be easily extended to include non-standard bases for additional security, but standard basecalling would not be possible without extensive amounts of training data.

Similarly, future updates to nanopore sequencing chemistry would likely require retraining the classification model, made feasible by the low training data requirements. Even if a new chemistry modifies the raw signal to the extent that they are no longer classifiable via raw signal, tags would still be readable by directly sequencing the DNA.

Contributions

My contributions include a strategy to label molbits using sequences, a model to classify which molbit is present in each read, and an approach to generating and balancing training and test data.

Future work

Currently, the model uses quite a large signal window (3k observations), such that the barcode only comprises 10% of the window. This is because the position of the barcode can vary significantly within the window. Further exploration of the barcode position may be useful in order to reduce this signal window, and therefore possibly also reduce the size and complexity of the model. If, perhaps, only 5% of the molbit barcodes appear outside the first 15% of the current window, it may be acceptable to sacrifice that 5% or find a better strategy for trimming the reads to place the barcode at the very beginning.

Additionally, for some applications beyond Porcupine, for example, trimming barcodes from multiplexing applications, it would be useful to identify where the barcode starts and ends rather than just labeling its existence.

Further evaluation could be carried out to test the false positive rate of identifying barcodes by attempting to classify invalid, poor quality, or random data (arbitrary sequences). Such testing would be necessary to validate whether Porcupine's molbit barcodes can be used for multiplexing samples.

2.4 Writing and reading tags

2.4.1 Introduction

Given a set of molbits and a way to read them, Porcupine's final demonstration includes writing real information into tags and reading it back. At this stage, accuracy and speed are top concerns for producing a usable system. Even if the per-bit error rate is just 1%, the error is too high for a reliable system without error correction. Likewise, the time required to decode a tag is typically limited by the amount of time needed to gather the data, so I explore time to decode here as well.

To encode information in Porcupine, I begin with a digital tag of 0's and 1's. An operation is performed on this digital tag to add error correction (see Methods). This converts the digital tag into a codeword, which is the length of the digital tag plus some additional parity bits for error correction (Figure 2.1). After converting to a codeword, each bit position in the codeword is assigned a specific molbit; if the bit is set to 1, it is added to the molecular tag mixture, and if it is set to 0, it is left out. Once all 1-molbits have been added to the molecular tag, the tag mixture is complete. At this point, it can be prepared for sequencing and dehydrated for longer term storage.

The error correction is carefully tuned to the expected per-bit error rate of the tag, which is influenced by a variation in read counts. Ideally, all 0-bits have no reads, and all 1-bits have many reads, but this is not the case, and some bits are flipped when I attempt to binarize the read counts. In this section, I describe Porcupine's implementation of error correction as a random generator matrix. However, since Porcupine simply provides a set of bits, the exact error correction implementation can be chosen based on the application and its error

tolerance.

Here I demonstrate the method and performance of applying error correction, end-to-end decoding of the acronym "MISL", decoding time, and analysis of dehydrated tag longevity.

2.4.2 Related work

Error correction

Error correction is often used in digital systems to recover from bit errors. ECCs reduce the possibility of unrecoverable data despite the presence of errors by adding redundancy or projecting data into a larger space with greater separability, allowing more bits to be flipped before the data is decoded incorrectly. Much like digital systems, any form of practical DNA data storage, requires error correction so that the data can be recovered even in the context of errors. Currently, errors in DNA-based storage systems occur at a substantially higher rate than in digital systems, since errors can be introduced via DNA synthesis, copying, storing, or sequencing, all of which have higher error rates than bit flips in silicon.

DNA data storage often uses rotating codes or forward error correction [9, 10]; however, Porcupine's sparse encoding can be more closely compared to a series of bits than to larger-scale DNA data storage. Many of the most widely used codes for this context are linear codes [45], often using random binary matrices to transform messages into codewords using matrix multiplication, so we chose random linear codes instead.

2.4.3 Results

Encoding data in molecular tags with error correction

Next, I composed actual molecular tags. I assigned each molbit a unique position in a binary tag, allowing each 1 or 0 to represent the presence or absence of a specific molbit over the course of a single tag sequencing (decoding) run. To determine presence or absence, I used CNN-classified read counts for each molbit. Ideally, 0 bits would have zero reads, and 1 bits would have nonzero reads. However: two factors complicated setting this threshold to de-

termine bit presence: (1) nonzero read counts for molbits not present in an experiment, and (2) significant variations in counts for molbits present in an experiment, which was found to be up to 20-30x in the training and test sets (see Appendix Figure A.6 and Supplementary Methods in Appendix Section A.1). Fortunately, these variations were consistent when comparing the ratios of molbit counts in the training and test data (Figure 2.3b). I accounted for this variation by scaling all read counts by a fixed vector based on these ratios. Thresholding and scaling read counts reduced the per-bit error rate from 2.9 +/- 1.8% to 1.7 +/- 1.6%, a 42% reduction.

Since a reliable tagging system should have a very low chance of incorrect decoding (e.g., 1 in 1 billion), I decided to further reduce the overall tag decoding error rate by including error correcting codes (ECCs) as part of the tag design (Figure 2.3c). Melissa Queen contributed most of this aspect of the project. The simplest non-ECC method for encoding information in these tags is a naive 1:1 mapping between digital bits and molbits; however, with this method, even a single bit error makes the tag unrecoverable (i.e., produces an incorrect decoding). In Porcupine, bits are set to 1 or 0 using a threshold for presence or absence on the read counts, meaning that any 0-bits above this threshold are instead flipped to 1, and vice versa. ECCs reduce the possibility of unrecoverable tags despite the relatively high per-bit error rate by reserving a smaller number of bits for the digital message and creating a codeword by projecting this message into a larger space with greater separability. This allows more bits to be flipped before the message is decoded incorrectly. To encode the digital message, we simply multiply the message by a binary matrix of random numbers, known as a random generator matrix (See Methods). The number of bits reserved for the ECC depends on the application's error tolerance and the per-bit error rate (Figure 2.3d). As the error rate increases, the chance of incorrect decoding increases exponentially. Thus, the number of bits for the message must be chosen carefully. I chose a message size of 32 bits, which at an error rate of 1.7% produces a 1.6×10^{-11} chance of incorrect decoding and permits $\sim$4.2 billion total unique tags, with correct decoding guaranteed at or below 9 bit errors.

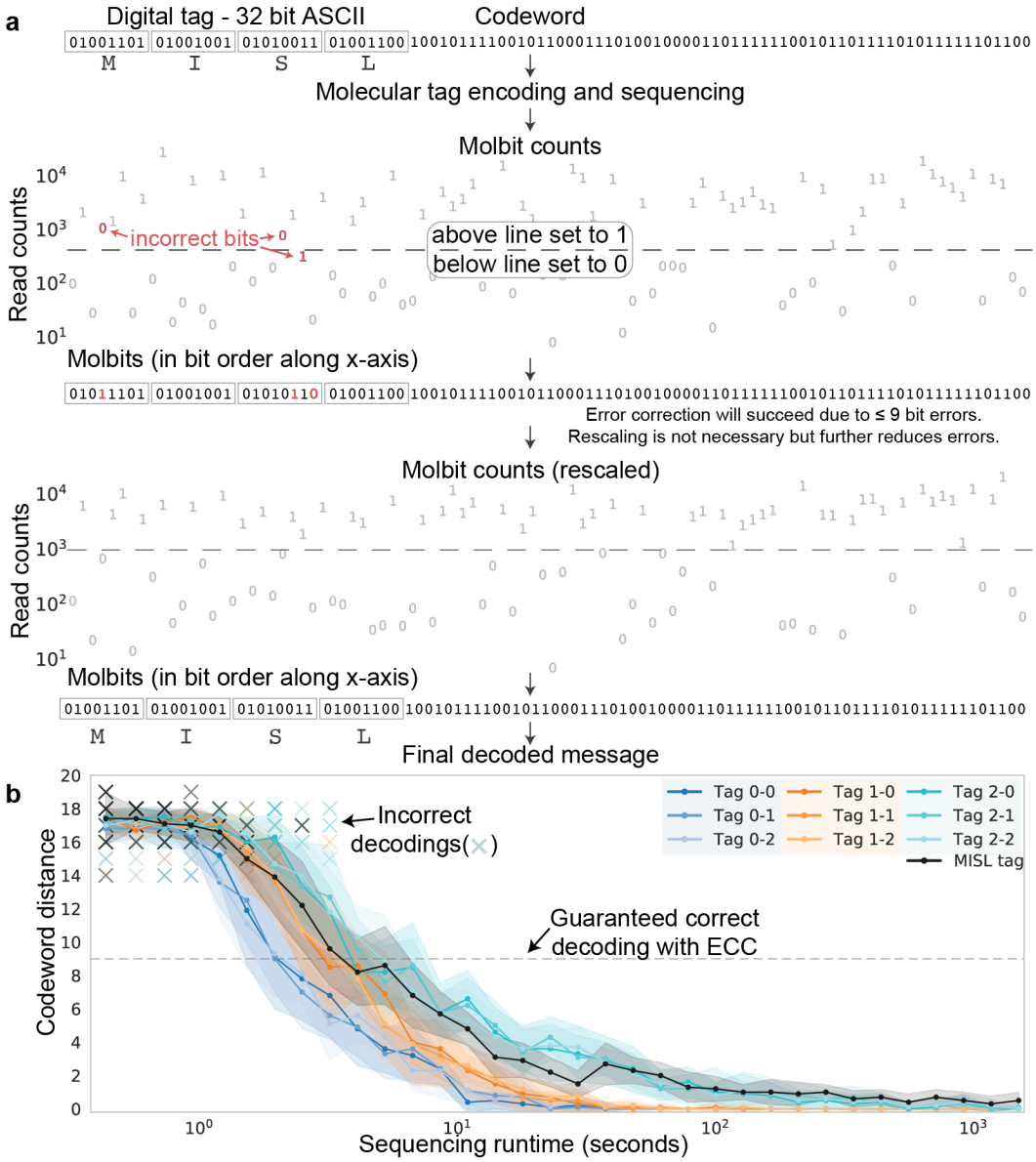

Figure 2.4: **End-to-end data flow for the acronym "MISL".** (a) Encoding began by converting "MISL" to ASCII (32 bits), then adding error correction to produce a 96-bit codeword. After converting the codeword to a molecular tag, it was sequenced for 35 minutes. Molbit reads were classified and counted, then binarized (dashed line shows the best threshold). Rescaling the counts according to known read count variation reduced these errors, in this case eliminating errors entirely. (b) Minimum tag decoding distance as a function of sequencing runtime. Each color group (blue, orange, aqua, black) represents a unique sequencing run, and each tag is a remapping of a valid codeword to molbit labels. Incorrect decodings are marked with an X. At each time point, reads were sampled (10k reads/min, n=10); the S.D. of this sampling is shown as the shaded area.

End-to-end encoding and decoding

Next, as a proof-of-principle for the tagging system, I demonstrated end-to-end tag encoding and decoding of the acronym "MISL," short for "Molecular Information Systems Lab" (Figure 2.4a). I began by encoding MISL into binary using ASCII, which uses 8 bits for each character, for a total of 32 bits, and multiplied this bit vector by the generator matrix to produce a 96-bit codeword. The molecular tag was then prepared as explained previously, with one modification for lab efficiency (see Methods). Once the molecular tag was assembled, it was prepared for sequencing and read out using an ONT MinION. I then identified the molbits from the raw data using the trained CNN classifier, accumulated a count for each molbit, and rescaled these counts to accommodate systematic read count variances. I then decoded the tag as described for the ECC by binarizing the counts using a sliding read count threshold to determine presence and absence, then finding the distance between the binarized counts and the nearest valid codeword at each read count threshold. When the edit distance is low enough to guarantee a unique correct decoding (this distance is 9 for this ECC), the molecular tag decoding is complete. The earliest correct decoding occurred less than 7 seconds after sample loading (109 molbit strands observed), demonstrating reliable encoding and decoding of a 32 bit message in only a few seconds using a portable sequencing instrument.

Measuring decoding time

Finally, to robustly estimate correct decoding time under these conditions, I simulated new tags using the original two test runs and one additional random tag by reassigning molbit labels within the tags (e.g., molbit 0 in the original tag is given a new label, molbit 42, in the simulated tag). This simulation impacts the error rate both positively and negatively due to the nonuniform distribution of distances between codewords in the ECC. To achieve this remapping, I generated new codewords with the same number of 1-bits as the original tag datasets. The original observed molbits were then randomly assigned to the new bit

ordering: 1-molbits in the original tag were assigned new 1-molbit labels in the synthetic tag. After creating these new codewords, I sampled reads without replacement, accumulated a count for each molbit, and decoded as above, with 10 repetitions per run time, where run time was estimated by the average of 10k reads per minute. I found that some tags could be successfully decoded with only a few seconds of sequencing data, and all tags after just 10-15 seconds (Figure 2.4b).

Dehydrating tags

Tags can be prepared for sequencing at the time of tag creation and then shelf-stabilized by dehydration, further reducing readout time at the cost of increasing "writing" time through sequencing preparation. Pre-preparation appears to have minimal risk, supported by correct tag decoding and minimal changes to sequencing yield, Q-score, and basecalled sequence length when sequencing a freshly prepared and dehydrated tags (at 0 and 4 weeks from dehydration (see Appendix Figure A.7).

Preparing the tags in this manner has several impacts on tag reading and overall longevity. First, adding the sequencing adapter immediately after tag creation prevents environmental contamination throughout the life of the tag. Since DNA requires a sequencing adapter to move sufficiently slowly and uniformly through the nanopore, other non-tag DNA that comes into contact with the tag after creation simply cannot be sequenced since it lacks the adapter. Second, it reduces readout time at a cost of increasing writing time. Third, as shown in Appendix Figure A.7, dehydrated tags appear to have an extended lifespan, at least four weeks with no indication of decreased read quality. Although I did not do extensive investigation in this area, there are two main points of "failure" (degradation): the DNA degrading, and the sequencing adapter detaching from the DNA, especially due to the difference in size between the adapter and DNA. Dehydrated DNA degradation has been studied to an extent in the context of DNA data storage [51], and sequencing adapter attachment and longevity remains an open question.

2.4.4 Methods

Error correction using random linear codes

Error correction was implemented by Melissa Queen.

To improve the system's resilience we overlay an error correcting code. The error correcting code (ECC) maps the original messages into a higher dimensional space that provides a greater distance between any two messages. In this higher dimensional space a message must accumulate many errors before it is decoded incorrectly, allowing us to reduce the chance of incorrect decoding for a given fixed error rate.

The error correcting code protocol has two stages: encoding and decoding. We use a random linear code, which consists of a fixed, randomly chosen $n \times 96$ generator matrix that encodes an n-bit message via a simple and efficient matrix multiplication. The resulting 96-molbit codeword may accumulate errors during creation, storage, or retrieval. However, we can decode the 96-molbit codeword back to the n-bit message with high probability using brute-force nearest neighbor decoding.

We also computationally examined whether limiting the set of molbits to only the most performant would improve overall tag accuracy. Unfortunately, as the total number of molbits drops, they must each become more reliable in order to maintain the desired chance of incorrect decoding of 10^{-9} (Appendix Figure A.8). Another way to think about this tradeoff is to consider that even unreliable molbits are still conveying useful information that the ECC is able to use to deduce the corrected message. For example, a collection of 96 molbits can reliably encode a 32-bit message even if each molbit is wrong 3% of the time on average. But if we reduce the pool slightly to just 90 molbits, then they can only be wrong 2% of the time on average. At 64 molbits, a dramatic reduction in the corruption rate would be required, down to the order of 0.15%.

2.4.5 Discussion

This demonstration of end-to-end encoding and decoding establishes Porcupine as a functional molecular tagging system, complete with error correction and tags that encode real digital information. Successful decoding requires less than 10 seconds of nanopore sequencing data on a MinION, making it possible to read Porcupine tags on disposable flowcells like the Flongle with one quarter of the pores of a MinION without prohibitively increasing readout time.

Here we explored a simple implementation of error correction to reduce a relatively high error rate (1.7%) to 1 in 1 billion chance of incorrect decoding. Although decoding remains robust to bit errors, more stability in read counts would increase the amount of information encoded by reducing the number of bits required for error correction. Without resolving this variation, the number of bits can still be substantially expanded by adding more insert lengths to take advantage of the modular system design.

I also demonstrate that tags can be prepared for sequencing at the time of tag creation and then shelf-stabilized by dehydration, further reducing readout time at the cost of increasing "writing" time through sequencing preparation. Pre-preparation appears to have minimal risk, supported by correct tag decoding and minimal changes to sequencing yield, Q-score, and basecalled sequence length when sequencing a freshly prepared and dehydrated tags (at 0 and 4 weeks from dehydration (see Appendix Figure A.7).

Contributions

My contributions include a method for binarizing read counts into a bit string, encoding real data into a molecular tag with error correction (credit goes to Melissa Queen for the ECC design, its implementation, and the decoder), and analysis of time required for decoding based on simulated tags.

Future work

In the future, a more sophisticated ECC would be worth pursuing. There has already been some interest in this since the random generator matrix is a fairly naive approach. This could result in an increase in the number of encoding bits, decrease in error, or both. This could also take advantage of an asymmetric error profile, in which bits are not uniformly flipped to either 0 or 1, but one direction is more frequent than the other (in this case, 0-to-1 flips seem more likely).

Another avenue for improvement is automating tag mixing. Pipetting an average of 48 unique molbits per tag is a labor-intensive process (which Karen Zhang graciously carried out several times for this project). Lab automation robots that can work with 96-well plates would interface well with Porcupine's 96 bits. PurpleDrop [62] could also work well with Porcupine, with some modifications. PurpleDrop is an inexpensive digital microfluidic device (also designed in the MISL lab). The main limitation of this device is the restricted number of input and output channels, but work is currently being done to explore alternative mixing options.

Additional experiments could be carried out regarding dehydration, including longevity, surface adhesion to a variety of surfaces, and the dehydration method. We experimented with lyophilization in this implementation, but it would be ideal to dry the tags at room temperature with average humidity. Applying tags to an actual object was also a missed opportunity for the paper. It would be exciting to experiment with this, and make Porcupine's applications more concrete.

2.5 Summary

In summary, Porcupine offers a method for molecular tagging based on the presence or absence of synthetic DNA sequences that generate unique nanopore raw current traces. By directly manipulating segments of nanopore raw current and keeping unique sequences short, Porcupine reduces synthesis costs for the end user and produces visually unique nanopore

current traces, enabling high accuracy decoding. The speedy decoding time means this system can decode using newer technologies such as the Flongle, a cheaper, single-use flowcell produced by ONT with a quarter of the pores of the MinION in as little as 1-3 minutes. In addition, tags can be prepared for sequencing at the time of tag creation and then shelf-stabilized by dehydration, further reducing readout time at the cost of increasing writing time through sequencing preparation.

While encoding a single molbit per strand has advantages, including the ability to combine pre-prepared molbits into a tag with few technical requirements, the information density is low, limiting the amount of information that can be stored. Instead, molbits could potentially be strung together sequentially in one long fragment to achieve a higher information density. I explore this in another project described in the next chapter.

A provisional patent has been filed by the University of Washington, 16/879,214, "MOLECULAR TAGGING SYSTEM WITH NANOPORE-ORTHOGONAL DNA BARCODES", on May 20, 2020, covering aspects of this work including barcode design and readout. Inventors include Kathryn Doroschak (me), Luis Ceze, and Jeff Nivala.

Chapter 3

DATA STORAGE USING SEQUENTIALLY ENCODED MOLECULAR BITS

3.1 Introduction

DNA data storage and molecular tagging share a common goal of encoding information in DNA, but they differ in how each prioritizes cost, information volume (storage size, in bits), and reading and writing speeds. DNA data storage is known for its goal to encode a large volume of information in very little physical space (on the scale of many gigabytes, e.g. archival storage) [14], although cost and speed are important as well. In contrast, molecular tags must prioritize costs and speed, so traditional DNA data storage approaches are not appropriate. The current most cost effective approach to traditional DNA data storage uses array-based synthesis [40]. This method produces high volumes of unique DNA strands but is relatively expensive (on the order of $0.10 per base) and has a high latency between ordering and receiving the data – even if the process only takes days, it still cannot be completed at the time the information is generated. This limits the ability to write low volumes of data efficiently, and there is currently no practical way to encode information without ordering sequences from a third party or purchasing an expensive DNA synthesizer device.

The Big Bits pilot project presents an alternative method for DNA data storage, borrowing from Porcupine's modular approach. Instead of using presence or absence of molecular bits to encode 0's and 1's (Figure 3.1a), Big Bits uses multiple molbits in a single DNA fragment to encode 0's and 1's sequentially (Figure 3.1b). Nanopore ionic current traces are still used for readout. This approach allows on-demand writing and is explored as a proof of concept in this chapter. As a result, Big Bits provides a more flexible method to encode and read data which would allow ordinary, non-specialized labs to write arbitrary data in

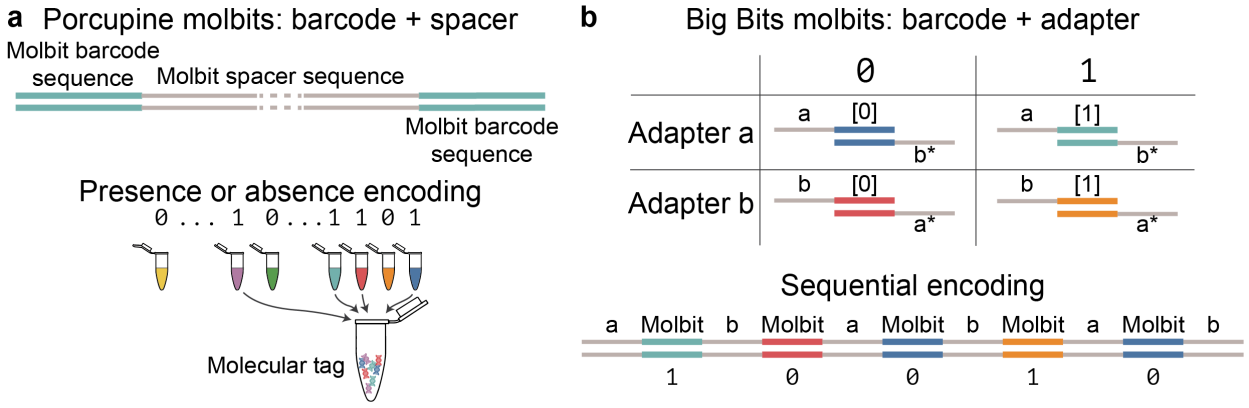

Figure 3.1: **Structure of molbit strands in Porcupine (a) and Big Bits (b).** (a) Porcupine contains one bit per DNA strand, the unique combination of a specific molbit barcode sequence and a specific spacer sequence length. Molecular tags are then created using presence or absence; if a bit is set to 1, it is added to the molecular tag mixture, and if it is set to 0, it is omitted. (b) Big Bits contains multiple bits per DNA strand. Molecular tags are created by appending molbits one after another into a single strand, where the bit position is important. Thus, each molbit has a single stranded adapter in order to attach to the next molbit, plus a unique molbit barcode sequence.

DNA, reduce costs and latency by bringing writing in-house, and enable near-instantaneous readout using commercial nanopore devices. This work is presented in three parts: molbit structure and design, acquiring and labeling training data, and reading bit strings.

This project was carried out during an internship with Microsoft Research in Summer 2020.

3.2 Molecular bit design

3.2.1 Introduction

Molbit design for both Big Bits and Porcupine is guided by the same overall goal – to create DNA sequences that generate highly separable nanopore current traces. However, Big Bits' alternate scheme for encoding multiple molbits within a single DNA fragment means that Big Bits requires fewer unique molbits, but significantly longer strands.

Transforming a bit string into a molbit strand is much more conceptually straightforward

in Big Bits than Porcupine. Porcupine and Big Bits both begin with digital data in the form of a bit string of 0's and 1's. In Big Bits, the entire digital bit string is encoded in each strand (Figure 3.1b), rather than Porcupine's single bit per strand. Just as each bit position in the digital bit string is set to either 0 or 1, each position in a molbit strand contains either a 0-molbit or a 1-molbit. The 0- and 1-molbits are a pair of sequences which produce highly separable nanopore signals. To attach molbits to one another in a strand, molbits contain a different single-stranded overhang on each end of the fragment. These adapters are much longer than those in Porcupine so they can hybridize to form long strands without needing to ligate after every addition. Like Porcupine, error correction would be added before converting digital data to molecular data in a real system, however, the error rates must first be better understood and characterized. Error correction is thus out of scope for this pilot project. Big Bits also uses raw signals to distinguish bits, although decoding several molbits is much more challenging than classifying just one bit per read in Porcupine.

Molbit design occurs on a much smaller scale in Big Bits than Porcupine due to the greatly reduced number of bits. In Porcupine, the information capacity is finite and limited by the number of unique molbit barcodes and distinct molbit lengths; the only way to increase this is to design more molbits. This is no longer true for Big Bits' encoding scheme. Big Bits can write arbitrarily long molbit strands with very few unique molbits because of the position-based encoding. The maximum amount of digital information is instead limited by the maximum length of the molbit strand – a physical limitation of long DNA fragments with the potential for accumulating errors with each molbit addition.

Therefore, one goal of Big Bits is to encode as many bits as possible in each fragment. A higher density can be achieved by shortening the length of each molbit sequence, however, reducing the molbit sequence length decreases the maximum attainable squiggle diversity. Here I explore the balance between length and uniqueness. Length poses additional challenges for finding a unique set even though fewer unique sequences are required.

In this proof of concept for molbit design, I created multiple sets of molbits. This includes both the molbit itself (the part of the sequence that encodes the 0 or 1) and adapters to

join them together via hybridization. I explored two different sequence lengths and varying numbers of adapters in each set, and used an evolutionary algorithm to design the short bit sequences.

3.2.2 Related work

Molbit design

The related work for Big Bits' molbit design is identical to the related work for Porcupine, which can be found in Section 2.2.2.

DNA data storage

Goals for DNA data storage typically include maximizing information density, minimizing the coverage/physical redundancy required for recovery, and minimizing costs (currently dominated by DNA synthesis), as described in detail by Ceze et al. [14] in a review. Logical densities for in vitro DNA data storage range from 0.19 bits per nucleotide [24] to 1.94 [3], or when considering only the payload, 0.29 [24] to 3.37 [16]. Big Bits is not comparable in information density to any of the approaches described in this review by at least one order of magnitude. For comparison, my proposed implementations for Big Bits could be reasonably measured instead by nucleotides per bit instead of bits per nucleotide.

Nanopores have also been used for readout in DNA data storage [47]. Lopez et al. increased the total sequencing and decoding capacity by assembling long reads (via Gibson assembly) to increase throughput and a random access approach.

3.2.3 Results

Designing molbit sets

One objective of Big Bits is to maximize information density by encoding as many bits as possible in each DNA fragment. Since the overall fragment length is proportional to the number of bits encoded, each molbit must be made as short as possible. The trade-off

Molbit set	Molbit length (nt)	Adapter length (nt)	Unique adapters	Unique molbits
Set 1	15	15	2	4
Set 2	10	15	2	4
Set 3	15	15	4	8

Table 3.1: **Characteristics of designed molbit sets.**

is that shortening the sequence length reduces the maximum attainable squiggle diversity. Additionally, since 5-6 nucleotides fit inside the sensitive region of the nanopore, all 5-6 nucleotides influence the signal produced at any given time. As a result, the nanopore signal for the first and last 2-3 nucleotides of each molbit is impacted by flanking sequences. This creates a rough lower bound on the shortest possible barcode length — 6 nucleotides is certainly too short, but it is unclear how many more nucleotides are required to produce enough signal to differentiate the molbits. Porcupine uses 36 nt for the unique portion of its molbits, which is long enough to produce a considerable amount of sequence and squiggle diversity. Here, I explore sequences of length 10 and 15.

I designed three sets of molbits, each varying in their molbit length or the number of unique adapters (Table 3.1). To design the molbits, I used Porcupine's evolutionary design algorithm with parameter modifications to suit the shorter lengths (see Methods). Figure 3.2 shows the DTW scores for each set before and after evolution. There is no theoretically ideal DTW score so evolution continues until convergence (local optimum). Additionally, the maximum DTW score for each set is limited by the molbit length. Shorter sequences are overall less distinct, so DTW scores in Figure 3.2 are only directly comparable between sequences of equal length (Sets 1 and 3, in this case, since they are both 15 nt).

Choosing adapter sequences

Choosing effective adapter sequences is critical in order to sequentially encode molbits. In Figure 3.1b, adapter sequences are present in between each molbit. These are functionally

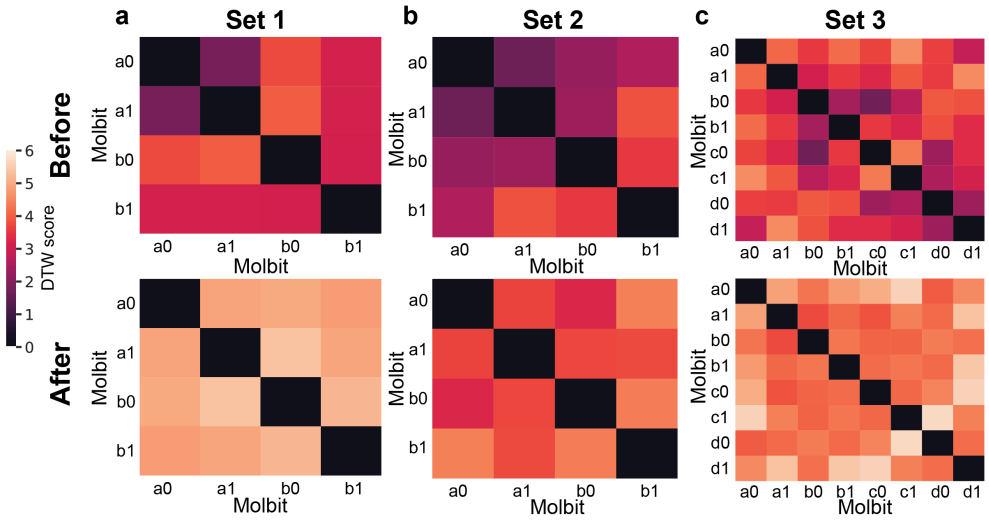

Figure 3.2: **Dynamic time warping scores before (top) and after (bottom) evolving three sets of molbits.** Plots a-c represent molbit sets 1-3 respectively. DTW scores are lowest for Set 2 due to the shorter sequences (10 nt compared to 15 nt for Sets 1 and 3).

necessary to assemble strands of information together. When individual molbits are not yet assembled into molbit strands, the adapter sequences are single-stranded overhangs; molbits can then be combined into strands by adding different molbits with complementary adapters. A basic requirement for adapters is that they must not contain the same (complementary) sequence on both ends, or the molbits will hybridize and unintentionally form long strands. Thus, the molbits alternate between at least two adapters, but using more adapters could reduce the number of extension steps by allowing several molbits to be added to a solution concurrently.

I also ensured that the adapters would not fold onto themselves, and that they were sufficiently long (15 nt) to stably hybridize without requiring ligation after each addition. These were selected from a predeterimined list of orthogonal primers (see Methods).

3.2.4 Methods

Code availability: A portion of the code to implement these methods is freely available as part of Porcupine. The rest of the code was written while employed by Microsoft Research, and is licensed by Microsoft to myself for the purposes of this thesis (not publicly available).

Molbit design algorithm

Big Bits uses the evolutionary model from Porcupine, found in Section 2.2.4. Evolution was stopped after 15 rounds, as sequences typically stopped improving at or before that point. The maximum sequence similarity under Smith-Waterman (match=1, mismatch=-1, gap=-8) was set to 5.

Porcupine uses a free energy calculator based on python bindings for NUPACK [69]. However, due to NUPACK's restrictive license, this could not be used in a commercial setting with Microsoft. Instead, I used seqfold [65], a python package implemented using the same algorithm as UNAFold and mfold [70], which produced similar enough results to NUPACK to achieve the same purpose.

Selecting adapters

Adapters were selected from a list of primers of length 15 nt sourced by Yuan-Jyue Chen. This list contains validated primers that are known to have minimal interactions with one another. I computed the folding energy at room temperature (21 C) for all primers in this list using seqfold [65] and selected two or four with the lowest potential for self-folding.

3.2.5 Discussion

While molbit design in Big Bits is nearly identical to Porcupine, interesting trade-offs arise between the number of desired molbits and their length, impacting the theoretical uniqueness of the sets of bits. Generally, the more molbits there are in a set, the less distinct each molbit is from the rest of the set, reflected in lower DTW scores. Additionally, longer molbits

can produce more distinct squiggles and therefore higher DTW scores. Porcupine uses a relatively large number of molbits (reducing theoretical DTW scores), but longer sequences (increasing DTW). Big bits uses very few molbits (increasing DTW), but very short sequences (decreasing DTW). It is also unclear how DTW scores relate to classification accuracy, which is the true measure of separability in a real world system. Characterizing this trade-off could lead to a better selection of sequence length and molbit set size for various applications.

Selecting adapters is more consequential for Big Bits than Porcupine due to how the bits are assembled, yet adapter selection was only minimally explored in this pilot project. In Big Bits, only the individual molbits are preassembled, so the efficiency of writing (i.e., effectiveness of adding each next bit) depends on how well each adapter hybridizes and whether it remains stable until the next round of ligation to permanently attach the molbits together. In contrast, for Porcupine, molbits are fully assembled before encoding information and nothing needs to happen chemically for them to be combined in a single tag. Since molbit squiggle diversity is impacted by the adapter sequence, selecting adapters with this in mind could produce a more effective set of molbits without increasing the molbit length.

While I can speculate about the effectiveness of different bit lengths and adapters using theoretical data, this can only truly be evaluated holistically after going through the process of gathering and labeling test data, and training a model to identify them.

Contributions

My contributions include a structural design for sequentially encoding bits for DNA data storage, an approach to designing molbits and their adapters using an evolutionary algorithm and folding energy, and three sets of molbits.

Future work

Although this project examines the lower bounds of sequence length, evaluating longer molbit sequences like those from Porcupine could also be useful. In Big Bits, it is possible that the overall fragment length will be less of a concern than anticipated, or that even broken

fragments are useful for decoding information. Nanopore sequencing can handle substantially long fragments (easily 10k+ nt, 100k+ nt not uncommon), but longevity of storage would need to be evaluated further since the likelihood of damage increases as fragments get longer. Porcupine's molbits have already undergone several rounds of testing so I know they are unique, and this may be advantageous for Big Bits.

Adapter selection also has ample room for improvement. First, adapters could be evaluated not just for sequence-based distinctness and folding potential, but also for hybridization energetics. If the adapters strongly and stably bind with their reverse complement to the best extent possible, more molbit strand extensions could occur before ligation. Second, it may be possible to shorten the adapter to just 4 nt by using Golden Gate assembly as in Porcupine, following the method used by Potapov et. al [56] to assemble a lac cassette with up to 24 fragments. This would allow assembly of up to 24 molbits at a time, and enable easier creation of longer molbit strands.

3.3 Acquiring and labeling training data

3.3.1 Introduction

After designing molbit sequences and their adapters, the next step is to evaluate how separable they are in practice. A theoretical understanding of their uniqueness using DTW scores and simulated squiggles is insufficient for determining whether the molbits are effective. Much like for Porcupine, separability can only be measured by generating real data and training a model to identify the bits within each strand. In this section, I discuss how this training data was acquired and labeled.

To test these molbits, they were first assembled into molbit strands. Then I labeled the bits that are present. In an ideal training dataset, we would be able to (1) precisely control the order and position of molbits in each molbit strand so the bits contained within each strand is known, and (2) observe all or nearly all possible bit orderings in the dataset. In reality, these two desires are conflicting — we can realistically produce either a few well-

controlled sequences or a large variety of unknown sequences. Controlling the order of the bits requires adding 0- or 1-molbits to the strand individually, requiring either a painstaking manual assembly or an automated fluidic system. Each added bit in length doubles the number of sequences to produce if all unique combinations are generated. In contrast, random assembly produces greater sequence diversity in fewer steps. At each molbit position, both the 0-molbit and 1-molbit are added to the solution containing the molbit strands; either molbit may randomly bind to the existing molbit strands. This comes at a cost of making labeling more difficult and likely less accurate. Pre-assembled molbit strands cannot be ordered from companies like IDT and Twist due to complexity restrictions caused by the repetitive sequences. However, even if repetition was not an issue, it would likely be cost prohibitive to order a sufficiently diverse set of molbit strands.

Ultimately, I chose the second option – random assembly of molbit strands. Labeling this data is nontrivial due to the inherent lack of ground truth labels and because the high level of repetition causes existing genomic-focused alignment tools to fail. Here I describe our data collection methods and my labeling strategy.

3.3.2 Related work

Traditional genomic sequence aligners use the Smith-Waterman algorithm [61] to align small seeds to a reference genome. Two dominant long read aligners, BWA-MEM [42] and Minimap2 [43], both use a seeding-and-chaining approach to identify regions where several seeds closely align; however, this approach fails for short, highly repetitive sequences. First, genomic aligners typically expect that the reference genome is longer than each read, or at least comparable in size. When that assumption is violated, only partial alignments can be found, and typically only one or a few occurrences of each reference is returned for each read. Second, it is expected that each read maps to only one (or very few) unique locations in the reference genome. Big Bits presents the opposite situation: each read maps to all reference sequences many times. Lastly, there is an assumption that most originate from only one reference sequence. In typical genomic experiments, some DNA may be spliced (i.e.

two genomic regions that are not adjacent in the reference genome are fused together) and some aligners may accommodate a splice or two per strand. Reads do not contain multiple reference sequences within them. This occasionally occurs in standard sequencing experiments due to unintentional splicing, and some aligners may accommodate a splice or two per strand. For Big Bits, in this context each read can be considered a multi-way splice of multiple reference sequences.

3.3.3 Results

Assembling molbit strands

Note: Yuan-Jyue Chen carried out the molbit assembly in the lab (see Methods).

Molbit strands were first generated by assembling the individual molbits into longer (8 bit) strands in successive rounds of hybridization (Figure 3.3). Strands were seeded using a tether containing a hanging $a*$ adapter. In the first round of extension (the first bit position), both molbits $a0$ and $a1$ were added to the tethered solution. This effectively randomly distributed 0's and 1's across the first bit position in the molbit strands by hybridizing the $a*$ tether with the a adapter from either molbit $a0$ or $a1$. Unattached molbits were then washed out of solution, leaving a $b*$ overhang available for the next round of extension. In the second round, both molbits $b0$ and $b1$ were added to the tethered solution, again randomly distributing 0's and 1's across the molbit strands. The strands were ligated after every set of 4 extensions. A total of 8 rounds of extensions was performed; theoretically, this means each molbit should contain 8 bits. After performing sequencing, I found that the molbit strands were much longer than anticipated, discussed further in the next section.

To give an idea of the scale of the molbit strand assembly process, millions of strands were likely generated, as estimated by sequencing yield rate and read counts. After the molbit strands were assembled and ligated, the tether was detached and the sample was prepared for sequencing (see Methods). Sequencing yielded ∼400,000 reads in one hour with no signs of a depleting sample (the rate of sequencing remained consistent).

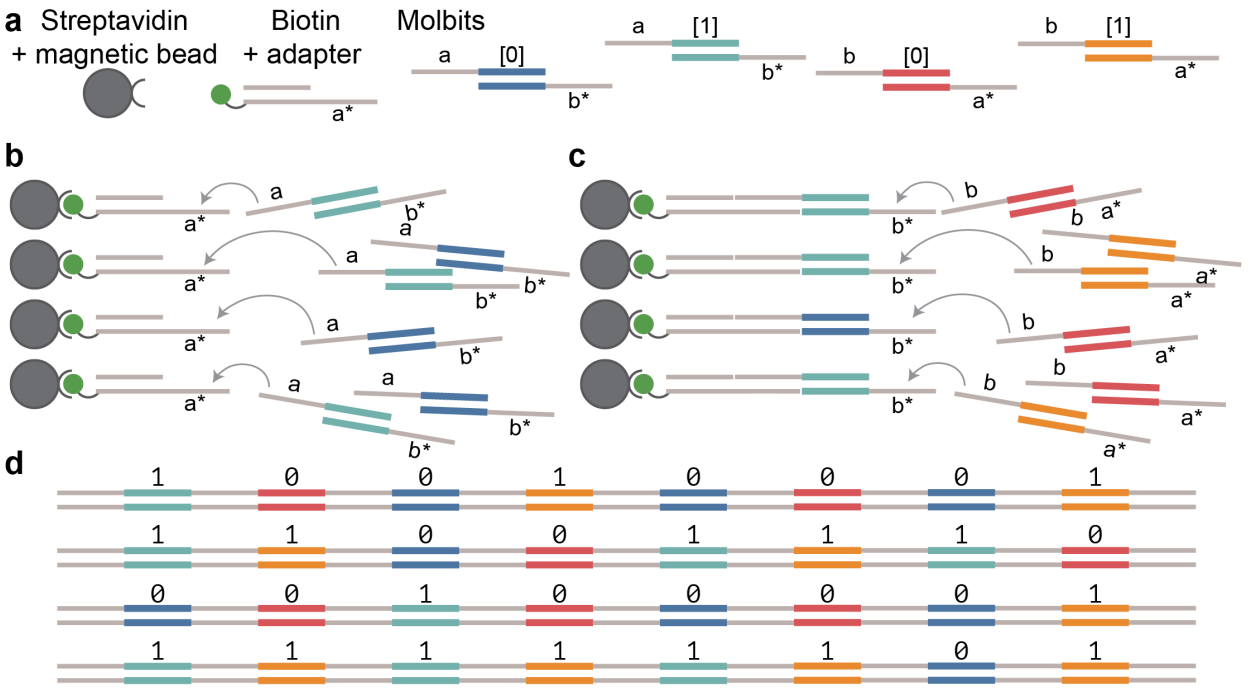

Figure 3.3: **Overview of the molbit strand assembly process.** (a) Inputs for strand assembly, excluding reagents. The biotin and streptavidin act as a tether preventing circular strands from forming. (b) With the a* overhang exposed, both the a[0]b* and a[1]b* molbits are added to the system where they randomly hybridize with a*. Remaining molbits are washed out of the solution. (c) With the b* overhang now exposed, both the b[0]a* and b[1]a* molbits are added to the system where they randomly hybridize with b*. Remaining molbits are washed out of the solution. After repeating (b) and (c) twice (i.e. after four molbits have been added), strands are ligated to permanently attach the molbits. (d) Fully assembled 8-bit molbit strands, with the tether cleaved from the strands. The strands can now be prepared for nanopore sequencing.

Labeling molbit strands

We assembled, sequenced, and labeled molbit strands for only molbit set 1 (15 nt molbits, two alternating adapters, sequences in Appendix Table B.1) due to time constraints. A label is defined as the string of bits present in the molbit strand. Before alignment could be carried out, the reads first needed to be basecalled. I used the standard basecalling software Guppy (see Methods).

Labeling posed a significant challenge due to the repetitiveness of the sequences. Traditional (genomic) sequence alignment tools (i.e., those mentioned in Related Work) work by aligning short segments of each read to a genome ("seeding"), ideally resulting in a cluster of segments that align near each other in the genome, narrowing down the approximate location of the read ("chaining"). Then a second pass of alignment is completed to refine the precise location of the read. Repetitive regions of genomic DNA, for example, Short Tandem Repeats (STRs), are notoriously difficult to align and call variants in a first-pass alignment partly because the initial seeding is too ambiguous, so specialized callers have been published that vary widely and are often highly specific to the sample preparation [23, 37, 31, 64]. Since molbit strands consist of just a few relatively short, repeated sequences, they effectively represent a worst case scenario for alignment.

Since complex genomic alignment tools were ineffective, I implemented a simpler approach that searches along each read for molbits. This algorithm slides a window along the length of the read and finds the closest sequence match to any of the possible molbits in each window, accounting for the alternating adapters. Because nanopore sequencing can read either the forward or reverse strand, this was repeated for the reverse complement of the sequence and the better of the two matches was selected. This method is not computationally efficient (although it is at least easily parallelized by applying this operation to many reads at once), and would require optimizations to move beyond a proof of concept. See Methods for more detail about the algorithm.

Measuring correctness of these labels is nontrivial since the molbit strands were not

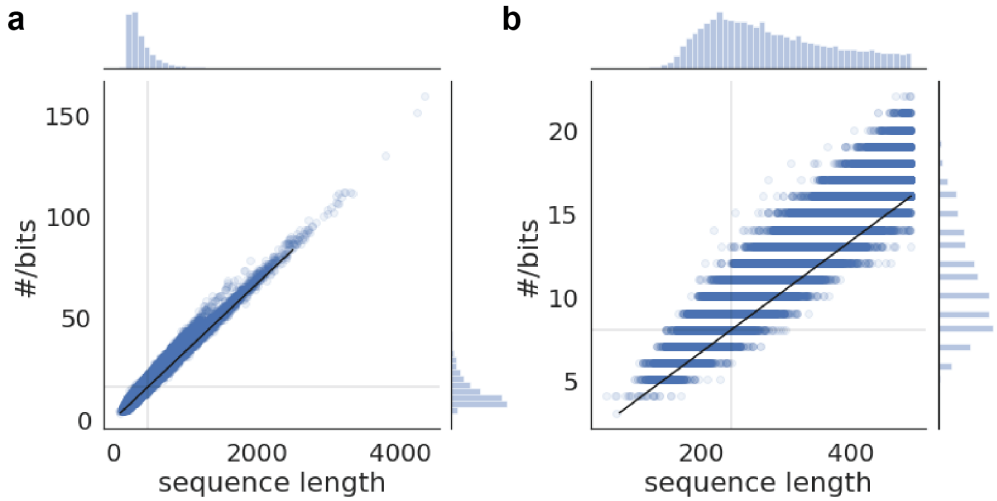

Figure 3.4: **Relationship between the length of sequences and the number of bits identified.** (a) Each point in this scatterplot represents one read's sequence length and the number of bits identified in that read (per the molbit sequence labeling algorithm). The black diagonal line is the expected number of bits for a given sequence length. The faint gray horizontal and vertical lines show the experimentally intended number of bits and sequence length, respectively. The histograms at the top and right of the figure show the distribution of sequence lengths and number of bits, respectively. (b) Same dataset and plot type as in (a) but zoomed in on the lower end of the plot.

generated in a controlled manner – there is no true known label. To approximate correctness, I compared the sequence length of each read with the number of bits identified. Figure 3.4 shows a close relationship between sequence length and the number of bits identified, plus the distribution of all sequence lengths and the distribution of the number of bits identified. While most sequence lengths are near the expected 240 nt, there is a long tail of longer sequences, indicating either spurious molbit additions during assembly or sequencing artefacts. The number of bits per strand is proportional to the sequence length (Pearson correlation r=0.989, p<0.0001), indicating that the labeling strategy is at least generating a reasonable number of molbits given the sequence length.

3.3.4 Methods

Code availability: The code was licensed by Microsoft to myself for the purposes of this thesis and is not publicly available.

Molbit labeling algorithm

The goal of this algorithm is to identify which molbits are present in a sequencing read, using only the basecalled sequencing read and a set of query sequences (molbit sequences). Here, "search window" refers to the segment of a sequencing read which is being examined for the presence of a query sequence, "search length" refers to the number of nucleotides comprising the search window, and "slide length" refers to the offset between the start of each window, moving from 5' to 3' in the sequencing read.

The algorithm has two phases: first, identifying molbits by calculating a match score for each molbit in each search window, and second, tracing through the scores to find the lowest-scored path. The first phase starts with a search window at the beginning of the read. The edit distance is computed between the search window and each of the molbit sequences plus its adapter (i.e. searching for [a + a0], [a + a1], etc.), using a utility from a data storage decoder proprietary to Microsoft. The scores are recorded for that window, and the next

search window is determined by sliding along the read. I found that a search length of 40 and a slide length of 15 worked well.

The second phase traces back through the scores to produce a list of molbits. For each window, the molbit with the best (lowest) score is chosen and added to a list. Note that because the search length is longer than the length of the molbit plus adapter, and the slide length is much shorter than the search length, the algorithm will frequently output the same molbit for sequential search windows. These are then collapsed into one instance of the molbit, alternating between adapters. The score of the overall molbit strand is the sum of the best scores for the molbits in the strand.

The algorithm is run on both a read and its reverse complement separately, then the lower overall score is chosen. I parallelized this method to reduce runtime.

Experimental methods

Molbit strand assembly

Yuan-Jyue Chen carried out the molbit strand assembly and provided this record of the experiment.

1. **Bind an anchor strand to T1 beads.** A biotinylated anchor strand was mixed to bind to streptavidin coated magnetic beads (Thermo Fisher Dynabeads™ MyOne™ Streptavidin T1; catalog #: 65601) and sat at room temperature for 10 minutes. The beads sat next to a magnet for 1 minute, followed by 2 times wash using 1X Binding and Wash (B&W) Buffer (1M NaCl, 0.5mM EDTA, and 5 mM Tris-HCl (pH 7.5)). The beads were resuspended with 20 uL of 1X B&W Buffer.

2. **Hybridize with DNA a[0]b* and a[1]b*.** 20 uL of the T1 beads were mixed with 12 uL of mixture of two different DNA a[0]b* and a[1]b* at 10 uM and sit at room temperature for 5 minutes. The beads sat next to a magnet for 1 minute, followed by one time wash using 1000 uL 1X B&W buffer. The beads were eluted in 20 uL of 1M NaCl.

3. **Hybridize with DNA b[0]a* and b[1]a*.** The beads were mixed with two different DNA b[0]a* and b[1]a* and sit at room temperature for 5 minutes. The beads sat next to a magnet for 1 minute, followed by one time wash using 1000 uL 1X B&W buffer. The beads were eluted in 20 uL of 1M NaCl.

4. **Repeat steps 2 and 3.**

5. **DNA ligation.** 20 uL of the DNA attached beads from step 4 were then mixed with 2uL of T4 DNA ligase, 3 uL of 10X ligation buffer and 5 uL of molecular water. The mixture sat at room temperature for 10 minutes.

6. **Repeat steps 2 and 3 twice.**

7. **USER digestion.** 20 uL of the sample from step 6 was mixed with 5 uL of USER enzyme, 3 uL of 10X Cut Smart buffer, and 2 uL of molecular water. The mixture sat next to a magnet for 1-2 minutes. The supernatant was transferred to another clean tube.

8. **Ligation.** 35 uL of the sample from last step was mixed with 10 uL of T4 DNA ligase and 5 uL of 10X ligation buffer.

9. **PCR Purification.** The sample from step 8 was purified by following a protocol of QIAquick PCR purification. The purified sample was eluted using 50 uL of Elution Buffer. The concentration of the purified DNA was quantified using a BioPhotometer, and the concentration was 20 ng per uL.

Sequencing preparation and basecalling

Molbit strands were prepared for sequencing using ONT's Ligation Sequencing Kit (SQK-LSK109) following the kit protocol and ONT's Flow Cell Priming Kit (EXP-FLP001). Molbits were sequenced on a R9.4.1 MinION flow cell. Molbit strands were basecalled using Guppy GPU v3.2.2+9fe0a78.

3.3.5 Discussion

This phase of the pilot project demonstrates assembly of molbit strands, acquiring sequencing data, and labeling the resulting reads. Overall, this was successful and produced usable training data for the model discussed in the next section.

For DNA data storage applications, the ideal length of molbit strands is much longer than 8 bits, but I intended for the strands to contain exactly 8 bits as a proof of concept. This initial test saw uncontrolled extension that caused strands to grow past 8 bits. This effect was both positive and negative; observing strands that were quite long indicates that it will be possible to encode long strands, but the uncontrolled extension needs further investigation to be able to reliably encode information. Additionally, since this process was carried out manually, it is labor intensive to generate non-random molbit strands, and automation would be required for this approach to be practical.

Labeling these randomized molbit strands is challenging due to the repetitive nature of the strands — there are just four recurring molbits in the set I tested experimentally. Without the ability to use traditional sequence aligners since they fail for repetitive sequences, I implemented an approach to labeling sequences using a sliding window approach. This is certainly not optimal in time complexity, but since the labeling process only needs to occur once, it was not worth spending time to optimize. Since label quality has a critical impact on the success of model training, it is possible that labeling will be a limiting factor in the overall accuracy of the method and success of the information storage approach. Time was also a barrier for more sophisticated methods development.

Contributions

I contributed a method for labeling highly repetitive, complex sequences. I also led the process of data acquisition, carried out labeling for the molbits I designed, and analyzed label correctness.

Future work

In the future, there is room for more work in developing better ways to label individual reads. For example, one option is an HMM. In this setup, molbit labels would be emissions, basecalled reads would be observations, and the states would be the molbit and adapter sequences.

Further experimental work needs to be done to evaluate options for better controlling encoding.

3.4 Reading bit strings

3.4.1 Introduction

At this stage, I have acquired raw data for one set of molbits and assigned labels with a reasonable level of confidence. The objective in this section is to identify the molbits using the ionic current data as input, and the previously determined labels for training. In this machine learning task, the goal is to use one time series (raw nanopore data observations) to predict another time series (molbit labels). Since both the input and output in this setup are time series data, this can be framed as a sequence-to-sequence prediction task. One alternative to a sequence prediction task is to segment molbits and then classify them individually. This alternative is arguably more challenging since in order to train a model to segment the data, I would need labels indicating where the data should be segmented, which was not practical for the remaining time available.

In this section, I pursued sequence prediction rather than the segmentation and classification approach. The prediction target could be either a fixed-length array (since there is a theoretically "known" molbit length) or a variable-length sequence of molbits. Additionally, this could either predict a binary string (e.g. [0, 1, 1, 0]) or the full adapter plus molbit (e.g. [a0, b1, a1, b0]). Predicting binary may seem like a more natural approach, but there is high variability within each class; for example, each 0 represents two unique squiggle signatures, a0 and b0. Predicting the combination of a molbit and adapter reduces the variability in

the raw signal within each class to just a specific squiggle signature. Both were tested, but ultimately, predicting the full adapter plus molbit was most successful.

3.4.2 Related work

Sequence prediction is commonly used for applications like handwriting transcription [26], speech recognition [5], and machine translation between human languages [66]. Natural language processing, DNA sequences, and, broadly, signal processing applications share commonalities in their characteristics and tools used to process them. Basecalling is yet another application of sequence prediction, transforming current observations into nucleotides assumed to have produced those current observations. Identifying molbits is similar to basecalling but with a wider field of view.

Early basecallers used HMMs [59, 19, 20] due to the relatively distinct mean and variance of each kmer, visually producing a step function pattern in the current trace. Modern basecallers have used various techniques including CNNs, recurrent networks like LSTMs [29, 63] (currently disfavored due to speed), and loss functions based on connectionist temporal classification (CTC) [28, 25].

3.4.3 Preliminary Results

Prediction task setup

Here I explore both fixed length classification and variable length sequence prediction. For fixed length classification, the model predicts just the first 8 bits, the number of bits I intended to incorporate into each molbit strand (although Figure 3.4 demonstrates that in most cases, far more than 8 bits were included per strand). For variable length predictions, up to 40 molbits can be predicted.

Accuracy is measured using different methods based on each prediction target. Random guessing is equivalent to 25% accuracy, or choosing 1 of the 4 molbits at random. When I predict only the first 8 bit positions, accuracy is defined as the number of correct bits divided

by 8. Model performance is measured via cross entropy loss. In contrast, for variable length prediction, model performance is measured via CTC loss since the timing and offsets of each prediction vary. For example, if the first bit was omitted but the rest of the bits were correct, that is a reasonable prediction, but computing accuracy by just comparing positions would measure this badly due to the offset. Instead, I propose finding the best scoring overlap between the two sequences.

Model development progress

Results for molbit identification have been promising, but not yet accurate enough to consider model development complete. Time was a limiting factor here — the internship ended before this phase was complete. Ideally, multiple iterations of the entire design process would be carried out.

The classification approach predicting just the first 8 bits used Porcupine's model (Appendix Figure A.3) as a baseline. The best configuration (including modifications to input data, downsampling, batch size, etc. but not the model) performed at an average of approximately 50% accuracy for both training and validation. This may seem like a significant improvement over random guessing (25%), however, all models I tested picked up on the alternating pattern very quickly, i.e., it predicted an "a" adapter then a "b" adapter, regardless of bit. The 50% accuracy reflects random guessing between both bits for the same adapter at each position. Because the molbits in Porcupine were 40 nt and the model was directly reused, it appears that the field of view in the CNN may have been too large for this application.

An approach using Porcupine's model with fewer layers and a smaller field of view performed much better on training data (97.9% accuracy) but showed evidence of extreme overfitting (random (50%) predictions on validation). This indicates that further tweaking would likely produce a reasonably performing model. I attempted to improve generalization using quick fixes, including increasing dropout and stricter regularization, before moving on to a different prediction task and output strategy in the interest of time.

The next approach predicts arbitrarily long sequences of bits (up to 40 molbits). This uses the same model as described in the previous paragraph, except this model uses CTC for prediction and loss. This model also suffered from similar generalization problems, but can still be considered an improvement over the previous approach since it can predict more realistic sequence lengths and not just 8 bits. Future developments should start here, tweaking the internals of this model to improve generalization, and ideally implementing a reasonable accuracy calculation outside of CTC.

A final, untested approach would seek to segment the molbits from the raw signal first, then classify them separately. A challenge with this approach is identifying where and how to segment the data. Mapping the raw signal to the nucleotides that produced it is no longer natively possible with modern basecallers because they do not return the location of each nucleotide in the raw signal. If segmentation was straightforward, this classification approach would likely be successful.

3.4.4 Methods

Code availability: The code was licensed by Microsoft to myself for the purposes of this thesis and is not publicly available.

Molbit identification model

The baseline model used was Porcupine's CNN, which is detailed in Section 2.3.4 and shown in Appendix Figure A.3. Dozens of modifications were made to this model for Big Bits, and the model represented here is a pared down version of Porcupine's model with fewer layers, and critically, a narrower field of view in the original nanopore ionic current. I show a diagram of the model with exact parameters for each layer in Appendix Figure B.1.

The pared down model consists of a 2 layer CNN, followed by one fully connected layer with 20% dropout, and a final fully connected layer with softmax for the output. When only the first 8 molbits were classified, cross-entropy loss was used to measure model improvement. When a variable length sequence was predicted, CTC loss was used. Balancing the training

dataset was not necessary in this case because there was an approximately even distribution of molbits in all positions.

Pytorch FAST5 data loader module

Reading from FAST5 files for training a model is nontrivial. Using default data loading tools can significantly slow training, especially if indices are shuffled between epochs.

Nanopore reads are stored with data for many reads in a single file (default is 4000) *and* many files. This means that training a model requires either loading all data into an array and saving it separately for convenience, which duplicates large amounts of data, or managing a set of training indices that map to read IDs, which in turn map to filenames. Without custom Pytorch tools, the latter is slow and impractical, since it requires searching and opening several large files to retrieve the desired data.

Keeping all data in FAST5 files is best for reproducibility (the original data source often gets lost upon exporting to an array) and for keeping storage requirements as low as possible. To do this and yet maintain high training speeds, I implemented a Pytorch Dataset and DataLoader that can fetch data from FAST5 files in the background. This Fast5Dataset optionally performs downsampling using a Bessel filter, effectively reducing the sampling rate and allowing the model to train on fewer data points. Critically, all the basic Pytorch Dataset functions are implemented so the Fast5Dataset can be dropped in to existing models without modification. Examples of implemented functions include index shuffling and retrieving data (nanopore ionic current and its label) just from an index.

3.4.5 Discussion

Overall, I demonstrated a partial proof of concept for the Big Bits molecular data encoding scheme. Further work needs to be done in model development and possibly testing other sets of molbits. The biggest limitation was time. I believe the accuracy can be improved with more time spent developing and tweaking the model. There may also be fundamental

limits to the accuracy based on the length of the signal relative to the adapter. This could be tested with another round of molbit design, increasing to 20 or 25 nucleotides.

To test the overall concept of encoding molbits sequentially, I would propose using existing Porcupine bits. These would be combined at random, sequenced, and then labeled using the Big Bits approach to labeling sequences. The Porcupine model could then be retrained using the variable length prediction target (using CTC loss). The results of this could shed light on two aspects of the Big Bits overall approach: (1) molbit sequence length and the resulting squiggle diversity (given that Porcupine consists of molbits that have already been proven sufficiently distinct), and (2) whether the labeling approach is reasonable.

Unfortunately, error correction could not be evaluated for this project as it is too early in the development process. A few prerequisites must be met: (1) more bits must be encoded in a single strand — 8 bits is not sufficient for encoding much information, especially with error correction and (2) better accuracy is needed. Porcupine achieved just 1.7% average per-molbit error over each tag, which is quite high for a tagging system but possible to minimize with error correction. Current error rates in Big Bits would require an enormous number of parity bits.

Contributions

My contributions include a code skeleton for developing and testing the machine learning model, a Pytorch FAST5 data loader module, and an initial set of models for identifying bits.

Future work

Continued model development should be a priority for future work, including improving generalization of the current best models. Coming full circle back to basecalling, perhaps emulating modern basecallers but with a wider field of view could be fruitful. The prediction task is identical to basecalling but with more signal contributing to each call. Compared to a

signal-based molbit labeling approach, this would take advantage of a larger range of signal, improving speed, and still avoiding a follow-up step of needing to align the sequences.

Another round of molbit design could help investigate whether molbit detection would benefit from the greater variability offered by longer sequences. One could argue that because basecalling successfully operates on a per-nucleotide level, it is certainly possible to identify molbits that are short (10-15 nt), but it may still be beneficial to increase the length.

The approach of segmenting the molbits first and then classifying them would be another worthy approach for the future.

In order for Big Bits to be successful overall, error correction must be implemented, so it will be necessary to experimentally determine how many bits can reasonably be included per fragment. This will drive what kind of encoding scheme might be used.

3.5 Summary

In summary, Big Bits is a promising alternative method for encoding general-purpose information in DNA. With its modular approach and storage density between that of molecular tagging and DNA data storage, Big Bits offers a middle ground between larger scale information storage and feasibility of reading and writing in a standard laboratory environment. Here I designed three sets of molecular bits which can be combined serially to encode long molbit strands. After a colleague (Yuan-Jyue Chen) helped to generate data intended to contain 8 random molbits, I then labeled these molbit strands using a sliding window algorithm due to technical limitations of existing alignment algorithms and how repetitive sequences are handled. With the labeled data, I developed a prototype model to identify the molbits (requires further development).

Despite a promising start, some specific challenges must be overcome for Big Bits to be successful, both experimental and computational. Experimentally, investigation is needed into how to better control the exact bit encoding and its length. I intended to create a molbit strand with 8 random bits, but observed extension far beyond 8 bits. This could possibly have been caused by incomplete washing causing molbits to still be present in the

solution when the next bit was added, or perhaps strands fused together after cleaving the tether during the ligation step. Computationally, model development is also incomplete. The limitations in model accuracy could be due to the model needing tweaking, or it could be a fundamental barrier due the molbit sequences being too short. The latter could be investigated by carrying out another round of molbit design with longer bits or simply using Porcupine bits, which have already been optimized.

Error correction also remains an open problem for this application. In order to implement this, the per-bit error rate must both be known and low (roughly 1-2% or lower). Error correction will be more complex for Big Bits than Porcupine due to the possibility of insertions and deletions. It is beneficial to know that bits may only be inserted or deleted in pairs (in the case of two alternating adapters), but a random generator matrix will not suffice and the methods used will be closer to those used for DNA data storage than molecular tagging.

Chapter 4

PEPTIDE BARCODE SENSING WITH NANOPORETERS AND PORETITIONER

4.1 Introduction

4.1.1 NanoporeTERs: Nanopore-addressable protein Tags Engineered as Reporters

Recently, we developed a novel technique to use the MinION to detect genetically engineered reporter proteins, which we call *NanoporeTERs* (NTERs), short for Nanopore-addressable protein Tags Engineered as Reporters [13]. Reporter proteins are used to measure biological activities such as transcription. Previous technology was limited by the inability to detect single molecules or by overlapping detection channels. For example, genetically encoded fluorescent reporter proteins like GFP are commonly used, but only a few colors can be detected simultaneously with typical equipment [35]. Instead, NTERs replace fluorescence by using barcoded peptides that can be read individually using the MinION.

Peptides present a unique challenge for molecular sensing using nanopores. For DNA sequencing, each strand's flow through the pore is slowed down and regulated by a helicase-like sequencing adapter that unwinds the double helix. Peptides have no analogous native molecule to "unwind" or unfold the secondary and tertiary structure and feed the amino acids through the pore at a steady rate. Peptide sizes also vary greatly; some smaller peptides flow through so quickly that they are undetectable or register as just a short blip in the ionic current trace, and larger peptides may block the pores completely. Charge and solubility also influence how likely a peptide is to enter the pore, whereas these characteristics are nearly uniform for DNA fragments.

NTERs are structurally designed to make use of the unique and varying characteristics of peptides. First, a negatively charged amino acid tail threads the molecule through the

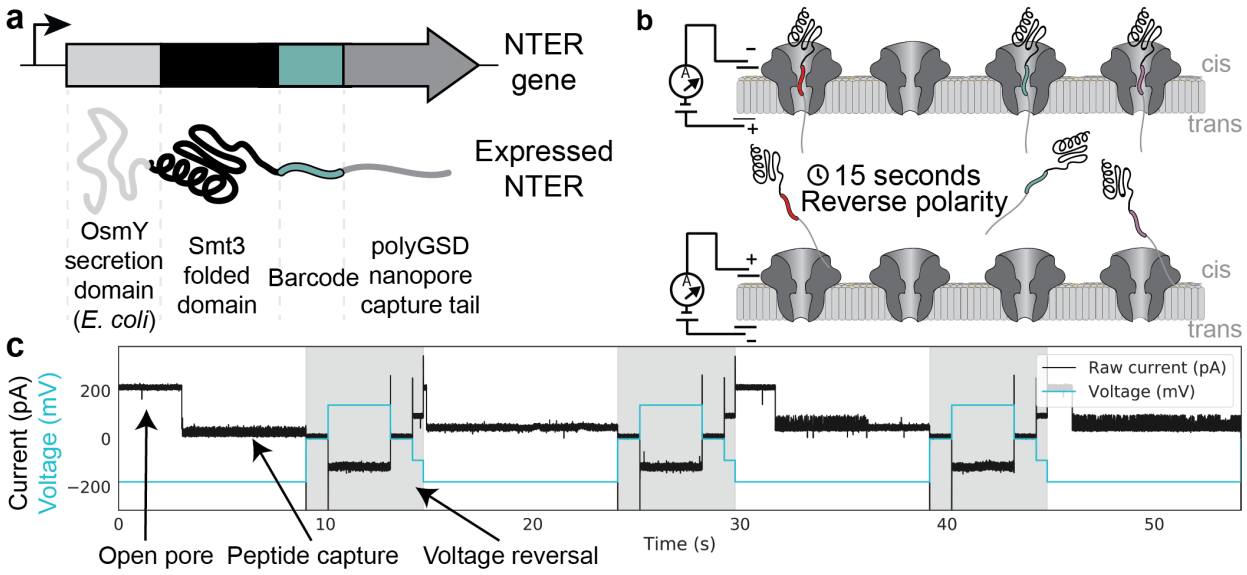

Figure 4.1: **NanoporeTER structure and overview of data gathering process.** (a) Anatomy of a NanoporeTER (NTER) gene and protein. (b) Diagram of captured (top) and ejected (bottom) NTERs. (c) Nanopore ionic current trace. When voltage is "normal" (-180 mV), an NTER may be captured, causing the current to drop. NTERs are ejected by reversing the voltage (gray regions).

pore, followed by an amino acid barcode which is blocked from fully translocating by a larger peptide domain (Figure 4.1a). Unlike DNA which fully translocates through the pore, the NTERs remain lodged in the pores until the voltage applied to the pores is reversed, dislodging the NTERs back to the cis side of the membrane (Figure 4.1b). This voltage reversal occurs at fixed intervals, producing a recurring ionic current pattern, with pores constantly cycling between a fully open state, to capturing an NTER, to voltage reversal, and back (Figure 4.1c).

In addition to the biological challenges of using peptides for molecular sensing, a technical challenge arose due to a lack of infrastructure. Infrastructure that processes DNA nanopore data cannot be natively transferred to peptides. When data is collected for DNA sequencing, only DNA sequencing reads are returned — data *only* from when DNA is actively in the pore. When data is collected for peptides, we receive ionic current data for all of the nanopores in the array, from the time the machine is turned on until the run is completed. This lack of infrastructure meant we needed to implement tools that can identify when peptides reside in the pore, extract these peptides from the ionic current trace, and analyze the resulting reads.

Here I present Poretitioner, a user-friendly command line tool and python package to support analysis of NanoporeTERs and other sensor molecules being developed in the lab. Poretitioner executes a series of data processing steps within a pipeline, producing segmented peptide data, classification and quantification results.

4.2 Poretitioner: Infrastructure for general-purpose nanopore sensing

4.2.1 Introduction

To support the NanoporeTER publication [13] and the experimental work leading to it, I previously developed a pipeline to process nanopore ionic current data and facilitate its analysis. The pipeline was initially designed concurrently with experiments, quickly adding functionality to support new approaches. However, as a result, it consists of a set of inefficient

modules that are hard to use, generate large intermediate files, and are overall insufficient for distribution beyond the MISL lab. The original NTER pipeline can be considered a record for reproducing this specific publication, but a standalone tool is valuable and necessary for others to use NTERs or other analytes in practice.

The improved pipeline, Poretitioner, is a standalone tool that supports analysis of NanoporeTERs, other sensor molecules being developed in the lab, and future exciting applications using molecular detection on the MinION. I also present a specification for a new variant of the FAST5 format to record peptide capture data, plus additional information like classification results and parameters to filter the data.

4.2.2 Related work

Bioinformatic pipelines

Bioinformatic pipelines are broadly defined as a series of steps executed sequentially to analyze biological data. Most existing methods for processing bioinformatic data are offline, using the algorithmic definition in which all data is available for analysis at the beginning of processing rather than streamed and analysed as data is generated. For example, an analyst may perform sequence alignment on the full dataset, then quality filtering on the output of sequence alignment, followed by variant calling. Pipelining tools like Galaxy [1], Snakemake [39], and Nextflow [17] remove the error prone burden of manually scripting several steps or algorithms, offer better reproducibility, and some enable faster computation via distributed computing.

In contrast to traditional sequencing and bioinformatics, nanopore data is natively suitable for streaming. The potential for realtime analysis is often touted as a benefit of switching to nanopore sensing. The recent public release of ONT's MinKNOW nanopore driver software and development of selective sequencing methods [55, 46] makes streaming a realistic option for sequencing data, and may also be possible to implement for non-DNA experiments in the near future. (Selective sequencing uses an API to actively and rapidly respond

based on which type of molecule is currently in the pore, and eject the molecule if it does not match desired criteria.) However, this currently cannot be extended to non-sequencing applications since MinKNOW (the host computer software that drives the nanopore device) writes bulk raw data to disk in a way that is not simultaneously readable, disabling streaming. This is likely not a technical limitation as the input/output library, HDF5, is capable of simultaneous reading and writing, but rather an oversight that can be fixed if alternative, non-sequencing nanopore applications become more popular.

Read segmentation

When nucleotides flow through a nanopore with the help of an unwinding mechanism, they produce a characteristic, relatively standard signal that is easily and automatically identified and separated (*segmented*) from the rest of the nanopore signal. The unsegmented nanopore trace contains other states such as open pores with no analyte, reversed voltage, and non-functional pores. Since segmentation happens automatically within ONT software and only sequencing is supported, there is no equivalent for segmenting peptides and other analytes. Thus, some other method of segmentation is necessary.

Early publications segmented data by hand, for example Nivala et al. [50]. This was possible due to the limited amount of data produced for a single pore (dozens to a few hundred peptide captures across multiple runs). Recent approaches that generate more data require automatic segmentation. For example, Chavis et al. used a simple heuristic to segment peptides, requiring the current to be lower than 65% of the average open pore current [15]. Schreiber et al. developed a recursive Bayesian model to identify change points within nanopore current, published in a package called PyPore [59]. For each step in the recursion, this method attempts to split the current into two segments by scoring all possible boundaries with a log-likelihood function, and repeating on each half if successful. This approach works reasonably well, however, in order to perform segmentation quickly, it requires a complex platform-dependent Cython installation. One advantage of a recursive approach is that it can identify segments that vary in length when each segment has a distinct

and unchanging mean and variance.

Raw signal classification

Peptide and other analyte classification methods are similar to DNA, which can be found in Section 2.2.2.

4.2.3 Results

Brief comparison of the various FAST5 file types referenced here:

- **FAST5**: File format used to record nanopore data from a DNA sequencing run, with one sequencing read per DNA fragment read by the nanopore.

- **Bulk FAST5**: File format used to record all nanopore current from all pores, from the time the device is turned on and the run starts until data collection is stopped. This data is unsegmented.

- **Peptide FAST5**: File format created by me to record nanopore data produced by Porcupine from a bulk FAST5 file. This includes segmented reads, filters applied, and classification results.

Collecting ionic current data

The pipeline begins by acquiring ionic current data for a nanopore peptide run, which is recorded by MinKNOW in a bulk FAST5 file (Figure 4.2a). A single run may consist of multiple experiments carried out by loading a sample, waiting for data collection, washing the flowcell to remove the sample, and then loading a new sample, all without stopping the data collection.

Previously, The NanoporeTER methods subdivided ionic current from each sample into its own bulk FAST5 file as though it was its own run. This allowed the pipeline to be run on

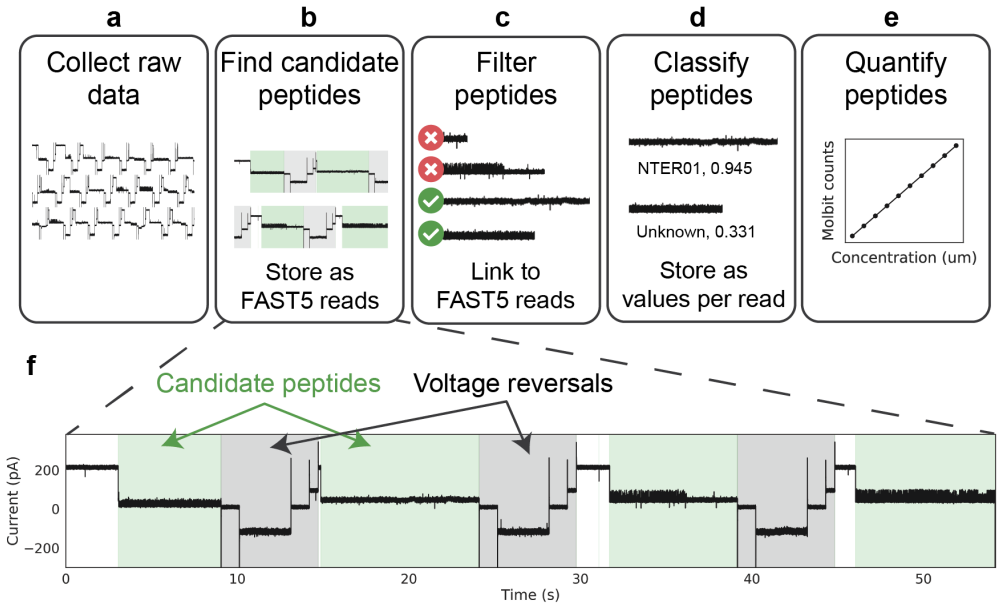

Figure 4.2: **Poretitioner pipeline for segmenting and analyzing NTER peptides.** (a) Bulk ionic current data is collected across all channels. (b) Candidate peptides are found using a threshold on normalized ionic current, where regions of current that extend below open channel current are identified as reads and stored in peptide FAST5 files. (c) Reads are filtered to remove spurious calls, based on heuristics tuned to the particular experiment type. Reads that pass the filter are stored as hard links in the peptide FAST5 file. (d) Reads are red into a classification model. The resulting class label, score, and whether the score meets the confidence requirement is also stored in the peptide FAST5. (e) The concentration of the peptide is optionally calculated. This is currently supported for samples with one purified analyte. (f) Example ionic current trace. Green regions are what Poretitioner would output as candidate peptides in step (b). Gray regions reflect time periods where the voltage was reversing to eject captured peptides. The remaining white regions show the ionic current when no capture is in the pore (open pore current).

	NanoporeTER methods	Poretitioner
Original dataset	26.4 GB	26.4 GB
Saving analyte sub-runs	22.5 GB (peak: 53.0 GB)	–
Segmentation	65.3 GB	17.4 GB (peak: 22.2 GB)
Filtering	1 MB	11 MB
Classification	60 MB	172 MB
Quantification	1 MB	1 MB
Total	**87.8 GB (114.2 GB total)**	**17.6 GB (44.0 GB total)**

Table 4.1: **NanoporeTER and Poretitioner disk storage space profiling**. Values represent the disk space used for each individual step in the pipeline, for the same 90 minute nanopore data acquisition run with 7 sub-runs. Where a peak value is recorded, data temporarily took up that amount of space before compression. Two numbers are given for the total: the size of all steps excluding the original dataset, and in parenthesis, the total including the original dataset. The segmentation step uses more data for NanoporeTERs due to a larger data type (float - 64 bits vs. int16 - 16 bits) and numpy files.

	NanoporeTER methods	Poretitioner
Saving analyte sub-runs	1.6 GB	–
Segmentation	76.5 GB	44.9 GB
Filtering	*Part of classification*	352 MB
Classification	26.4 GB	445 MB
Quantification	19.9 GB	445 MB
Peak memory usage	**76.5 GB**	**44.9 GB**

Table 4.2: **NanoporeTER and Poretitioner memory profiling**. Values represent the peak memory usage for each individual step in the pipeline, for the same 90 minute nanopore data acquisition run with 7 sub-runs. The reduced memory for Poretitioner's filtering, classification, and quantification is due to sequential processing rather than reading all data in at once.

	NanoporeTER methods	Poretitioner
Saving analyte sub-runs	0:28:57	–
Segmentation	1:25:53	1:29:36
Filtering	*Part of classification*	0:05:57
Classification	15:12:30 (*0:09:25)	0:14:39
Quantification	0:05:20	0:01:45
Total	**17:12:40 (*2:09:35)**	**1:51:57**

Table 4.3: **NanoporeTER and Poretitioner runtime profiling**. Values represent the runtime for each individual step in the pipeline, for the same 90 minute nanopore data acquisition run with 7 sub-runs. Values are recorded in the format hours:minutes:seconds. Data was stored on an SSD for this experiment, to minimize the effects of IO on runtime (the available HDD was nearly full and variable seek time could impact results). The long NanoporeTER classification runtime is not a recording error, it was due to an inefficiency, which when fixed produced the runtime in parenthesis. Both are included since the un-fixed run was used for the NanoporeTER publication.

each individual sub-run for convenience at a cost of nearly doubling disk space requirements for an already large file (tens of GB). Later, each capture would be extracted again during segmentation, further increasing storage requirements. Poretitioner simply uses timestamps to extract data from the original bulk FAST5 file without fully extracting it to a new file, then stores only individual reads, which consume much less space. Table 4.1 shows a comparison of disk usage for the NanoporeTER method versus Poretitioner, which is discussed in more detail in the results section titled "Comparison between Poretitioner and NanoporeTER publication methods."

Segmentation: Coarsely identify candidate peptides

Segmentation is the process of identifying when a molecule has been captured in the pore, and extracting ionic current data for that read (Figure 4.2b). To do this, Poretitioner first identifies nonfunctional channels so they can be removed from analysis (see Methods). Any

of the 512 active MinION pores can become erratic or unresponsive over time for a multitude of reasons, including clogging, degradation over time, too high of voltage, etc. Once a channel loses its function, it rarely regains it, so I do not search those channels for potential peptides.

For each functional channel, the segmenter isolates regions of the raw signal where the voltage is negative and the pore can accept a capture, effectively eliminating the gray regions in Figure 4.2f. Then it identifies candidate captures within each of these regions using a current threshold and a set of heuristic filters (minimum, maximum, standard deviation, and length of the raw current). I then extract the data as it was originally recorded in the bulk file, much like reads in DNA, and store them in new peptide FAST5 files (see Methods).

The open channel current is also computed at this stage. Open channel current is the mean ionic current value, in pA, when the pore is fully open. These values average around 220 pA, but can vary from 185 pA to 250 pA between channels on the same flowcell. Since NTERs' mean current level is one of their most distinguishing characteristics, they must be normalized with respect to the open channel current in order to be directly compared to one another. Individual peptide barcodes produce relatively stable current values while they are captured, representing just a fraction of the overall range of possible current values that peptide barcodes can produce. This is in contrast to DNA strands which tend to express their full variability within each strand, assuming a roughly even distribution of nucleotides, meaning each strand can be normalized independently from the channel context using techniques like Median Absolute Deviation [52].

Filtering: Remove candidate peptides outside the expected range

Once reads have been segmented, Poretitioner optionally further filters the reads using heuristics derived from initial peptide experiments to reduce spurious calls and noise (Figure 4.2c). Sometimes it is desirable to only loosely segment the data initially, returning everything that is blocking the pore at any point. For example, calculating the overall pore residency may be useful later on to investigate the amount of time that the pores were available to accept captures for calculating concentration. In past implementations, the data was duplicated

here to produce an additional standalone dataset for training models. Poretitioner avoids this duplication by simply linking to existing data within the peptide FAST5 files. Multiple unique filters can be applied and stored in the same peptide FAST5 files, as long as each filter has a unique name.

Classification: Identifying peptides

Next, reads are classified (Figure 4.2d). In the NanoporeTER publication, a CNN or Random Forest was used for this step [13]. Poretitioner was designed to allow arbitrary classification models to be incorporated into the pipeline. All a user needs to do is write a function with the same inputs (segmented FAST5 files) and outputs (classification label, score, and whether the score meets a confidence threshold) in order to be used by Poretitioner. This is in contrast to previous model implementations which required data to be stored separately or manual feature extraction to produce classification results. In Poretitioner, the classification results are stored directly in peptide FAST5 files, including the classification model name and version, confidence threshold used, and classification results for each read. Multiple classifiers can be applied to and stored in the same dataset, as long as each has a unique name.

Quantification: Calculate peptide concentration

Optionally, peptides may also be quantified by measuring either the average time between captures or the capture frequency (number of captures per pore per minute), and comparing the values to a pre-calculated concentration curve (Figure 4.2e). These curves were calculated as part of the NanoporeTER publication [13], and would need to be recalibrated for each new experiment type and analyte. For clarity, my contributions for this step of the pipeline include the initial implementation of only one of the two quantification methods, but both were heavily refactored for incorporation in Poretitioner.

Comparison between Poretitioner and NanoporeTER publication methods

I compared the performance of Poretitioner and the NanoporeTER publication methods using one 90 minute dataset (a typical run length for these experiments). This 26.4 GB bulk FAST5 file was produced by a 90 minute nanopore run on a MinION. It consists of 5 minutes of buffer in the flowcell (no analyte), followed by an NTER barcode peptide run at six different concentrations separately at different time points during the run (varying between 10-20 minutes of data acquisition; longer times for lower concentration) with flowcell washing in between. I measured each pipeline step's disk space (Table 4.1), memory (Table 4.2), and runtime (Table 4.3). This was carried out using a server running CentOS 7 Red Hat Enterprise Linux, with a 48-core Intel Xeon CPU (E5-2650 v4 @ 2.20GHz), 250 GB memory, and an SSD to minimize the effects of IO on runtime (the available HDD was nearly full and variable seek time could impact results).

Reduction in disk space requirements was Poretitioner's first improvement upon NTER methods, and possibly the most critical given the volume of data produced by nanopore experiments. Most gains come from simply avoiding copying data when possible and using original data types. Overall, the entire pipeline produced an additional 87.8 GB of data using NanoporeTER methods, and only 17.6 GB using Poretitioner.

The largest contribution to the dataset size in the NanoporeTER methods occurs when saving sub-runs (which is completely avoided in Poretitioner), and in segmentation, where captures are stored as `numpy (.npy)` files. This file type has three issues: (1) the choice of data type bloats the file size, (2) files are not compatible cross-language and cross-version, and (3) information context is lost. For (1), data was stored in the `numpy` arrays as `float64`s, using 64 bits per data point. Data in the bulk and peptide FAST5 files is stored as `int16`s, using 16 bits per data point. One might suggest storing the `int16` values in `numpy` arrays, but this requires additionally storing the ADC values needed to convert to picoamperes or normalized current. This alone reflects a 4x increase in size for each capture. For (2), `numpy` files can only be opened in Python, and when a file is generated using Python 2.7, it may

not be opened using Python 3.x, and vice versa. For (3), when a capture is stored in a `numpy` array, the context about where it came from (e.g. which bulk FAST5 file, channel, start time, etc.) must be kept in a separate registry or it is lost. These three reasons also apply to other language- and package-specific file formats like `pickle (.pkl)` files. I cannot emphasize strongly enough that these file types should be avoided for nanopore data; storing as much data as possible in FAST5 files is preferred.

In addition to the drastic reduction in disk space, peak memory usage is significantly reduced (Table 4.2), and runtime is slightly reduced (Table 4.3). It is notable that such large improvements were possible without sacrificing these other aspects. In Poretitioner's segmentation implementation, memory is reduced by only loading ionic current from regions where captures can occur, rather than the entire run. It is reduced in the remaining steps by processing peptide captures sequentially, which does not appear to have an impact on the overall runtime. Segmentation takes approximately the same amount of time in both versions, about 90 minutes, roughly the same time as the initial data collection. This is promising for real-time streaming, especially since the code has not yet been optimized to its full potential. The new code structure allows for easier improvements to parallelization and streaming. During the process of profiling these methods, I discovered an inefficiency that caused classification to take 15 hours to run (an unexpectedly, wildly inefficient data structure update; we all previously assumed it was the classification itself that took so long). Fixing this reduced the runtime to nine minutes. Both numbers are included in the table for posterity.

4.2.4 Methods

Identifying nonfunctional channels and open pore current

At this time, two steps require knowledge of the signal for an entire channel at once: removing nonfunctional channels and finding the open channel current.

A channel is deemed nonfunctional if it has little current variation (mean $<$ 20 and

standard deviation < 50), if current drops to near-zero at some point during the run (at least 5 seconds), or if the pore never returns to an open state. The latter criterion can become a limitation if the sample concentration is so high that peptides are captured nearly instantaneously when the pore is open, so this value can be tuned.

Open channel current is computed by simply extracting all ionic current values within a reasonable expected range for an open channel and finding the median. One limitation is that its robustness depends on the concentration of the sample (i.e. how much time the channel spends in an open pore state) and can drift over time, especially for long runs.

Segmentation

The inputs to segmentation include a bulk FAST5 file, which contains a time series of voltage information and the ionic current for each channel, and additional parameters used to tune the segmentation.

The segmenter first identifies regions during the run where the voltage is set such that a pore can accept a capture. Typically, this is -180 mV, but different voltage settings may be used for alternate experiments. For example, both -140 mV and -220 mV were tested, producing drastic changes to the ionic current trace and therefore requiring different segmentation parameters. The open channel current and nonfunctional channels are also determined before proceeding with segmentation.

Once the windows of time where peptides can be captured are found, the ionic current signal is extracted for each window. Because the open channel current can drift over time, I recompute the open channel current within each window in attempt to produce a more accurate normalization. The ionic current signal is then normalized by dividing by the open channel current, effectively converting from pA to a range of [0, 1] where 0 represents a completely blocked pore and 1 represents an open channel. Potential peptide captures are identified by applying a threshold to this fractionalized current, returning any regions where the current dips below the threshold, presumably where a peptide or other contaminant has entered the pore. Basic filters like a minimum length are applied to avoid excessive

spurious calls. Additionally, the user can choose whether to only return potential captures that remained in the pore until the voltage reversal, i.e. were ejected.

Information about each capture is stored in peptide FAST5 files, including the start time point and duration of each potential capture, open channel current, raw ADC current from the bulk FAST5 file, and all metadata included about the ADC current from the bulk FAST5. Each returned capture is also assigned a globally unique read id. These stored captures will now be referred to as reads.

Filtering

Filtering is implemented as an independent step to further refine the set of captures that are examined for processing. Filters are currently simple and use summary statistics, including just the mean, standard deviation, median, minimum, maximum, length, and whether it was a terminal capture (remaining in the pore until ejected). Poretitioner provides a simple way to implement new filters if desired by adding a function to the code and including it in the callback to apply all filters.

Classification

I did not personally contribute to the actual classifier training or development, but it is included in the pipeline. In Poretitioner, the user specifies which classifier to run as part of the configuration. Metadata about the classifier is stored in the peptide FAST5 for reproducibility (see Peptide FAST5 implementation below).

Quantification

As in the NanoporeTER paper [13], Porcupine provides two methods for quantifying captures: (1) average time between captures, and (2) capture frequency. The average time between captures is computed by sorting the captures by channel and then by time, calculating the time between the end of one capture and the start of the next, minus any voltage

cycling windows. This method requires a degree of confidence in the identified captures, as false calls can dramatically reduce the perceived time between captures. Capture frequency, by comparison, counts the number of reads in the channel and divides by the amount of time. Both methods can be run in windows of time spanning the duration of the run to monitor changes in capture rates over time, which correlates with concentration. A function is provided to convert both values to concentration, per the fit in the NanoporeTER paper [13].

Peptide FAST5 implementation

The peptide FAST5 files are modeled after standard DNA FAST5 files, with modifications to accommodate additional data. For context, FAST5 files are built on top of the HDF5 file type. HDF5 files are structured internally like file systems, meaning data is organized and accessed using paths, groups (similar to directories), and datasets. Peptide reads are stored nearly identically to DNA sequencing reads, with the addition of storing open channel current, since peptides require this for normalization and that information is lost when moving from the bulk file to just the captures. Metadata about the segmentation process is also stored in the peptide FAST5 file. This is not done for DNA, likely because it is consistent between runs (this is not true for peptides, which may require different segmentation parameters depending on the run). That information can be found in the peptide files at `/Meta/Segmentation`.

Filter metadata and results are both stored in the peptide FAST5 file. To avoid data duplication and improve convenience of accessing filtered results, the results are stored within their own group in the peptide FAST5 file using hard links to the original segmented read. For example, the read's original path may be `/read_xxxxxx`, but if the read passes the filter it would be stored at `/Filter/filter_01/pass/read_xxxxxx` with a pointer to the original path.

Metadata about the classifier is stored in the peptide FAST5 for reproducibility, including the classifier name, version, model file, and threshold used to set labels. When the classifier

is finished running, the result is stored at `/Classification/<classifier name>/read_x/` and includes the best class, best score, and the assigned class based on a score threshold.

FAST5 compression is handled by calling `h5repack` on the files. Currently gzip compression is being used since it is built in to the HDF5 library, but FAST5-specific VBZ compression by Oxford Nanopore Technologies should be used in the future.

The most recent version of the specification can be found in the Poretitioner GitHub repository.

Experimental methods

The following text in this section is an excerpt from the NanoporeTERs publication [13]:

NanoporeTER construction, cloning, expression, and purification: The initial NanoporeTER protein was constructed with a gBlock (Integrated DNA Technologies) composed of the Smt3 and tail sequence and cloned into plasmid pCDB180 downstream of the OsmY domain. The Q5 site-directed mutagenesis method (New England Biolabs) was used to generate the different NTER barcode mutants. All cloning was performed using the 5-alpha competent E. coli strain following NEB's cloning protocol (New England Biolabs). Sequence verification was obtained through Genewiz Inc. Expression of the NanoporeTER protein was done in BL21 (DE3) E. coli strain using Overnight Express instant TB medium (Novagen). Proteins were purified via immobilized metal affinity chromatography (IMAC) using TALON metal affinity cobalt resin (Takara). The purification used the associated buffer set from Takara, following their specified protocol. Proteins were concentrated using Amicon Ultra 0.5 mL centrifugal filters with Ultracel 30K (Amicon). The final concentration of proteins averaged 7 mg/ml from 5 mL overnight cultures. The purified proteins were stored for long-term storage at -80C in 10 uL aliquots, as well as for short-term storage at 4C.

E.coli raw culture mixing experiments: Cultures were picked from single colonies on plates and used to inoculate 3mL LB supplemented with 0.5 mM IPTG and kanamycin (induced), or 3mL LB supplemented with 0.2% glucose and kanamycin (inhibited). After

overnight incubation at 37C with shaking, cultures were equally mixed together in a total volume of 45uL, 50uL 4X C17 buffer (2 M KCl, 100 mM HEPES, pH 8), and 105 uL water (total volume 200uL). This solution was then immediately loaded into a MinION flow cell for analysis.

E. coli expression time course: Time course experiments were performed by diluting 30uL of overnight cultures (LB) into 3mL fresh LB supplemented with 0.5 mM IPTG and kanamycin (induced), or 3mL fresh LB supplemented with 0.2% glucose and kanamycin (inhibited). The cultures were placed in a shaker/incubator at 37C to allow for culture growth. Samples were then collected at 2, 4, 6, and 21-hour time-points. At each time-point, cultures were equally mixed together in a total volume of 10 uL, 50uL 4X C17 buffer, and 140 uL water (total volume 200uL). This solution was then immediately loaded into a MinION flow cell for analysis.

MinION experiments: All experiments were performed with unmodified R9.4.1 MinION flow cells (Oxford Nanopore Technologies) by diluting analyte solution into C17 buffer for a final concentration of 0.5M KCl and 25mM HEPES (pH 8), into the flow cell priming port. Flow cells were run on the MinION at a temperature of 30°C and a run voltage of -180mV with a 10khz sampling frequency and 15 second static flip frequency. Use of a modifiable MinKNOW script (available from ONT) enabled voltage flipping cycle parameters to be set as well as collection of raw current data across the entire run. Individual flow cells could be reused for different analytes after flushing them with 1mL C17 buffer three times between experiments. Flow cells were stored at 4°C in C18 buffer (150mM potassium ferrocyanide, 150mM potassium ferricyanide, 25mM potassium phosphate, pH8) when not in use.

4.2.5 Discussion

Overall, Poretitioner provides drastic improvements in usability. Previously, NanoporeTER methods were a collection of Jupyter notebooks and untested python modules — reproducibility was handled through creating a new notebook for each run, as a self-documenting

approach that served the lab well for several years. However, results were recorded in too many large, disparate files, some of which could only be read by Python 2.7 (deprecated in January 2020). The new peptide FAST5 files record more metadata about how the results contained within it were produced, and all results are recorded directly in these files. There is also a single entry point for the full pipeline running on the command line, or it can be used as a module with python bindings.

Poretitioner's storage and memory savings reduce the system burdens of processing nanopore ionic current data. The data footprint created by segmenting, filtering, classifying, and quantifying the data is reduced by approximately ∼80% (or ∼60% if the original data is included), while also improving co-location of ionic current data and pipeline results in the same peptide FAST5 files. Memory in particular is a bottleneck for running this pipeline on a local machine rather than a server. While Poretitioner reduces the memory requirement by ∼40%, reducing this is a priority for future versions. The runtime is similar for both versions, aside from copying sub-runs in the NanoporeTER methods. Although the reduction of classification time from 15 hours to 15 minutes was not a part of Poretitioner, finding and fixing this inefficiency has a significant impact on the rate of data processing, reducing the classification time from "overnight" to "coffee break."

I also improved reproducibility by storing the ionic current data for each read alongside the results of each pipeline step and metadata used to execute each step (e.g. filter parameters). This way, the results and the data that produced them are never separated. Thus, peptide FAST5 files can outlive the tenure of their parent graduate student. This also avoids local recalculation of the data, i.e. when someone is not sure how a file was created so they generate their own local version.

Contributions

Work for Poretitioner was carried out by myself and Jessica Dunstan, using previous code carried out for the NanoporeTER publication by myself, Jeff Nivala, and Karen Zhang.

Specifically, my contributions include refactoring and improving the aspects of the pipeline

that produced the scientific results (i.e. the individual steps of segmenting, filtering, classifying, and quantifying data), including validating Poretitioner's results against NTER data. Jessica Dunstan provided significant contributions to the infrastructure of Poretitioner.

I also defined the new peptide FAST5 specification for general purpose usage (non-DNA sequencing) based on reverse-engineering DNA FAST5 files and adding required functionality.

Future work

In the future, more implementation of parallelization and streaming could provide drastic gains in performance, especially on limited resource machines. Additionally, since selective sequencing is now publicly available, modifying the segmentation and quantification to allow for arbitrarily timed voltage cycling will be necessary (as opposed to the current process in which all channels cycle voltage concurrently). A new FAST5-specific compression type, VBZ compression, can also be implemented for improvements in disk storage space.

New modules could also be added to Poretitioner, including a visualization suite. There has already been interest by others in creating visualization tools for peptide data; adding these tools to Poretitioner would be fairly straightforward. Another potential improvement could be adding support for training new classification models, possibly building upon the Pytorch Fast5Dataset discussed as part of Big Bits in Chapter 3.

4.3 Summary

Poretitioner provides a robust tool set for analyzing non-DNA analytes run on MinION nanopore devices. This fills a niche in nanopore-based molecular sensing techniques and supports advancements in reporter protein engineering. Molecular sensing on nanopores collects large amounts of data (∼15-20 GB per hour), but only part of the ionic current trace actually contains captured molecules to analyze. Poretitioner extracts the ionic current trace for these captured molecules, saves the data as reads in new peptide FAST5 files, and supports downstream analysis. This downstream analysis can include identifying (i.e. classifying) peptides present in unpurified or unknown samples, calculating concentration of

the analyte, and in the future, possibly visualize nanopore data.

The pipeline is based on previously developed work [13]. The NanoporeTERs in this publication use peptide barcodes, but the technology is not inherently limited to peptides. Other applications in the future will require modifications to the segmentation approach and other steps in the pipeline, which is made easier by Poretitioner.

Poretitioner enables rapid development of molecular engineering techniques and creation of new nanopore-addressable reporter proteins by reducing the complexity and technical overhead of processing the data generated. Often, a new technique means that the segmentation method must be tweaked, the threshold lowered, or different filters must be implemented. This is a challenge that is often contradictory to a system meant to process many datasets consistently to generate results for a paper. All parameters in Poretitioner can be easily tuned on the command line or in a configuration file instead of set as a mutable variable in a notebook. Therefore, these parameters can be changed as quickly as the experiment.

Additionally, providing publicly available tools will promote external adoption of techniques developed in the lab. At the time of writing, there was already one known instance of a lab that wanted to use NanoporeTERs, but found that they were unable to fully use the pipeline as it previously existed. Interest in tools to support NanoporeTERs will likely increase upon publication (currently in review).

Chapter 5

CONCLUSIONS

This thesis presents novel computational methods for nontraditional applications of nanopores, extending beyond genomic DNA sequencing. Direct manipulation of raw nanopore signals opens up exciting new applications and possibilities for nanopore sensing. Despite increased commercial adoption of the MinION, few methods were previously available for raw nanopore signal design and manipulation beyond basecalling, and none existed for molecular tagging, extended sequential encoding of nanopore-orthogonal molecular bits, or peptide sensing on the MinION. Through this work, I:

1. Developed Porcupine, a molecular tagging system that tags physical items using synthetic DNA that I custom-designed to produce visually distinct nanopore signals. I executed this project from conception to producing real, reliable molecular tags and a method to read them.

2. Presented Big Bits, a proof of concept for a method of sequentially encoding molecular bits in long strands for DNA data storage.

3. Implemented a pipeline to support sensing peptide barcodes and other molecules in commercially available nanopore devices.

Porcupine democratizes access to molecular tags, removing expensive barriers found in other molecular tagging systems like the need for 2nd-generation sequencers, introducing portability and real-time decoding, and allowing tag creation to be brought in-house. This culminated in a patent application to protect the commercial interest (16/879,214).

Although Big Bits is still in an early testing stage, it has the potential to enable applications of medium-to-large scale DNA data storage, where non-specialized labs can store data at lower costs without having access to large-scale DNA synthesis. Further investigation needs to be done to tune the size and distinctiveness of the molbits, but early work is promising.

Poretitioner enables a wide array of future work in both the MISL lab and other labs, making this new field of protein sensing on the MinION more standardized and approachable. The range of potential uses for commercial nanopore arrays is quite broad; protein sequencing would be among the most ambitious. Infrastructure tools like Poretitioner can support rapid exploration in this area by reducing the computational and technical barriers to entry.

Each of these methods is the first of its kind in its application, from nanopores in molecular tagging, crossing molecular tags with DNA data storage, and computationally enabling analysis of an entirely different class of molecules on commercial nanopore devices.

BIBLIOGRAPHY

[1] Enis Afgan, Dannon Baker, Bérénice Batut, Marius Van Den Beek, Dave Bouvier, Martin Ech, John Chilton, Dave Clements, Nate Coraor, Björn A. Grüning, Aysam Guerler, Jennifer Hillman-Jackson, Saskia Hiltemann, Vahid Jalili, Helena Rasche, Nicola Soranzo, Jeremy Goecks, James Taylor, Anton Nekrutenko, and Daniel Blankenberg. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. *Nucleic Acids Research*, 46(W1):W537–W544, 2018.

[2] Nada M. A Al-salami. Evolutionary Algorithm Definition. *Applied Sciences*, 2(4):789–795, 2009.

[3] Leon Anavy, Inbal Vaknin, Orna Atar, Roee Amit, and Zohar Yakhini. Data storage in DNA with fewer synthesis cycles using composite DNA letters. *Nature Biotechnology*, 37(10):1229–1236, 10 2019.

[4] Christopher E. Arcadia, Eamonn Kennedy, Joseph Geiser, Amanda Dombroski, Kady Oakley, Shui Ling Chen, Leonard Sprague, Mustafa Ozmen, Jason Sello, Peter M. Weber, Sherief Reda, Christopher Rose, Eunsuk Kim, Brenda M. Rubenstein, and Jacob K. Rosenstein. Multicomponent molecular memory. *Nature Communications*, 11(1), 2020.

[5] Sercan Arik, Gregory Diamos, Andrew Gibiansky, John Miller, Kainan Peng, Wei Ping, Jonathan Raiman, and Yanqi Zhou. Deep Voice 2: Multi-Speaker Neural Text-to-Speech. *Advances in Neural Information Processing Systems*, 2017-December:2963–2971, 5 2017.

[6] C. Bancroft. Long-Term Storage of Information in DNA. *Science*, 293(5536):1763–1765, 2001.

[7] Amir Barati Farimani, Mohammad Heiranian, and Narayana R. Aluru. Identification of amino acids with sensitive nanoporous MoS2: towards machine learning-based prediction. *npj 2D Materials and Applications*, 2(1):14, 12 2018.

[8] Nicholas A.W. Bell and Ulrich F Keyser. Specific protein detection using designed DNA carriers and nanopores. *Journal of the American Chemical Society*, 137(5):2035–2041, 2015.

[9] Meinolf Blawat, Klaus Gaedke, Ingo Hütter, Xiao Ming Chen, Brian Turczyk, Samuel Inverso, Benjamin W. Pruitt, and George M. Church. Forward error correction for DNA data storage. In *Procedia Computer Science*, volume 80, pages 1011–1022, 2016.

[10] James Bornholt, Douglas M Carmean, Luis Ceze, Georg Seelig, and Karin Strauss. Toward A DNA-Based Archival Storage System. *IEEE Micro*, pages 98–104, 2017.

[11] Rowena A. Bull, Thiruni N. Adikari, James M. Ferguson, Jillian M. Hammond, Igor Stevanovski, Alicia G. Beukers, Zin Naing, Malinna Yeang, Andrey Verich, Hasindu Gamaarachchi, Ki Wook Kim, Fabio Luciani, Sacha Stelzer-Braid, John Sebastian Eden, William D. Rawlinson, Sebastiaan J. van Hal, and Ira W. Deveson. Analytical validity of nanopore sequencing for rapid SARS-CoV-2 genome analysis. *Nature Communications*, 11(1):1–8, 12 2020.

[12] Brian J. Cafferty, Alexei S. Ten, Michael J. Fink, Scott Morey, Daniel J. Preston, Milan Mrksich, and George M. Whitesides. Storage of Information Using Small Organic Molecules. *ACS Central Science*, 5 2019.

[13] Nicolas Cardozo, Karen Zhang, Katie Doroschak, Aerilynn Nguyen, Zoheb Siddiqui, Karin Strauss, Luis Ceze, and Jeff Nivala. Multiplexed direct detection of barcoded protein reporters on a nanopore array. *bioRxiv*, pages 1–12, 2019.

[14] Luis Ceze, Jeff Nivala, and Karin Strauss. Molecular digital data storage using DNA. *Nature Reviews Genetics*, 20(8):456–466, 2019.

[15] Amy E. Chavis, Kyle T. Brady, Grace A. Hatmaker, Christopher E. Angevine, Nuwan Kothalawala, Amala Dass, Joseph W.F. F Robertson, and Joseph E. Reiner. Single Molecule Nanopore Spectrometry for Peptide Detection. *ACS Sensors*, 2017.

[16] Yeongjae Choi, Taehoon Ryu, Amos C. Lee, Hansol Choi, Hansaem Lee, Jaejun Park, Suk Heung Song, Seojoo Kim, Hyeli Kim, Wook Park, and Sunghoon Kwon. High information capacity DNA-based data storage with augmented encoding characters using degenerate bases. *Scientific Reports*, 9(1):1–7, 12 2019.

[17] Paolo DI Tommaso, Maria Chatzou, Evan W. Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. Nextflow enables reproducible computational workflows. *Nature Biotechnology*, 35(4):316–319, 2017.

[18] Kathryn Doroschak, Karen Zhang, Melissa Queen, Aishwarya Mandyam, Karin Strauss, Luis Ceze, and Jeff Nivala. Rapid and robust assembly and decoding of molecular tags with DNA-based nanopore signatures. *Nature Communications*, 11(1), 2020.

[19] Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison. *Biological Sequence Analysis*. Cambridge University Press, 4 1998.

[20] Sean R. Eddy. What is a hidden Markov model? *Nature Biotechnology*, 22(10):1315–1316, 10 2004.

[21] Carola Engler and Sylvestre Marillonnet. Combinatorial DNA assembly using golden gate cloning. *Methods in Molecular Biology*, 1073:141–156, 2013.

[22] Carl W. Fuller, Lyle R. Middendorf, Steven A. Benner, George M. Church, Timothy Harris, Xiaohua Huang, Stevan B. Jovanovich, John R. Nelson, Jeffery A. Schloss, David C. Schwartz, and Dmitri V. Vezenov. The challenges of sequencing by synthesis. *Nature Biotechnology*, 27(11):1013–1023, 11 2009.

[23] Arkarachai Fungtammasan, Guruprasad Ananda, Suzanne E. Hile, Marcia Shu Wei Su, Chen Sun, Robert Harris, Paul Medvedev, Kristin Eckert, and Kateryna D. Makova.

Accurate typing of short tandem repeats from genome-wide sequencing data and its applications. *Genome Research*, 125(5):736–749, 5 2015.

[24] Nick Goldman, Paul Bertone, Siyuan Chen, Christophe Dessimoz, Emily M. Leproust, Botond Sipos, and Ewan Birney. Towards practical, high-capacity, low-maintenance information storage in synthesized DNA. *Nature*, 494(7435):77–80, 2013.

[25] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *ACM International Conference Proceeding Series*, volume 148, 2006.

[26] Alex Graves, Abdel Rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2013.

[27] Renmin Han, Sheng Wang, Xin Gao, and Bonnie Berger. Novel algorithms for efficient subsequence searching and mapping in nanopore raw signals towards targeted sequencing. *Bioinformatics*, 36(5):1333–1343, 2020.

[28] Awni Hannun. Sequence Modeling with CTC. *Distill*, 2017.

[29] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8), 1997.

[30] Miten Jain, Hugh E. Olsen, Benedict Paten, and Mark Akeson. The Oxford Nanopore MinION: Delivery of nanopore sequencing to the genomics community. *Genome Biology*, 17(1), 2016.

[31] David Jakubosky, Erin N. Smith, Matteo D'Antonio, Marc Jan Bonder, William W. Young Greenwald, Agnieszka D'Antonio-Chronowska, Hiroko Matsui, Marc J. Bonder, Na Cai, Ivan Carcamo-Orive, Matteo D'Antonio, Kelly A. Frazer, William W. Young Greenwald, David Jakubosky, Joshua W. Knowles, Hiroko Matsui, Davis J. McCarthy, Bogdan A. Mirauta, Stephen B. Montgomery, Thomas Quertermous, Daniel D.

Seaton, Craig Smail, Erin N. Smith, Oliver Stegle, Oliver Stegle, Stephen B. Montgomery, Christopher DeBoever, and Kelly A. Frazer. Discovery and quality analysis of a comprehensive set of structural variants and short tandem repeats. *Nature Communications*, 11(1):1–15, 12 2020.

[32] Sarah S. Johnson, Elena Zaikova, David S. Goerlitz, Yu Bai, and Scott W. Tighe. Real-time DNA sequencing in the antarctic dry valleys using the Oxford nanopore sequencer. *Journal of Biomolecular Techniques*, 2017.

[33] Stephen C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3), 1967.

[34] Max J Kellner, Jeremy G Koob, Jonathan S Gootenberg, Omar O. Abudayyeh, and Feng Zhang. SHERLOCK: nucleic acid detection with CRISPR nucleases. *Nature Protocols*, 14(10):2986–3012, 2019.

[35] Benjamin Kleeman, Andre Olsson, Tess Newkold, Matt Kofron, Monica DeLay, David Hildeman, and H. Leighton Grimes. A guide to choosing fluorescent protein combinations for flow cytometric analysis based on spectral overlap. *Cytometry Part A*, 93(5):556–562, 5 2018.

[36] Julian Koch, Silvan Gantenbein, Kunal Masania, Wendelin J. Stark, Yaniv Erlich, and Robert N. Grass. A DNA-of-things storage architecture to create materials with embedded memory. *Nature Biotechnology*, 38(1), 2020.

[37] Kaname Kojima, Yosuke Kawai, Kazuharu Misawa, Takahiro Mimori, and Masao Nagasaki. STR-realigner: a realignment method for short tandem repeat regions. *BMC Genomics*, 17:991, 2016.

[38] Mikhail Kolmogorov, Eamonn Kennedy, Zhuxin Dong, Gregory Timp, and Pavel A. Pevzner. Single-molecule protein identification by sub-nanopore sensors. *PLoS Computational Biology*, 13(5), 2017.

[39] Johannes Köster and Sven Rahmann. Snakemake-a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19):2520–2522, 2012.

[40] Sriram Kosuri and George M. Church. Large-scale de novo DNA synthesis: Technologies and applications. *Nature Methods*, 11(5):499–507, 4 2014.

[41] Raga Krishnakumar, Anupama Sinha, Sara W. Bird, Harikrishnan Jayamohan, Harrison S. Edwards, Joseph S. Schoeniger, Kamlesh D. Patel, Steven S. Branda, and Michael S. Bartsch. Systematic and stochastic influences on the performance of the MinION nanopore sequencer across a range of nucleotide bias. *Scientific Reports*, 2018.

[42] Heng Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv*, 3 2013.

[43] Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *arXiv*, 2017.

[44] Jun Li, Haoqiu Wang, Lingfeng Mao, Hua Yu, Xinfen Yu, Zhou Sun, Xin Qian, Shi Cheng, Shuchang Chen, Junfang Chen, Jingcao Pan, Jueliang Shi, and Xuchu Wang. Rapid genomic characterization of SARS-CoV-2 viruses from clinical specimens using nanopore sequencing. *Scientific Reports*, 10(1):1–10, 12 2020.

[45] San Ling and Chaoping Xing. *Coding theory: a first course*. Cambridge University Press, 2004.

[46] Matthew Loose, Sunir Malla, and Michael Stout. Real-time selective sequencing using nanopore technology. *Nature Methods*, 2016.

[47] Randolph Lopez, Yuan Jyue Chen, Siena Dumas Ang, Sergey Yekhanin, Konstantin Makarychev, Miklos Z. Racz, Georg Seelig, Karin Strauss, and Luis Ceze. DNA assembly for nanopore data storage readout. *Nature Communications*, 10(1), 2019.

[48] Gediminas Mikutis, Claudia A. Deuber, Lucius Schmid, Anniina Kittilä, Nadine Lobsiger, Michela Puddu, Daphne O. Asgeirsson, Robert N. Grass, Martin O. Saar, and

Wendelin J. Stark. Silica-Encapsulated DNA-Based Tracers for Aquifer Characterization. *Environmental Science and Technology*, 52(21), 2018.

[49] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[50] Jeff Nivala, Douglas B Marks, and Mark Akeson. Unfoldase-mediated protein translocation through an α-hemolysin nanopore. *Nature Biotechnology*, 31(3):247–250, 2013.

[51] Lee Organick, Bichlien H. Nguyen, Rachel McAmis, Weida D. Chen, A. Xavier Kohll, Siena Dumas Ang, Robert N. Grass, Luis Ceze, and Karin Strauss. An Empirical Comparison of Preservation Methods for Synthetic DNA Data Storage. *Small Methods*, page 2001094, 1 2021.

[52] Oxford Nanopore Technologies. Mako: https://github.com/nanoporetech/mako, 2019.

[53] Oxford Nanopore Technologies. Scrappie: https://github.com/nanoporetech/scrappie, 2019.

[54] Poornima Parameswaran, Roxana Jalili, Li Tao, Shadi Shokralla, Baback Gharizadeh, Mostafa Ronaghi, and Andrew Z. Fire. A pyrosequencing-tailored nucleotide barcode design unveils opportunities for large-scale sample multiplexing. *Nucleic Acids Research*, 35(19), 2007.

[55] Alexander Payne, Nadine Holmes, Thomas Clarke, Rory Munro, Bisrat J. Debebe, and Matthew Loose. Readfish enables targeted nanopore sequencing of gigabase-sized genomes. *Nature Biotechnology*, 2020.

[56] Vladimir Potapov, Jennifer L. Ong, Rebecca B. Kucera, Bradley W. Langhorst, Katharina Bilotti, John M. Pryor, Eric J. Cantor, Barry Canton, Thomas F. Knight, Thomas C. Evans, and Gregory J.S. Lohman. Comprehensive Profiling of Four Base

Overhang Ligation Fidelity by T4 DNA Ligase and Application to DNA Assembly. *ACS Synthetic Biology*, 7(11), 2018.

[57] Jason Qian, Zhi-xiang Lu, Christopher P Mancuso, Han-ying Jhuang, Rocío Carmen Barajas-ornelas, Sarah A Boswell, Fernando H Ramírez-guadiana, Victoria Jones, Akhila Sonti, Kole Sedlack, Lior Artzi, Giyoung Jung, Mohammad Arammash, Mary E Pettit, Michael Melfi, Lorena Lyon, Siân V Owen, Michael Baym, Ahmad S Khalil, Pamela A Silver, David Z Rudner, and Michael Springer. Barcoded microbial system for high-resolution object provenance. *Science*, 368(June):1135–1140, 2020.

[58] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '12*, 2012.

[59] Jacob Schreiber and Kevin Karplus. Segmentation of noisy signals generated by a nanopore. *bioRxiv*, 2015.

[60] Martin A. Smith, Tansel Ersavas, James M. Ferguson, Huanle Liu, Morghan C Lucas, Oguzhan Begik, Lilly Bojarski, Kirston Barton, and Eva Maria Novoa. Barcoding and demultiplexing Oxford Nanopore native RNA sequencing reads with deep residual learning. *bioRxiv*, 2019.

[61] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

[62] Ashley Stephenson, Max Willsey, Jeff McBride, Sharon Newman, Bichlien Nguyen, Christopher Takahashi, Karin Strauss, and Luis Ceze. PurpleDrop: A digital microfluidics-based platform for hybrid molecular-electronics applications. *IEEE Micro*, 40(5), 2020.

[63] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, volume 4, pages 3104–3112, 2014.

[64] Haibao Tang, Ewen F. Kirkness, Christoph Lippert, William H. Biggs, Martin Fabani, Ernesto Guzman, Smriti Ramakrishnan, Victor Lavrenko, Boyko Kakaradov, Claire Hou, Barry Hicks, David Heckerman, Franz J. Och, C. Thomas Caskey, J. Craig Venter, and Amalio Telenti. Profiling of Short-Tandem-Repeat Disease Alleles in 12,632 Human Whole Genomes. *American Journal of Human Genetics*, 101(5):700–715, 11 2017.

[65] Joshua Timmons. seqfold: https://github.com/Lattice-Automation/seqfold, 2021.

[66] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 2017-December, 2017.

[67] Ryan R. Wick, Louise M. Judd, and Kathryn E. Holt. Deepbinner: Demultiplexing barcoded Oxford Nanopore reads with deep convolutional neural networks. *PLoS Computational Biology*, 14(11), 11 2018.

[68] Ryan R. Wick, Louise M. Judd, and Kathryn E. Holt. Performance of neural network basecalling tools for Oxford Nanopore sequencing. *Genome Biology*, 20(1), 2019.

[69] Joseph N. Zadeh, Conrad D. Steenberg, Justin S. Bois, Brian R. Wolfe, Marshall B. Pierce, Asif R. Khan, Robert M. Dirks, and Niles A. Pierce. NUPACK: Analysis and design of nucleic acid systems. *Journal of Computational Chemistry*, 2011.

[70] Michael Zuker and Patrick Stiegler. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic Acids Research*, 9(1), 1981.

Appendix A

SUPPLEMENTARY INFORMATION: PORCUPINE

A.1 Supplementary Methods

Evaluating read count variation

The largest source of bit errors in Porcupine can be attributed to the unusually large but seemingly consistent read count variance. Since each molecular tag is composed of equal proportions of each present molbit, the resulting read counts for each molbit after sequencing should be approximately equal; but instead, I measured significant variation in read counts. I first checked for obvious sequence-related issues by correlating quantifiable sequence characteristics, including GC content, minimum free energy (MFE), and homopolymer length, against read counts normalized as described in the main text. Evidence shows GC bias in genomic bacterial nanopore sequencing data on the MinION [41], based on non-stochastic bias introduced by the basecaller, not the sequencing device. With respect to MFE, I reasoned that although I controlled for MFE in the design phase, correlation between folding energy and read counts could still explain some variation. Additionally, I considered that the inclusion of long homopolymers could also have been associated with a reduction in read counts since imprecise calls of homopolymer length can potentially cause poor sequence alignment and therefore artificially low read counts [68]. I also checked for occurrences of the 4-base overhangs used for Golden Gate assembly (GAGT and GCTG), and found no notable trends. As shown in Appendix Figure A.9a, correlations between these various measures and normalized read counts were quite low (Pearson correlation (two-sided) for GC: p = 0.31, r=-0.10; MFE: p=0.33, r=0.09; homopolymer length: p=0.82, r=0.02; GCTG presence: p=0.90, r=0.01; GAGT presence: p=0.42, r=-0.08). Thus, these rough measures could not explain the source of read count variance.

Moving beyond these initial sequence bias analyses, I reasoned that read count variation errors can be introduced at four different points in the system: (1) DNA synthesis, (2) strand assembly and tag combination, (3) sequencing, or (4) analysis/labeling.

(1) To check whether there was a problem with the original DNA order, I re-ordered 15 molbits representing the five lowest, average, and highest read count molbits. When sequenced, the read counts of the re-ordered sequences were similar in proportion to their previous counts (Appendix Figure A.9b). Thus, if the source of variability is caused by a synthesis error, which I believe is unlikely, it is at least reproducible or possibly dependent on an unexplored aspect of sequence content.

(2) To evaluate the strand assembly and tag combination step, I examined potential secondary structure in individual molbits, specifically the same molbits from the re-synthesized tests. During the Golden Gate strand assembly step, double stranded DNA is separated into single strands, potentially enabling secondary structure formation. Sequences were screened for minimum free energy during the evolutionary modeling phase, but the screening did not cover all forms of secondary structure, particularly stem-loop structures with short stems and large loops. Some secondary structure can be seen across all three categories (see Appendix Figure A.10). It is possible that secondary structure could have lowered read counts, but I do not believe that it could have caused excessive read counts.

Additionally, I reasoned about potential experimental variation due to human error. When combining tags, some variation can be explained by pipetting. Here, I reasoned that up to 2-3x variation could be explained by this factor (bubbles, calibration, improperly depressed pipette, etc.). Also, if a bit were omitted, it would cause a near-zero or very low read count, but if a bit were accidentally added, it alone would not cause such a dramatic overabundance of read counts. Thus, human error of this type could cause a low read count but likely not an overly high one.

(3) It is possible the systematic sequencing errors contribute to molbit read differences, however, the strong correlation between basecalling and CNN-based molbit classification makes this less likely. Future experiments using a different sequencing platform (e.g. Illumina

or Sanger), could help to resolve this question.

(4) I do not believe basecalling or labeling is a significant contributor to the problem. With respect to basecalling, an average of 93.3% of reads could be basecalled with high confidence (Q-score ≥ 9) throughout all runs, leaving just $\sim$7% room for variation (compared to 200-300% variation overall).

Also note that the molbit counts for labels assigned via basecalling and alignment have high correlation with those assigned by the CNN (Pearson two-sided $p < 10^{-5}$, r=0.9998, Appendix Figure A.4). Although there is certainly bias in that figure since basecalled labels are used to train the CNN, this high correlation does reduce the possibility that either method is introducing a significant number of spurious calls.

Ultimately, the method was designed to be robust to these types of errors; however, I still sought to reduce them since any reduction in bit errors can exponentially reduce the overall decoding error rate.

A.2 Supplementary Tables

Molbit	Sequence	Molbit	Sequence
0	TTTATAGAAAACGTTTTGAAGAAGAAGATGATCTCTACTC	48	AGTTATCCACACACACCGCGCGTCACCTGTTTACGCACTC
1	TGTCCTACTATGTCTTCTCTCTTCTACTACTTACCTACTC	49	TGAGGAGAGAGAGAGAAGCCCTACCTACCACACATAACTC
2	GGATGGATGATCCCACACCTCACACGCAGGAGAGAAACTC	50	TTCGGTTTGCACCTCTCCCACACCCAAAAGGTTTGTACTC
3	CTAGTGGTAGATGTTGTGTGTGGCGCGAGAGAAAGCACTC	51	AATGTCGTTGTTGCTATCACACGTTACGCTGTTCCTACTC
4	TTGCGACGATGACTGACGACTGCACGAAAAGCTGGAACTC	52	CTGGAAAGCAGATCAGACCCACACCCTACACACACGACTC
5	GTGAGGAGGAGAAGTAAAAGAAAGCTTCGAGAGAGTACTC	53	GGGAGGGCTTCACGCACGTTTTACCACCCTATCTCCACTC
6	AGTTTACACGGCGCTCTTTCCGGTTTGATCTTGCACACTC	54	ATTAATCGATATTTTACACGCACGCACTACAGTTTAACTC
7	CTGTTTGCACACACACCCGCACACCCTGTTCCCTCGACTC	55	CGAGAAAACTCCACACCACCCGGTCCCTCCACACAGACTC
8	ATGCGTTGCGTTGTTTTGCGTTCCACACCACACGTTACTC	56	CGCGTGTGTTATGTGTCATCCCACTACACACACTGGACTC
9	ATGTTTACGCACGCGTTTTCCCACCCACGATGTTGTACTC	57	TTGAGGATGGGTTTTACCCTACTTTGCGCACACTCTACTC
10	ATCCAAAGAGAACTGGGATTTCTAAAAGAGAGAGAAACTC	58	AGTTCGTCGTCGTTTACCCACACCACACACACACACACTC
11	AGTCCCACCTCACGAGGAAAGAGAGGGAGGAGAGAAACTC	59	TTCTTGTTTTACACTGATCTTCTACCACACACCACGACTC
12	GTTTTACCTTTTGCGCTTTTGTCTTCGTTCGTCCCTACTC	60	CTCGAAGGGATCACTTTGCACAAAAGGAAGTCGCACACTC
13	GGCTCCCTACCACACACCACGTTTTGATGATAGTTGACTC	61	ACGGAGGAGGAAACTCAAAAGATCATACCCGCACTTACTC
14	TACGAGAGGAAGCTTCACACACCACCACGATCGGATACTC	62	AGAAGGGCTATCGAAAAGAGAGAGGTGAAGAGAGAAACTC
15	CTTGCGCACACCTCACACACGTGTTTGTGTTGTGTTACTC	63	ATCACACCTCGCACTCTTTGGGACGTGGCATGATGGACTC
16	CGATCCGCACACGCACGTCACACCTATCTTACGTGTACTC	64	GCCTACCACATGTGTGTTGTCGTGTGTTTCCTCGCGACTC
17	GAAGAAAAGAAAGAGAAGAGAAAACTCAAAAGATGAACTC	65	ATCAACGTTTGCGCACACGCAACTTTTGCGCAAATGACTC
18	GCCTCATATCAGTTCGGGTTTGCACTTTGGCACTGCACTC	66	GGTTTACCTAAAGAGAGGAGGGAATCTATAATACATACTC
19	TGATGTTTTGCCACGCCGCGCGTAGTAGTGTGTCCTACTC	67	CAGCAGTCCACACTACACACGGCTTTTCCCGCACTAACTC
20	GAGCACACCTCGTTTTACAAAAGATGGTTGTGTTGTACTC	68	TTCCGCTTTTGTTGTTGTATGTCTTTTACTGTTCCTACTC
21	GCCAGGTTTACCCTAATAATGTCATCTCCGCACCTTACTC	69	TGATGGAGGGAGGAGAGGAGGAAGCACACCCTCAACTC
22	CAGCTGGATGCGAGAGGGAAGCTGGTTGAAATAATAACTC	70	TATTCACCTGTACCCTCGAAGAGGTCCGAAAGAAAGACTC
23	GGACACTACCTACACTACCTACCCTATTTTATTTTGACTC	71	ATCTACCACGAGAGGTTTTGCGCACCTCGTTCACTGACTC
24	TGCGAAGAGGGAGGAGGAAACTCGGGACTTTTACGCACTC	72	TGATCACGTTTTACTACCTACACACTACCACCTACTACTC
25	AGTTTACGATCGAAAAGGCTTCCACACCACAACGGGACTC	73	ATCCTGCGCACGCGCACACCCTCTCTCTCACCTTTGACTC
26	CTACTACTACCTCCCTCTCTCACCCGCGTTGGTCTGACTC	74	GGACGCTTTTACGTTTGCCCTACACCCACCACGTTTACTC
27	TTCCACGCTTTACACACACACCACTGTCGTCGTGTCACTC	75	ATCGAAAAGCCCACCGGAGAGGAGAGAGAGAGAGGAACTC
28	TCCCACACGAAGAAGTCGACATAAGCAACGGCGACGACTC	76	TTTTACACCTCACACCCACCACAGGGAGGGATATCGACTC
29	AGACTTTGAGGGTGATCGTACGCACACACACACCCTACTC	77	CCACACACCACCTACCTACACACGAAGAGAGAGAGGACTC
30	CTCGTCCCTCTCTACACACTACGTTTTGCTTTGCACACTC	78	TACACACGCACGTAATCGTCAGATCGTCCACACGTAACTC
31	GTCACACACCCTCAATCCCACTGTTTGCGCATCCTTACTC	79	GGAGAGAGAAGCTCGAGGAGAGAGAGAGGGAATCCGACTC
32	AGCATAGCAGCACGTGTCTTTGCGCAGTTTGCACGAACTC	80	ATTACTACTGCGCGTGTTAGTTTTACCACACCTATAACTC
33	TCCTCTACCTACCGCTTTAGTGTTAAGGCTTTACTAACTC	81	GGAAAGAGCGGTACACTCTCCGAGGGAGAGGCCCACACTC
34	GAATCGATGATGAAAGAGAAAGGTTTTACTTTCACGACTC	82	CGTCCCTACGCGCTTTTACGCACGCACCCTGTCTGAACTC
35	GTCGCACACACCACACACGAAAGGTTAGTTGTTTGCACTC	83	TAACGCCTCTTTAGTTTGCGTGTCCTACTACACCTAACTC
36	CTCCAAAGAGAGGAGAGAGAGAAAAGAGAGGGAAGCACTC	84	GGGCTTCACACCACAGATCGAGGACGGCACACACTAACTC
37	GGAACTTTTACGCACGCGCACCCTCAGATGTTTATAACTC	85	GCCTATCTTTACACGCACACACACACGTGTTGTTATACTC
38	GTGCGTGTTGTGGGATCGAGAAAGCTTTTGCACGTAACTC	86	CTCCAACTTTTACGAGAGCAATCAGGAATGGACCAAACTC
39	AGGTGTTGCCTCACTAGTCTTTACACGCGCATTTTAACTC	87	CTCATCTTGCCCGCACCGCACGCTACCACACACGTCACTC
40	TAGCACCCGTTCCTATGTTTGTTGTTTTACACCCACACTC	88	TTTGTATGTTATCATAACACTGTTTGCACCTGCACTACTC
41	CTGATTGCTTCCAAGGGAGAGGGAAGCACACCGAAAACTC	89	CTTAGATGTTTTACGTTTATGTTTTGGTTTTCACTTACTC
42	AGAAGGGATCACGCATCCCTCTCACACGTTTACATAACTC	90	CTCGAGAAGTTGAGAGGTGAGGGTTTGCTTTACCGCACTC
43	CGCGCGTTCGTTTACGCACACCTATCTTTGCTTTATACTC	91	TTCACAAAGAGGAGAGGGAGAGAGAGAGTCAAAGGGACTC
44	ATCTACCACCTACCACGATCCCTGATCAGAAAGAGAACTC	92	CTCCACGCACCGTTTGTCCCACACCCACACCCACACACTC
45	CGCTTTACCCTCGTCCTATTTCACCCTCACACTTTTACTC	93	ATATCTTTGTTTACGTTTTGCACGCGTCCGCGCACAACTC
46	ATCCCACGAGGGAGGAGAAAAGTCCCACACCACAGAACTC	94	GGTTTACGCACCACACACACATTACACCTGCCTATCACTC
47	TTCCGCGCACGCTTTTCTCGTTTTGTCTTTTACACTACTC	95	GGTTTCCACTTTTGTTTTCCCACACGTTTCCCTACCACTC

Table A.1: **Porcupine's 96 designed molbit sequences**

Primer name	Sequence
400mer_F	GCCGGTCTCTGCTGCTTTAATAGTGGACTCTTGTTCCAAACTGG
400mer_R	GCCGGTCTCTGCTGTACCAGGATCTTGCCATCCTATGGAAC
1600mer_F	GCCGGTCTCTGCTGGATCCTTTTTTTCTGCGCGTAATCTGC
1600mer_R	GCCGGTCTCTGCTGGAATGAATCACCGATACGCGAGCG

Table A.2: **Primer sequences used to extract the insert strand from plasmid pCDB180 by PCR.**

A.3 Supplementary Figures

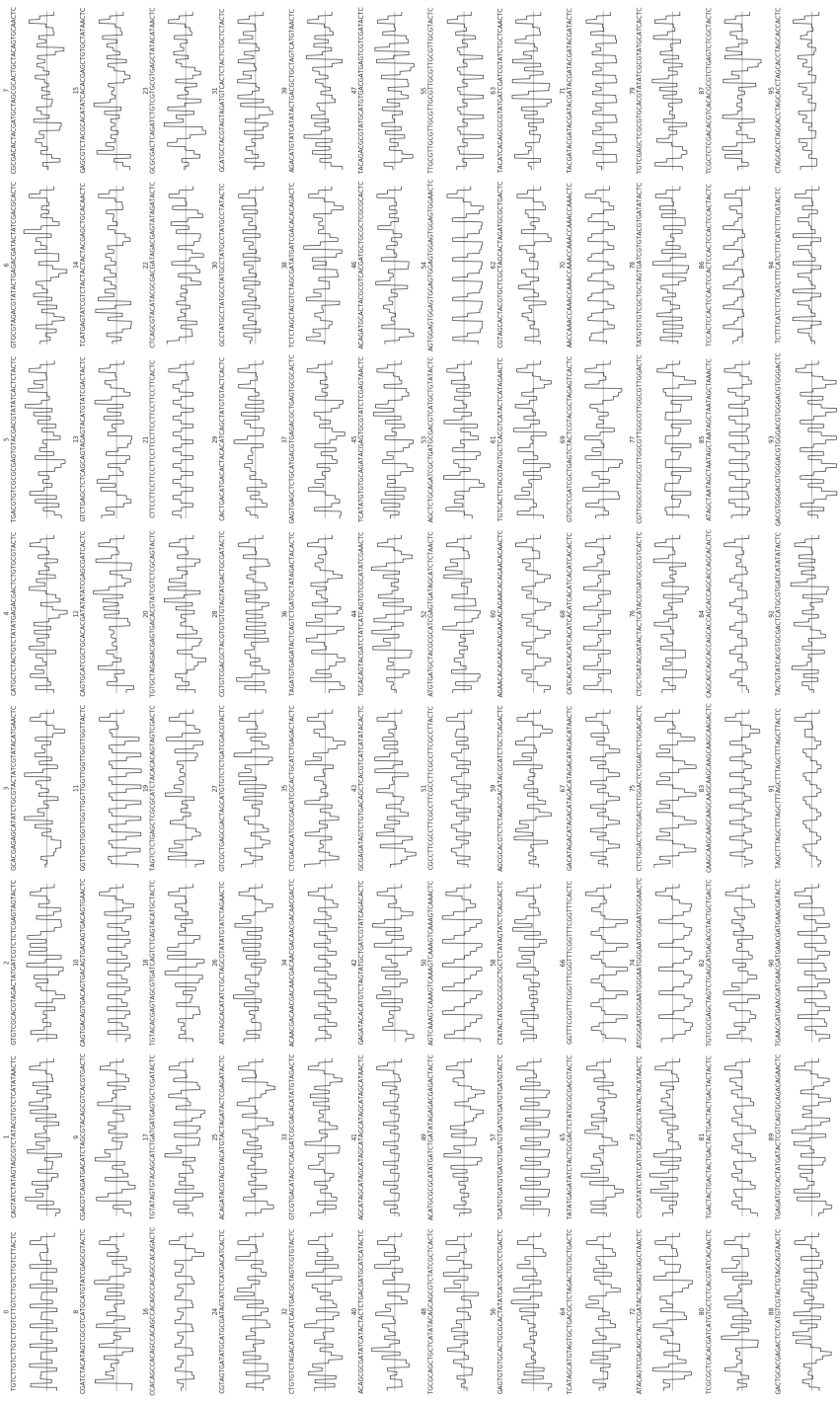

Figure A.1: **All 96 molbit sequences initialized before beginning the design phase using the evolutionary model.**

Figure A.2: **All 96 molbit sequences after 31 iterations of the evolutionary model.**

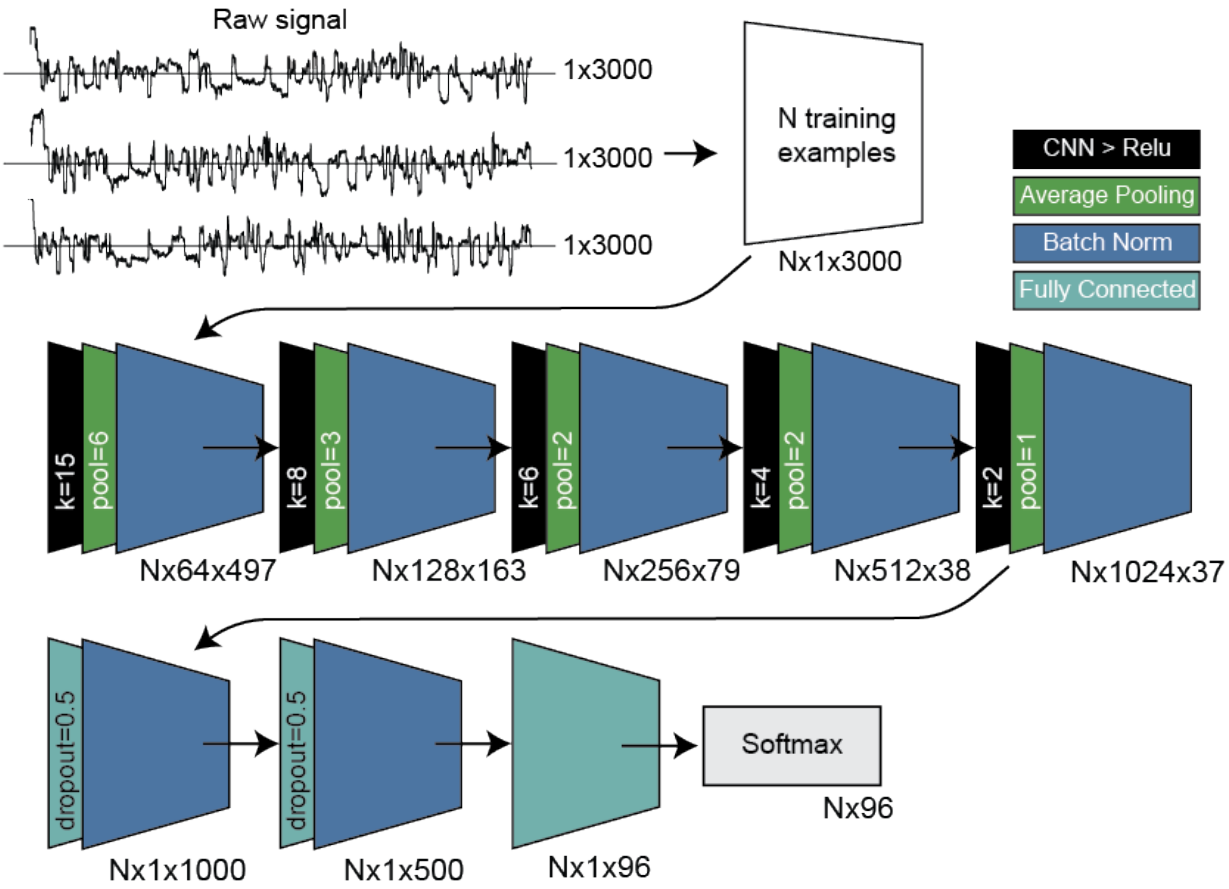

Figure A.3: **Molbit classification model and data flow.** Input data includes nanopore ionic current traces, which were rescaled using a Median Absolute Deviation method modified from Mako, trimmed to remove stalled signal characteristic to the beginning of sequencing reads, and truncated to the first 3000 data points in the signal time series. Rescaled training data then passes through a 5-layer CNN followed by 2 fully connected layers with dropout and a final fully connected layer with softmax as the output layer. In the CNN layers, k is the kernel size and pool is the average pooling kernel size.

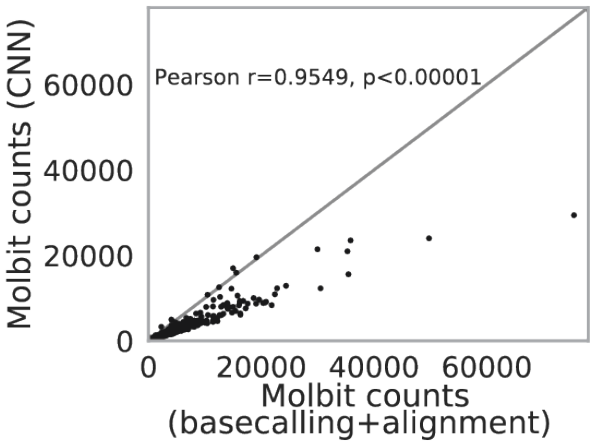

Figure A.4: **Comparison of molbit counts from sequence data vs. signal data.** Sequence data is labeled using basecalling plus alignment; signal data is labeled using a CNN.

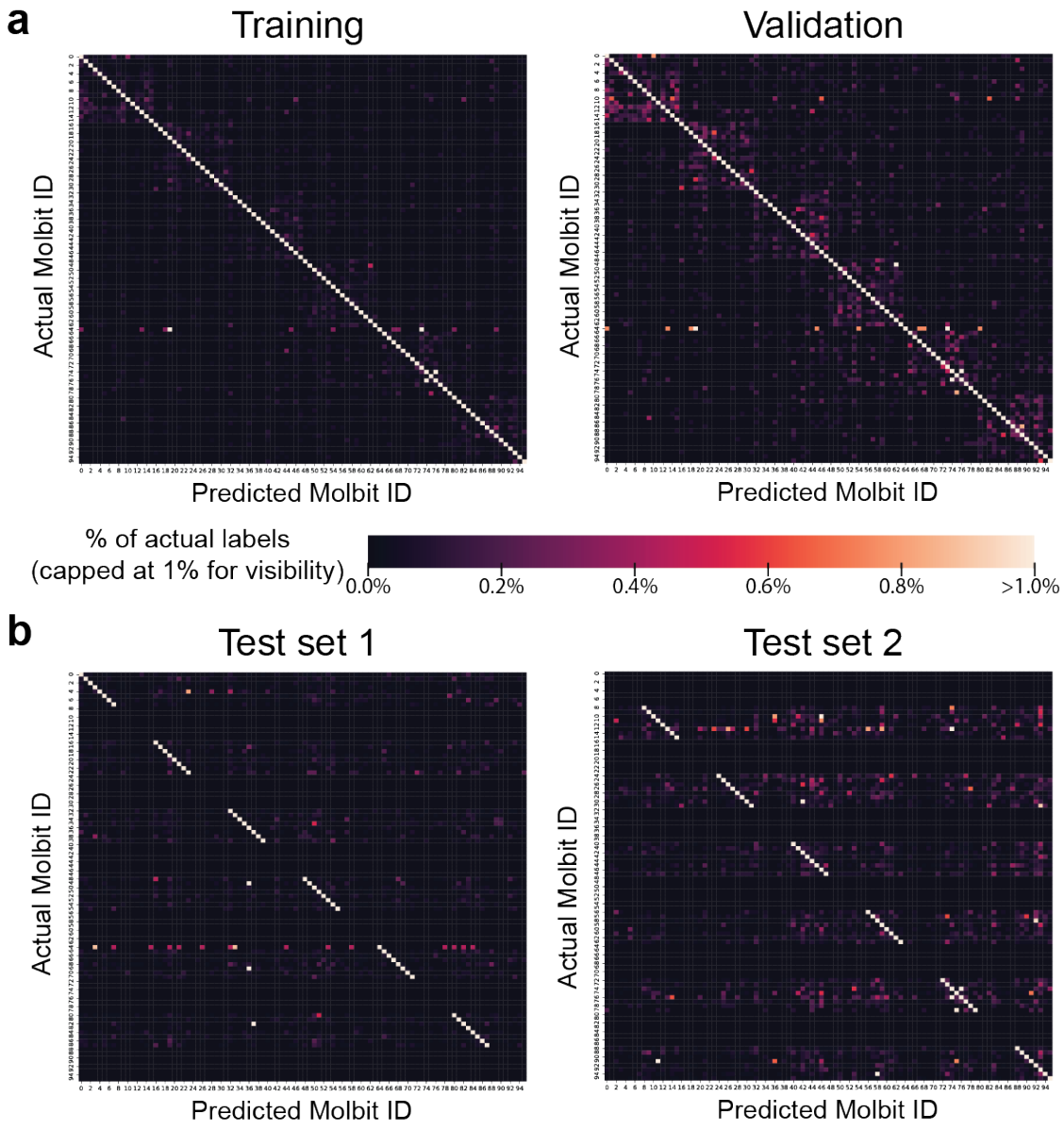

Figure A.5: **Confusion matrices for training, validating, and testing the molbit classification model.** (a) Training and validation. Since counts vary drastically for each molbit, values are normalized by the total number of actual (i.e., labeled via basecalling plus alignment) molbits. Due to high overall accuracy, the visualization is capped at 1% to make error patterns more visible. In the validation plot, some batching bias is visible, demonstrated by six squares surrounding the diagonal identity line. The 16 molbits within each of these six boxes were sequenced together in the same run. (b) Testing. As in (a), values are normalized by the total number of actual molbits. Each test set consists of a mutually exclusive set of half of the molbits, in this case arranged arbitrarily in groups of 8, causing the horizontal banding in the matrices.

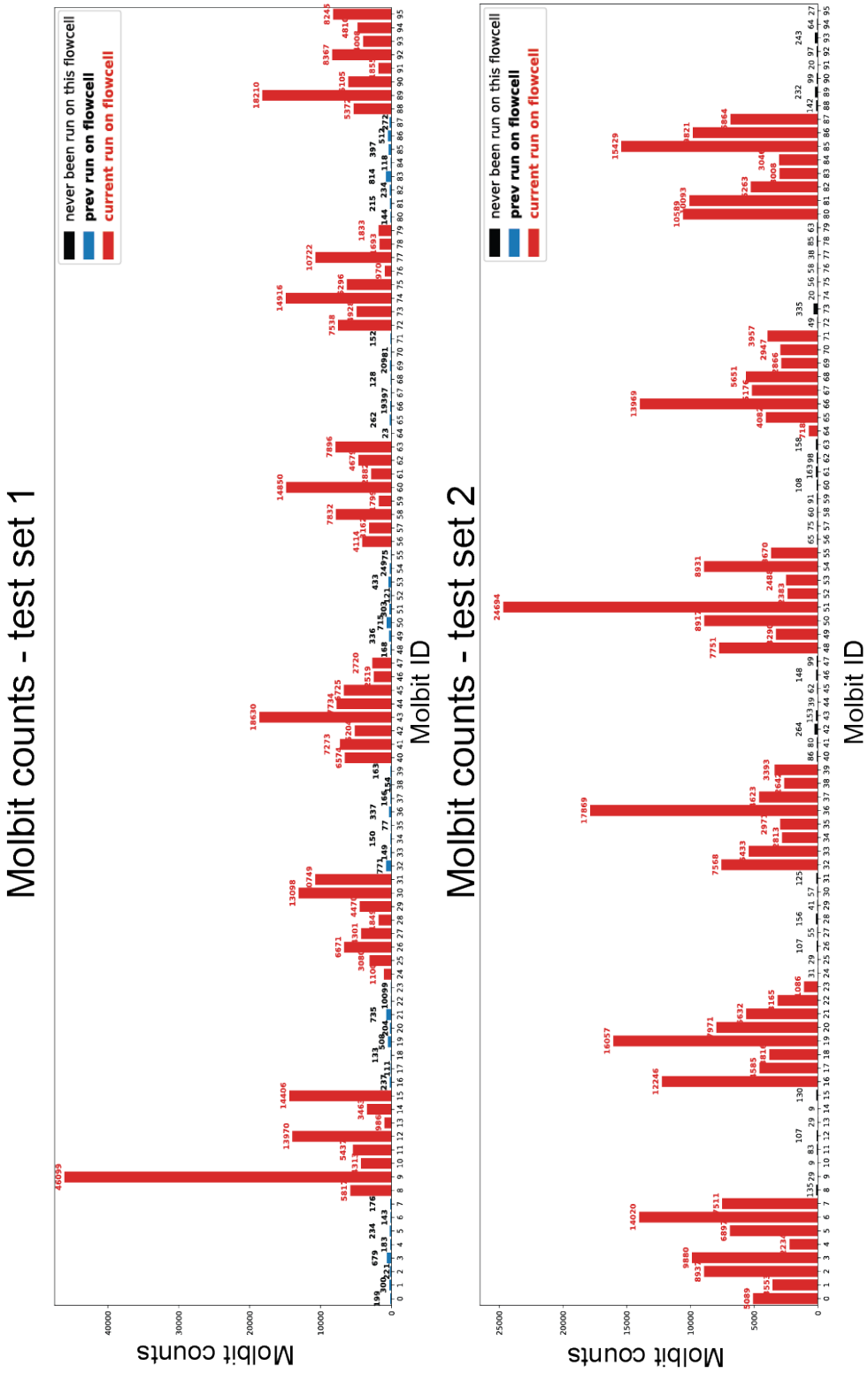

Figure A.6: **Read counts per molbit for test runs.**

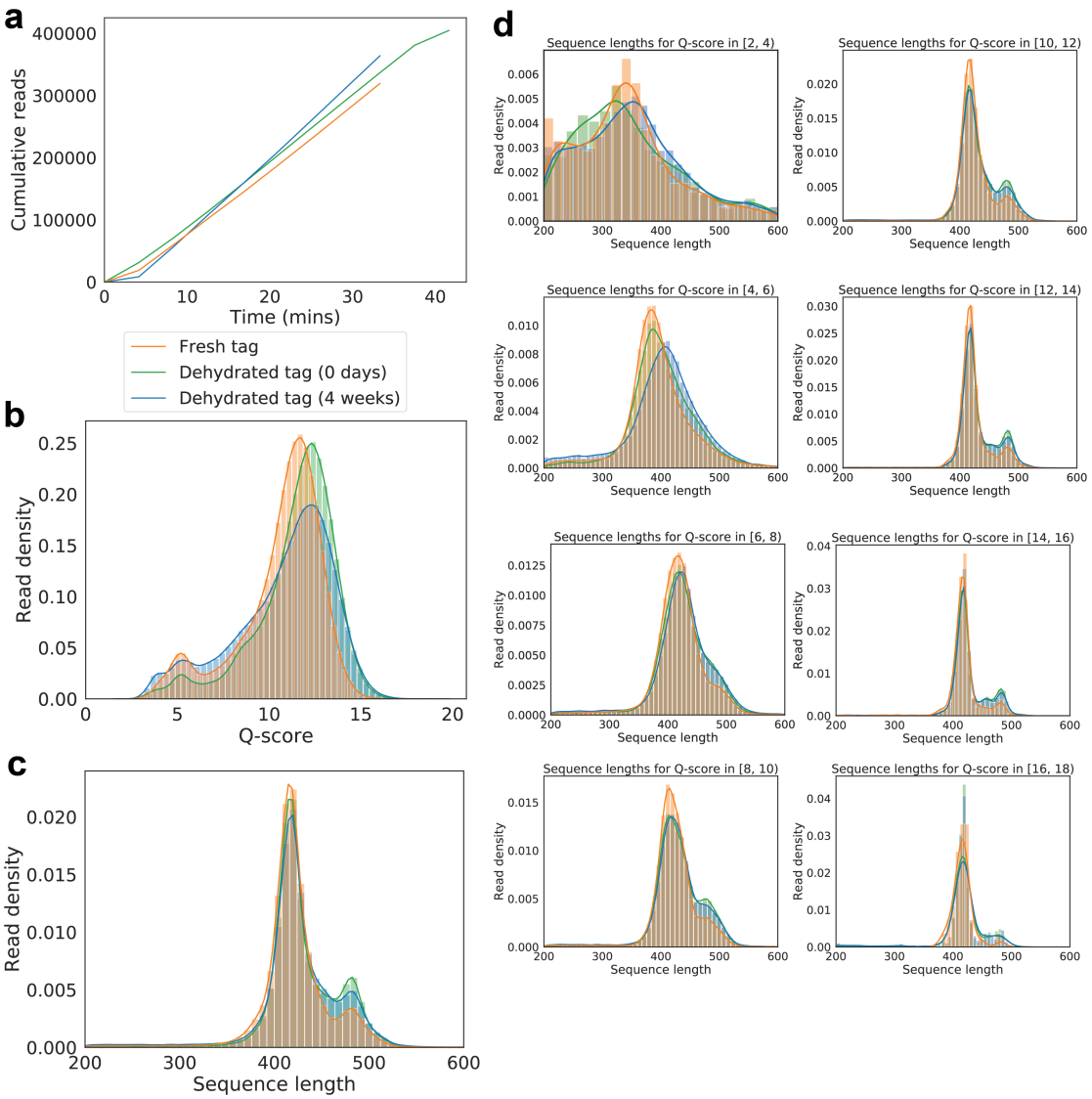

Figure A.7: **Quality of fresh versus dehydrated tags.** The fresh and dehydrated tag libraries were prepared at separate times, and the dehydrated tags were prepared as one library, then split after dehydration. (a) Cumulative sequencing read output over time. (b) Distribution of PHRED quality scores for all reads after basecalling using Guppy 3.2.2 (GPU version). (c) Distribution of sequence lengths after basecalling. (d) Same as (c) but split in bands of Q-scores.

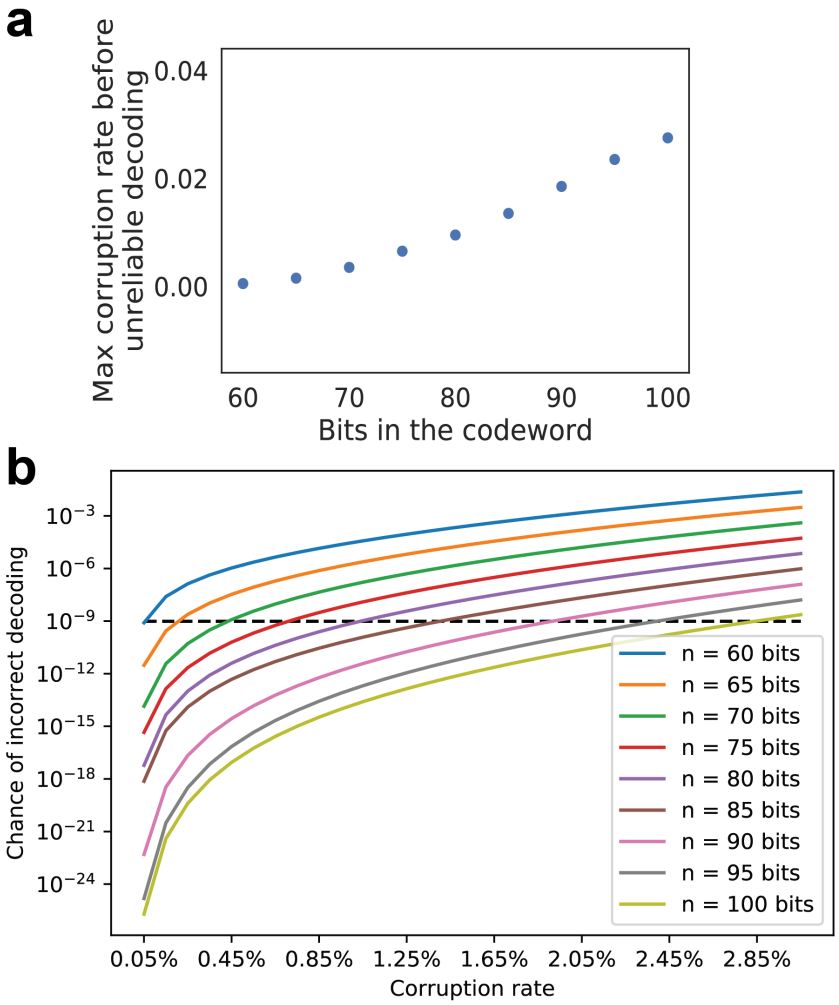

Figure A.8: **Impact of bits in the codeword on the required corruption rates.** (a) Bits in the codeword (n) vs. maximum corruption rate (k=32). Each point reflects the maximum corruption rate before the chance of incorrect decoding exceeds 10^{-9}. (b) Corruption rate vs the chance of incorrect decoding for different choices of n, (k=32). The dashed line is the desired incorrect decoding threshold; the goal is to stay below this line.

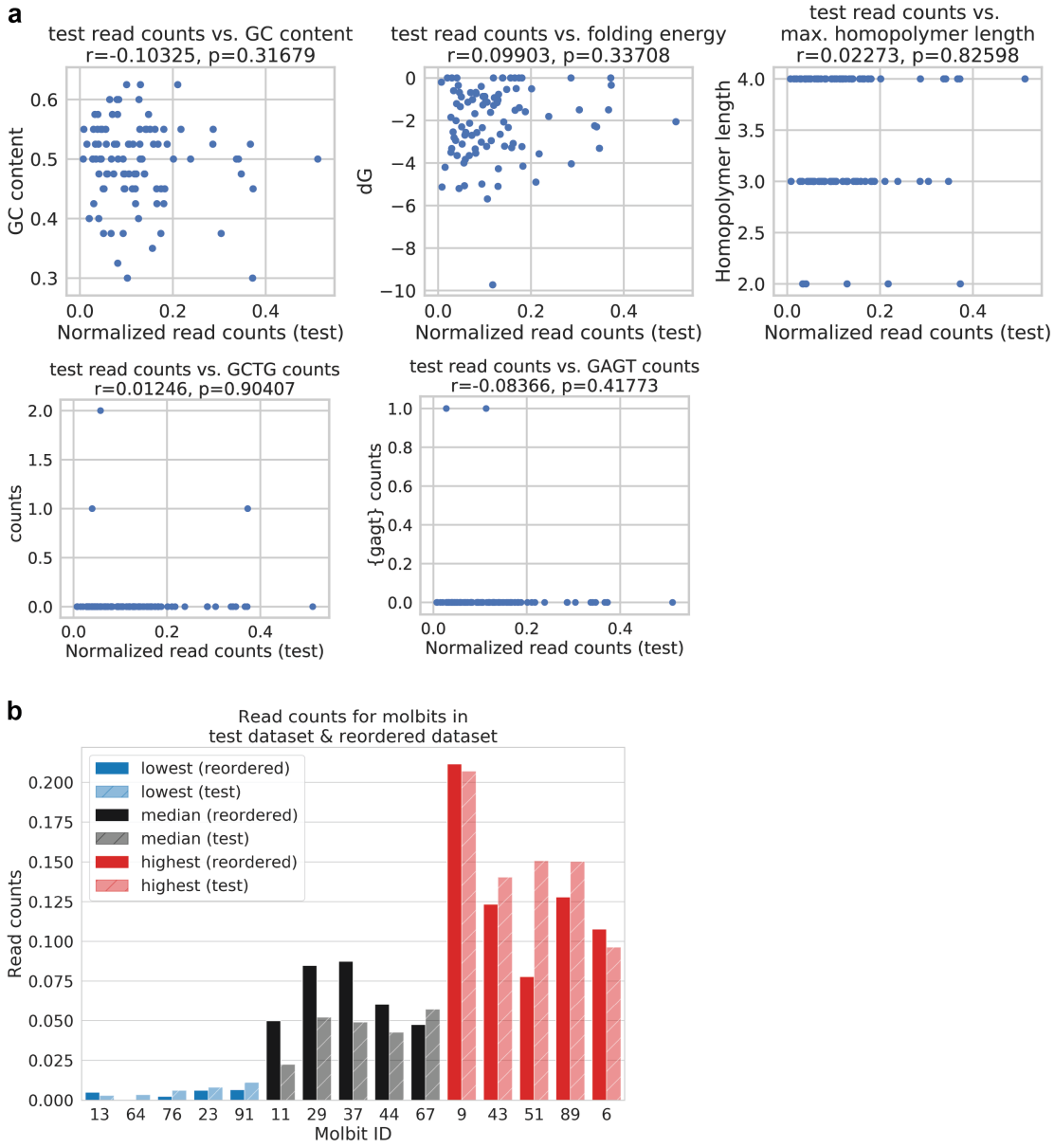

Figure A.9: **Read count variation analysis.** (a) Pearson correlation (two-sided) of normalized read counts against various sequence-related metrics: GC content, folding energy (δG minimum free energy), maximum homopolymer length, and mid-sequence presence of the assembly overhangs GCTG and GAGT). Normalized read counts were drawn from the two datasets used to test the model, and normalized training read counts showed similar trends. (b) Normalized read counts for molbit sequences that were reordered from IDT (solid bars), compared to the same molbits in the original test datasets (hashed bars). The reordered molbits were chosen randomly out of the lowest (blue), near-median (black), and highest (red) represented molbits.

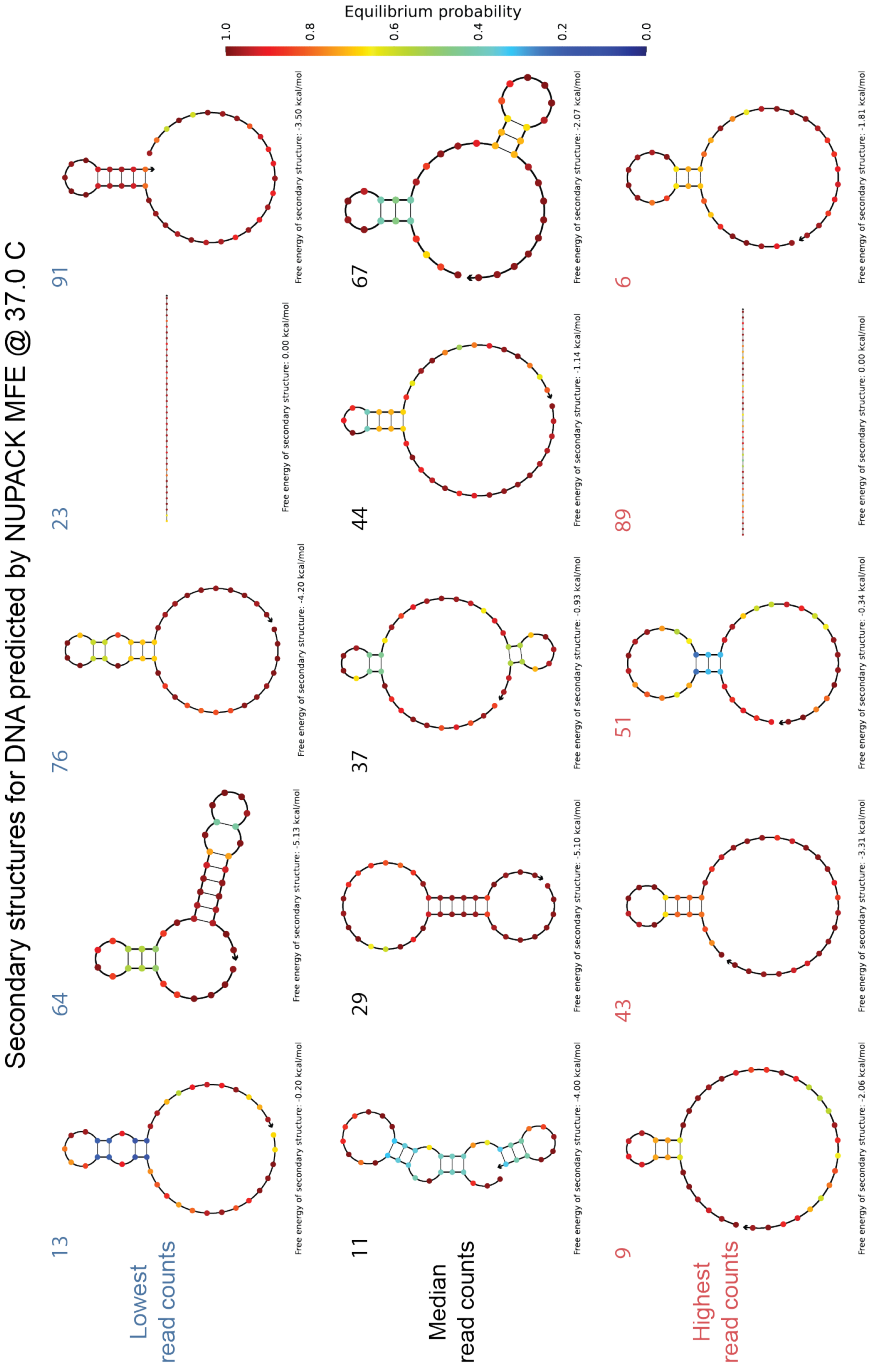

Figure A.10: **NUPACK predicted folding and minimum free energy for re-ordered sequences.** Top row contains low read count sequences, middle row contains average read count sequences, and bottom row contains high read count sequences.

Appendix B

SUPPLEMENTARY INFORMATION: BIG BITS

B.1 Supplementary Tables

Molbit name	Sequence
a0	TATCCAATCCTATTT
a1	GTGCGCCAACAGGAG
b0	CTTTGGAGAGTTTTA
b1	AGGGACTGTGTATCG

Table B.1: **Set 1 molbit sequences**

Molbit name	Sequence
a0	TTCCTATTTC
a1	GCGAAATTAG
b0	CTGGTGCATT
b1	GGTTCCTTAT

Table B.2: **Set 2 molbit sequences**

Molbit name	Sequence
a0	GTGAGCGGGAGACTA
a1	AAAGGAACTTCTCTG
b0	GATATGCGTCTATGT
b1	CTCCTGGACTTGAGA
c0	CTTGACTTTTCCGAC
c1	TCTTCCTCCCACGAG
d0	CGCTGGAGAAAGCTT
d1	GATTTCCCAAAGCTT

Table B.3: **Set 3 molbit sequences**

B.2 Supplementary Figures

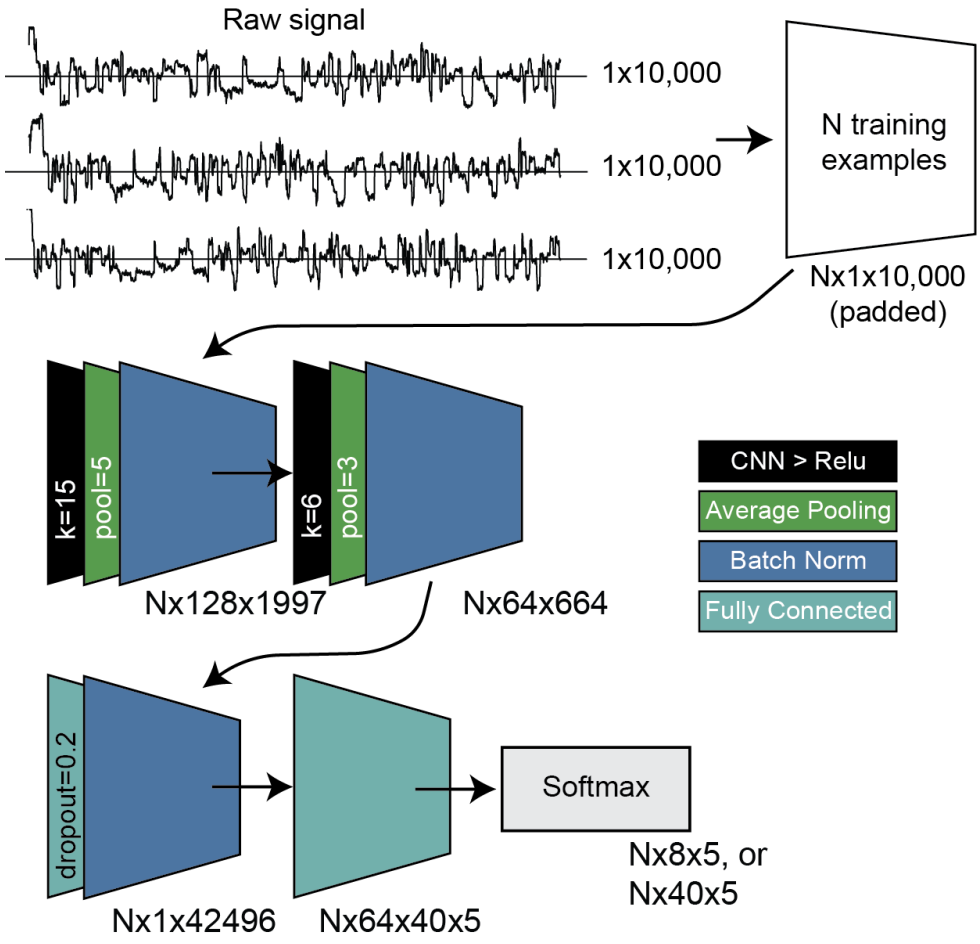

Figure B.1: **Big Bits molbit classification model and data flow.** Input data includes nanopore ionic current, which was truncated to the first 10,000 data points in the signal time series or padded with zeros if necessary. Training data then passes through a 2-layer CNN followed by one fully connected layer with dropout and a final fully connected layer with softmax as the output layer. In the CNN layers, k is the kernel size and pool is the average pooling kernel size.