NORTHWESTERN
UNIVERSITY

Computer Science Department

Technical Report
Number: NU-CS-2023-19

February, 2024

AI for Science: Graph Machine Learning as an Instrument for Understanding, Controlling, and Creating Physical Systems

Mattson Thieme

Abstract

We consider machine learning (ML) to be a programmatic analog of the scientific method. In this paradigm, a mapping encoded in the model weights (a theory) transforms input data (observations) into predictions, which are then compared against a target (validating the theory against reality). As training proceeds and the correspondence between *out-of-sample* predictions and reality grows, so too does our confidence that the learned mapping reflects some truth about the problem domain. If this agreement is sufficiently robust, an examination of the mapping may then yield insights into the rules and relationships governing the system under study. Inspired by the potential for ML to accelerate scientific discovery, we explore differentiable, graph-based algorithms that allow neural models to more efficiently extract and leverage latent *relational* information from raw data. Our focus on graph-based algorithms not only makes the work more general but also more readily applicable to the physical sciences, where graphs are the de facto representational structure. We introduce novel methods for 1) graph structure learning, where we fix the nodes and learn to retain/remove the edges of a graph; 2) graph partition learning, where we fix the edges and learn to cluster the nodes; and 3) molecular graph generation with a novel graph adaptation methodology, where we fix the edges and optimize graph-level properties by modifying node features. Finally, in the spirit of concretely accelerating scientific discovery with machine learning, we discuss our extensive work on AI for Science conducted in partnership with Fermilab, where we repurposed biomedical segmentation models for disentangling particle accelerator loss profiles, and recurrent sequence models for controlling high-frequency proton beam extractors.

Keywords

AI for Science, Graph Machine Learning, Graph Structure Learning, Graph Pooling, Drug Discovery, Particle Accelerators

NORTHWESTERN UNIVERSITY

AI for Science: Graph Machine Learning as an Instrument for Understanding, Controlling, and Creating Physical Systems

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Computer Science

By

Mattson Thieme

EVANSTON, ILLINOIS

March 2024

Abstract

We consider machine learning (ML) to be a programmatic analog of the scientific method. In this paradigm, a mapping encoded in the model weights (a theory) transforms input data (observations) into predictions, which are then compared against a target (validating the theory against reality). As training proceeds and the correspondence between *out-of-sample* predictions and reality grows, so too does our confidence that the learned mapping reflects some truth about the problem domain. If this agreement is sufficiently robust, an examination of the mapping may then yield insights into the rules and relationships governing the system under study. Inspired by the potential for ML to accelerate scientific discovery, we explore differentiable, graph-based algorithms that allow neural models to more efficiently extract and leverage latent *relational* information from raw data. Our focus on graph-based algorithms not only makes the work more general but also more readily applicable to the physical sciences, where graphs are the de facto representational structure. We introduce novel methods for 1) graph structure learning, where we fix the nodes and learn to retain/remove the edges of a graph; 2) graph partition learning, where we fix the edges and learn to cluster the nodes; and 3) molecular graph generation

with a novel graph adaptation methodology, where we fix the edges and optimize graph-level properties by modifying node features. Finally, in the spirit of concretely accelerating scientific discovery with machine learning, we discuss our extensive work on AI for Science conducted in partnership with Fermilab, where we repurposed biomedical segmentation models for disentangling particle accelerator loss profiles, and recurrent sequence models for controlling high-frequency proton beam extractors.

Acknowledgements

First and foremost, I would like to thank my advisor, Han Liu. His support, encouragement, guidance, and forthrightness helped me find my way in the churning waters that are AI research. Perhaps most importantly, he taught me that to do great research (and great things, more generally) we must work on *important* problems, not just clever ideas. And to do so in such a way as to grow a little bit every day. His confidence in me, unparalleled grasp of the fundamentals of machine learning, and breadth of knowledge made exploration feel safe, and eased the pressure that would otherwise make shortcuts tempting. I count myself extremely fortunate to have had him as an advisor and look forward to a long career of ongoing collaboration. I would also like to thank my committee, Doug Downey, Emma Alexander, and Abhishek Pandey, for their curiosity, engagement, and invaluable feedback throughout the process.

I am also fortunate to have met and worked with many exceptional people at Northwestern. I would like to thank Ken Forbus for believing in me, welcoming me to the university, and for his fascinating research probing the depths of analogy that inspires me to this day. I was also fortunate to have so many talented labmates, especially Zhihan Zhou, Weijian Li, Jing Jiang, Ammar Gilani, Jerry Hu, Jaiyi Wang, Guo Ye, and Qinjie

Lin, whose ideas and enthusiasm for research made the whole journey more fun. I'd also like to thank Andong Li Zhao for friendship, refreshing discussions, and books (at least one of which I still need to return).

This PhD cannot be separated from my experience collaborating with Fermilab. I would like to thank Kyle Hazelwood, Kiyomi Seiya, Vladimir Nagaslaev, and Aakaash Narayanan for innumerable discussions, helping me grok the physics, sharing your real selves, and sticking with it through all the challenges as we wove ML into the physical world. Our time together cemented my commitment to AI for Science and breathed purpose into the inherently uncertain process of refining ones research direction.

I was also fortunate to have the opportunity to work outside the university for two wonderful summers. First at the MIT-IBM Watson Lab in Cambridge, where Yada Zhu and Onkar Bhardwaj supported me in every way possible while teaching me the ins and outs of industrial research. And second, at AbbVie, where Majdi Hassan, Chetan Rupakheti and myself engaged in regular, friendly sparring matches over the latest and greatest ideas in graph machine learning. These conversations chiseled rough ideas into usable systems and were as fun as they were productive. I would also like to extend a special thanks to Abhishek Pandey for his flexibility, constant advocacy, and for making the collaboration with AbbVie a reality. This thesis, and this PhD as a whole, would not be what it is without his generosity and effort.

Even before starting this program, I was surrounded by supportive and brilliant people who nurtured my interest in research. I would first like to thank Oksana Ostroverkhova, who introduced me to research, brought me in as a full-fledged lab member when I'd barely declared a physics major, and gave me my first glimpse at how great research was done. As well as Nima Dinyari, who helped enormously in making the transition from industry to a PhD a reality, teaching me invaluable professional skills that I carry with me to this day.

I'd also like to thank Prashant Shah and Anthony Reina, who supported my every interest as I cut my teeth in industrial machine learning, providing me with an example for what a truly supportive work environment could be.

It isn't lost on me how fortunate I am to be surrounded by supporters, and it's something I've enjoyed from the get-go. For that, I would like to thank my parents, Steve and Sheila, for all their love and encouragement and genuine interest in what matters to me. I feel it in everything I do. As well as my siblings, Pierce and Elana, for holding me accountable, whether you knew it or not, and for living so confidently within your values. And my dear friends Benjamin and Joshua, for blazing the trail into academia, welcome breaks from the seriousness of the PhD and, most of all, for maintaining the bros for so long.

I'd also like to thank the newest (and most adorable) member of our family, Clara. Clara, your Mom and I love you so much we can hardly contain ourselves, and your giggles and hugs and all your many feelings, big and small, fill us with joy each day.

And finally, Kristin. I'll never forget that cool evening walk that launched this whole adventure, nor the hundreds of others that shaped it in the years that followed. Taking this leap was about so much more than what brought us here. It was about becoming the people we knew we could be, and that might be the great blessing of my life; that it's ours. Your love and belief in me transforms challenge into growth, and your playful confidence fills each new phase with meaning and delight. In addition to your unmatched capacity to see value in others that they haven't seen themselves, you have a magical way of engaging with ideas that both respects the present while inviting the next; drawing forth what's right and evaporating the rest. What's left is not just a single, crystalline concept, but a beautiful mosaic of interlocking ideas. These mosaics are reflected in every aspect of our lives, and make your side my favorite place to be.

Thank you for everything you do for our family. That we share this life will always be my greatest happiness.

So here's to our next adventure. May it be as full of growth, surprises and love as this was.

Contributions

Following a review of relevant background material in Part 1, Part 2 of this thesis flows in three parts as we move from graph structure learning to graph partitioning to molecular graph adaptation. Our contributions are as follows:

(1) Structure Learning: Fix the nodes, learn over the edges

We introduce a principled methodology for separating the distinct tasks of structure learning and node embedding in the widely used Graph Attention Network. This approach allows Graph Attention Networks to avoid the issue of an embedding mechanism obfuscating the structure learning signal and achieves an order of magnitude greater sparsification than comparable structure learning techniques while maintaining state-of-the-art performance.

→ Thieme, M., Zhu, Y., and Liu, H. The graph learning attention mechanism: Learnable sparsification without heuristics. *Under Review*, 2023.

(2) Partition Learning: Fix the edges, learn to partition the nodes

We introduce the first graph pooling algorithm that differentiably segments an entire graph into an arbitrary number of clusters of arbitrary size. Crucially, our approach makes no a priori assumptions about the number or size of the learned clusters while naturally partitioning the graph into *connected* substructures. The approach was inspired by pharmacology, where the properties of drug molecules (trivially represented as graphs) are functions of discrete, connected substructures within those molecules. We show that this method, while being the first graph pooling algorithm that makes no a priori assumptions about the number or size of the learned clusters, is capable of meeting or exceeding the performance of alternative graph pooling approaches on molecular and protein datasets.

→ Thieme, M., Hassan, M., Rupakheti, C., Thiagarajan, K. B., Pandey, A., and Liu, H. TopoPool: An adaptive graph pooling layer for extracting molecular and protein substructures. In *NeurIPS 2023 Workshop on New Frontiers of AI for Drug Discovery and Development* (2023).

(3) Molecular Graph Adaptation: Fix the edges, modify the nodes

We introduce the first-of-its-kind, end-to-end differentiable methodology for transforming molecular predictors into molecular graph manipulators. Specifically, we focus on a constrained variant of the bioisosteric replacement problem, where we learn to continuously vary atom types within a given molecular structure. By absorbing the vectorized atomic features of an input compound into a joint model containing these features and a trained property predictor, we can optimize for them directly and locate new compounds having the same structure but better properties. Challenges remain, but we demonstrate that the method is capable of differentiably optimizing compounds towards multiple properties while preserving the original structure, a first in the bioisosteric transformation literature. We are also proud to say that this method has yielded novel compounds that medicinal chemists deemed interesting enough to physically synthesize. This milestone inducts the method into a very small cohort of generative models that have produced a compound worthy of synthesis.

→ Thieme, M., Rupakheti, C., Cusack, K., Pandey, A., and Liu, H. Neural Atomic Replacement via Guided Molecular Adaptation. *In Preparation* (2024).

In Part 3, we discuss our extensive work with Fermilab on AI for Science, where we built semantic segmentation models to better understand accelerator behavior and developed novel sequence modeling methodologies to regulate the extraction of high-energy protons:

(4) Real-Time Loss De-Blending in the Main Injector and Recycler

Fermilab's Main Injector enclosure houses two accelerators: the Main Injector (MI) and the Recycler (RR). In periods of joint operation, when both machines contain high intensity beam, radiative beam losses from MI and RR overlap on the enclosure's beam loss monitoring (BLM) system, making it difficult to attribute those losses to a single machine. We introduce a novel, continuous adaptation of UNet [**89**] for disentangling the contributions from each machine to those measured losses. By extracting beam loss information at varying receptive fields, the method is capable of learning both local and global machine signatures and producing high-quality inferences using only raw BLM measurements.

→ K. Hazelwood*, R. Shi*, B.A. Schupbach*, **M. Thieme***, M.R. Austin, M.A. Ibrahim, V.P. Nagaslaev, D.J. Nicklaus, A.L. Saewert, K. Seiya, R.M. Thurman-Keup, N.V. Tran, A. Narayanan, H. Liu, S. Memik, "Real-time Edge AI For Distributed Systems (READS): Progress

*Equal Contribution

On Beam Loss De-blending for the Fermilab Main Injector and Recycler," in *Conference IPAC'21.*

→ K.J. Hazelwood*, **M. Thieme***, J. Arnold, M.R. Austin, M.A. Ibrahim, V.P. Nagaslaev, A. Narayanan, D.J. Nicklaus, G. Pradhan, A.L. Saewert, B.A. Schupbach, K. Seiya, R.M. Thurman-Keup, N.V. Tran, D. Ulusel, H. Liu, S. Memik, R. Shi, "Semantic Regression for Disentangling Beam Losses in the Fermilab Main Injector and Recycler" in *Conference NAPAC'22.*

→ Shi, R., Ogrenci, S., Arnold, J. M., Berlioz, J. R., Hanlet, P., Hazelwood, K. J., Ibrahim, M. A., Liu, H., Nagaslaev, V. P., Narayanan, A., Nicklaus, D. J., Mitrevski, J., Pradhan, G., Saewert, A. L., Schupbach, B. A., Seiya, K., **Thieme, M.**, Thurman-Keup, R. M., and Tran, N. V. Ml-Based Real-Time Control at the Edge: An Approach Using hls4ml. In the *31st Reconfigurable Architectures Workshop (RAW) 2024*.

(5) Slow Spill Regulation in the Muon-to-Electron (Mu2e) Conversion Experiment

A third-integer resonant slow extraction system is being developed for the Fermilab's Delivery Ring to deliver protons to the Mu2e experiment. Owing to the experiment's strict requirements in the quality of the spill, an advanced Spill Regulation System (SRS) is currently under design. We present two novel solutions to this problem: 1) a differentiable physics simulator for optimizing existing PID controller gains (the current system is designed to use a tuned PID to regulate the spill), and 2) a recurrent ML agent capable of replacing the PID controller entirely.

→ A. Narayanan*, J. Jiang*, **M. Thieme***, J. Arnold, M. Austin, J.R. Berlioz, P. Hanlet, K.J. Hazelwood, M.A. Ibrahim, V. P.

*Equal Contribution

Nagaslaev, D.J. Nicklaus, G. Pradhan, P.S. Prieto, B.A. Schupbach, K. Seiya A. Saewert, R.M. Thurman-Keup, N.V. Tran, D. Ulusel , H. Liu, S. Memik, R. Shi, "Machine Learning for Slow Spill Regulation in the Fermilab Delivery Ring for Mu2e", in *Conference NAPAC'22.*

→ A. Narayanan*, **M. Thieme***, K.J. Hazelwood, M.A. Ibrahim, H. Liu, S. Memik, V. P. Nagaslaev, D.J. Nicklaus, P.S. Prieto, K. Seiya, R. Shi, B.A. Schupbach, R.M. Thurman-Keup, N.V. Tran, "Optimizing Mu2e Spill Regulation System Algorithms", in *Conference IPAC'21, Campinas, Brazil, 2021.*

→ Xu, C., YC. Hu, J., Jiang, J., Memik, S., Shi, R., Shuping, A. M., **Thieme, M.**, and Liu, H. Beyond PID controllers: PPO with neuralized PID policy for proton beam intensity control in mu2e. In the *Machine Learning and the Physical Sciences Workshop, NeurIPS 2023.*

Through these contributions, we show how a principled understanding of widely used machine learning algorithms allows them to be adapted and repurposed to great effect.

*Equal Contribution

Table of Contents

List of Tables

List of Figures

Part 1

Preliminaries

1 | Setting the Stage

Breakthrough discoveries often follow the invention of new instruments (Figure 1.1) because new instruments not only allow us to observe *known* things in finer detail but to observe *new* things altogether. Microscopes revealed the tiny worlds of cells and microbes, spectrometers opened the door to an expanding universe, and electroscopes gave the first hints at the force that would one day power all of civilization. In this dissertation, we'll argue that machine learning is such an instrument, and that, through careful design and implementation, its utilization in science may bring about comparably consequential breakthroughs.

1.1. The Interplay of Scientific Discovery and Instrumentation

Already, machine learning has revolutionized how we think about designing predictive models in a wide array of fields, particularly those with large volumes of data or many moving parts. The most famous among them may be AlphaFold [**58**], the AI system developed by DeepMind that achieved groundbreaking accuracy in protein structure prediction. As of this writing, it has been used to enumerate 'the entire protein universe' predicting the shape of 200 million proteins, nearly every one known to science [**20**]. These shapes not only help us understand how the proteins function, interact, and affect

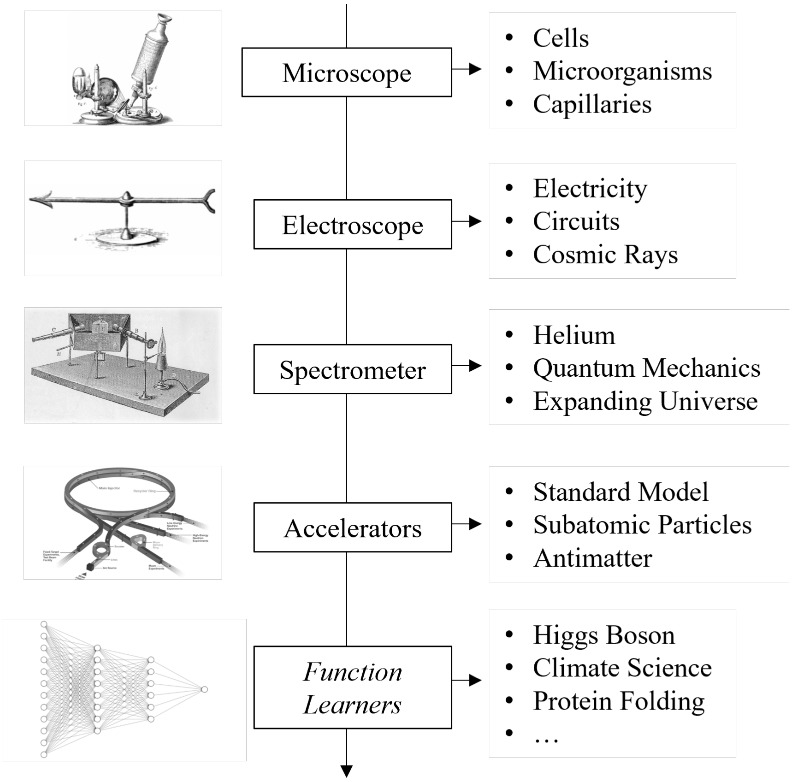

Figure 1.1. Just a few examples of the many discoveries that arose from the invention of a new scientific instrument.

molecular processes, but can help scientists develop therapeutic treatments and drugs that target these proteins more specifically and effectively.

In addition to leveraging the predicted protein structures yielded by AlphaFold, drug discovery has benefited from many other recent advances in machine learning as well. Finding safe and effective drug molecules requires sifting through millions of possible drug compounds, filtering out the clear failures, retaining the most promising candidates for further analysis and, eventually, physically synthesizing and testing thosse cadidate compounds. Machine learning has taken center stage in this screening process and promises to dramatically lower the cost and time to bring a new drug to market [**105, 35**].

Scaling up the number of interacting components, climate modeling, where billions of moving parts affect each other chaotically over long periods of time, is engaged in a similar transition towards learnable models. These models may supplant even deterministic physics simulators [**66**]. Not only do the learned models exceed the predictive performance of the brute-force incumbents, but they're able to do so with only a fraction of the compute. An important factor if you're making new predictions every day.

This is just a sampling of the *many* examples of ML's utility in science and engineering. In each case, ML systems are transforming raw data into scientific insights at a rate not possible without these learning models. And as we continue to use and rely upon these models to solve problems and predict the behavior of complex systems, we should think hard about what sorts of assumptions they should make about the forms that the solutions they're learning may take. In machine learning, these assumptions - baked into the architectures of the networks - are called *inductive biases*, and choosing the appropriate inductive bias is not only key to learning efficiency and performance, but to learning the *right* stuff.

1.2. Analogy

Making assumptions is central to all learning, and human learning is no exception [**11**]. In humans, our capacity for combinatorial generalization or, as Alexander von Humboldt put it, "making infinite use of finite means" [**110**] is enabled by our cognitive representation of the world. Most simply, we represent our world as a composition of entities and their interactions [**80, 86, 39, 59**]. This representational structure likely evolved due to the stunning utility of making *analogies*, in which we align similar structures (typically across two disparate domains) and use our knowledge about one to draw inferences about the other [**18, 48, 49, 36**]. This comes in particularly handy when we're in a new situation and need to quickly make a prediction about what happens next.

Science is filled with examples of this type of analogical thinking. Niels Bohr famously extended Rutherford's model of the atom by thinking of the nucleus and electrons like a sun and planets [**15**]. While not entirely correct, this understanding paved the way for more rigorous and exact descriptions that followed. Similarly, Charles Darwin drew an analogy between artificial selection (the breeding of animals and plants) and the processes that emerged naturally in wild ecosystems, leading to the formulation of his theory of evolution by natural selection [**99**].

Similarly, James Clerk Maxwell, who discovered the eponymous *Maxwell's Equations*, was aided by an analogy to fluid flow. Blending the works of Euler, then Faraday, who modeled magnetic fields after waves in a fluid and vibrations in a line of force, respectively, he proposed that lines of magnetic force may behave something like vortices in a fluid [**73, 17**]. This analogy, while again imperfect, catalyzed the discovery of the laws of electrodynamics and changed our world forever.

> *From a long view of the history of mankind—seen from, say, ten thousand years from now—there can be little doubt that the most significant event of the 19th century will be judged as Maxwell's discovery of the laws of electrodynamics.*
>
> - Richard Feynman, 1963 [**31**]

Following these examples, it's reasonable to ask why analogy works *so well* for improving generalization. And why does internal structure play such a dominant role in the behavior of complex systems? These questions approach the nature of reality from an interesting angle and touch on the fundamental mysteries of why the universe is comprehensible at all.

1.3. Symmetries and Scale Invariance

Without flying too close to the sun, one could venture a guess that the answer as to why analogy works so well may have something to do with how symmetries and scale invariance manifest in the physical world. In physics, symmetries play such a central role in the behavior of a system as to uniquely determine which physical quantities are conserved [**81**]. Temporal invariance yields the conservation of energy, spatial invariance yields the conservation of momentum, rotational invariance yields the conservation of angular momentum, and so on. This opens the door for the possibility that other, higher-order types of symmetries may manifest in ways that make analogously structured systems behave similarly even when their constituent parts are different, thereby making analogical reasoning more effective [**47**].

Additionally, many complex systems exhibit emergent properties that are scale-invariant, meaning that similar patterns and behaviors can emerge at different scales [**9, 71**]. This scale invariance may also contribute to the effectiveness of analogical reasoning because knowledge may be, in effect, *scaled up and down* as needed. Knowledge acquired in everyday life might be can applied *down* to understand interacting systems in microbiology, or *up* to understand entire ecosystems. This is simply because each of these examples are, in some general way, scaled versions of one another - complex systems of interacting agents - and as such might follow similar laws and act in similar ways [**112**].

Kenneth Craik, in his 1943 book “The Nature of Explanation” [**60**] - one of the first works to propose that thinking may be an act of manipulating an internal representation of the world - asked a similar question and came to a similar conclusion:

> *...[a human mental model] has a similar relation-structure to that of the process it imitates. By ‘relation-structure’, I do not mean some obscure non-physical entity which attends the model, but the fact that it is a working physical model which works in the same way as the process it parallels... physical reality is built up, apparently, from a few fundamental types of units whose properties determine many of the properties of the most complicated phenomena, and this seems to afford a sufficient explanation of the emergence of analogies between mechanisms and similarities of relation-structure among these combinations without the necessity of any theory of objective universals.*
>
> - Kenneth Craik, 1943*

It would then stand to reason that the *optimal* internal representation should mirror something fundamental about the structure of the external world, so that the outcomes of *mental* operations made on the internal representation would mirror the outcomes of *physical* operations made in the external world. To this author, it seems likely that our internal representation of the world was constructed in this way and that the structured, compositional nature of our internal representations is evidence that this type of structure is central to the nature of the external world.

If instead of describing a *human* learning model, we were describing a *machine* learning model, we would call this tendency to represent the world in a given way an *inductive bias*. And specifically, we call to the tendency to represent the world in terms of objects and their relations, a *relational inductive bias.*

*Excerpt taken from the exceptional *Relational inductive biases, deep learning, and graph networks* paper that inspired many of the ideas in this dissertation. [**11**]

1.4. Relational Inductive Bias

We can think of inductive bias as the assumptions we make about the form of the solutions we're after. Perhaps in humans, but certainly in machines, these assumptions are baked into the neural architectures. These assumptions can take many forms, for example, Convolutional Neural Networks (CNNs) assume the signal is translation invariant, while Recurrent Neural Networks (RNNs) assume it is temporally invariant. This looks suspiciously similar to the conservation laws introduced by Noether [**81**], but what matters is this: whatever the model architecture, it is making some assumptions about the structure of the signal or the function it is aiming to find.

Given the clear indication that structure and compositionality are fundamental to the laws governing the behavior of our universe, it seems fitting to do our best to imbue *machine* learning models with a similar set of assumptions that evolution has baked into *human* learning models (or, perhaps more accurately we could say *organic* learning models, so that we include all the other intelligent creatures that share our world).

This is where the concept of graphs comes into play. Graphs are data structures composed of three things: objects, their properties, and the relationships between them. The same organizing principle that governs analogy and human cognitive architecture more broadly. In this dissertation, we'll focus on *graph* machine learning, or methods for applying machine learning to these sorts of structured objects (graphs) in a way that respects the unique properties of those structures. We'll discuss these properties in depth in Chapter 2, but for now it is sufficient to know that there are ways of building learning systems that operate on a similar kind of data that *we* do - graph-structured data - and that using these techniques allows them to construct a similar internal representation of that data to one that we might.

1.5. First Principles

It's common to talk about the signal that is buried in noise, but it is often also buried under the assumptions we make about its form. As we've already mentioned, we can't do away with assumptions altogether, but through careful design, we can align them more closely with the principles we've just discussed in order to let our models learn more useful representations of their problem domain. This is, obviously, the first thing one considers when building a new model, but methodologies often inherit functions and techniques that come with unintentional or overlooked baggage. This is worthy of extra consideration, and in Chapter 4 we provide a concrete example of how redesigning some basic architectural components from first principles can materially improve structure-learning efficiency.

1.6. Purpose

This is all to say: we choose to study graphs in the context of machine learning because it encourages us to imbue our models with the same basic assumption that we make when trying to understand the behavior of complex systems: that the relevant actors are objects, their properties, and their relationships. With this understanding, graph awareness allows our models to leverage the same simplifying assumptions that we do, to focus on what matters most (internal structure) and to bring their learned representations into line with reality.

2 | Background

This section is broken up into three parts to provide the necessary background for each of the parts of the thesis.

2.1. Learning on Graphs

Broadly, graph-structured machine learning is a suite of approaches for imbuing models with relational inductive biases. We'll talk more about this soon, but inductive biases are just what allow learning algorithms to prioritize one representation over another, and relational inductive biases encourage models to represent their environment as sets of entities and relations (also called graphs). Practically, these types of structured representations reduce sample complexity and tend to improve generalization. Existentially, they recognize that internal structure is the essence of the thing. Be they atoms, proteins, families, or economies, it's the relationships between their constituent parts that make them interesting and unique.

Before we get too far, let's align on what exactly we mean by graphs. We typically express graphs with the following notation: $G(V, \mathcal{E})$, which denotes a graph G composed of a set of nodes V and a set of connections joining those nodes $\mathcal{E}$. These connections are called "Edges", hence the $\mathcal{E}$, and are what adds structure to the object. Edges not

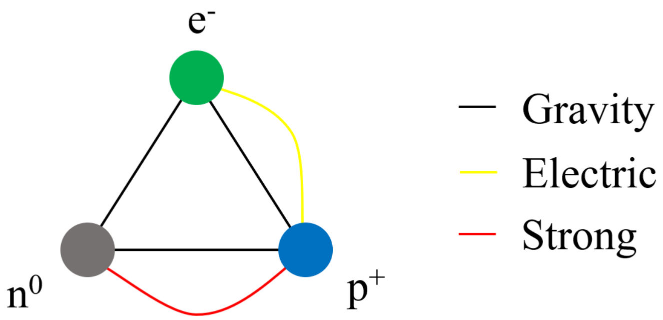

Figure 2.1. Edges adjoining two nodes can indicate both the presence and type of relationship. In this example, neutrons (n^0), protons (p^+), and electrons (e^-) all interact via gravity, but only protons and electrons interact via the electric force, and protons and neutrons via the strong force.

only reflect that a relationship between two nodes exists but can also reflect the *type* of relationship. In Figure 2.1, we show an example of how a graph can be used to describe a system of interacting parts, each with their own properties, interacting via multiple distinct interaction types.

Graphs pose an interesting learning challenge because they don't behave like most other types of data. First, it is important to note the language used above: G is composed of a *set* of nodes V and a *set* of connections $\mathcal{E}$. Because these are sets, the particular order in which they appear in some representation is irrelevant. This yields a unique requirement for any learning model operating on graphs: permutation invariance. This means that the output of any learning model operating on graphs must be invariant with respect to the particular order in which the nodes and edges are presented. In the simplest sense, graphs represents *locality*, meaning which nodes are relevant to which others. We'll discuss what *relevant* means more concretely soon.

2.1.1. Permutation Invariance

To formalize this notion, let's first discuss how graphs are typically represented. Individual nodes $v_i \in V$ typically have some features associated with them. These features vary by use case but generally represent node-specific information like type, etc. If we assume that we have d features per node, we express the node feature matrix X as follows: $X = \{v_0, v_1, \ldots, v_N\}, X \in \mathbb{R}^{N \times d}$. Edges can be represented a number of ways but the simplest is via an adjacency matrix. For a graph with N nodes, the adjacency matrix represents all possible N^2 pairwise interactions, taking the form of an $N \times N$ matrix we'll call A. Elements $A_{i,j}$ in A take on values of 0 or 1, where $A_{i,j} = 0$ means there is no edge between nodes i and j, and $A_{i,j} = 1$ means there is an edge between nodes i and j.

Now, if we have some function f acting on the graph G, this function needs to be invariant to the particular permutation of the nodes in X. Suppose we introduce a permutation matrix P which can be used to permute the rows of another matrix. P is a square binary matrix that has exactly one entry of 1 in each row and each column and 0's elsewhere. The function f is permutation invariant if and only if applying any permutation P to the nodes of the graph (and correspondingly to the feature matrix X) does not change the output of the function:

$$f(X) = f(PXP^{\intercal}) \tag{2.1}$$

As we'll discuss in the following section, neural networks operating on graphs satisfy this requirement by using permutation invariant aggregation functions like addition, multiplication, etc.

However, another challenge remains that functions operating on graphs need to account for: graphs can have widely variable numbers of nodes with widely variable connectivity structures. How we aggregate and normalize this information will have dramatic effects on the quality of the models we can learn.

Even having only covered these basics, it is clear that learnable functions operating on graphs need to satisfy a variety of properties unique to graphs. Just like image processing birthed the convolutional network, which satisfied the translation invariance requirement of image data, a new kind of neural network was birthed to satisfy the permutation invariance of graph data: the graph neural network.

2.1.2. Graph Neural Networks

The development of GNNs has been driven by a continuous interplay between theoretical advancements and practical applications. The field remains vibrant and rapidly evolving, with ongoing research addressing both fundamental challenges and innovative uses of GNNs in complex, real-world scenarios. This thesis introduces novel algorithms for, and applications of, GNNs, and as such we will cover a bit of both in this introduction.

2.1.3. A Brief History

Following around a decade of work applying recursive neural nets to graphs and tree-structured data, the term Graph Neural Network was introduced in 2005 [**40**]. This work laid the foundational framework for neural networks operating on graphs and introduced a differentiable method for directly processing graphs and producing node-level outputs. Following this introduction, the exploration and development of key concepts in GNNs exploded, such as neighborhood aggregation and the use of recurrent neural network (RNN) structures to process graph data.

In the years that followed, spectral methods took center stage, with one of the seminal works being Bruna et al.'s 2013 paper which proposed a form of graph convolution using the eigen-decomposition of the graph Laplacian [**19**]. However, while effective, spectral methods were (and still are) computationally expensive and fundamentally unscalable. In the face of usable datasets of exponentially increasing size, this felt untenable, and more efficient solutions were devised.

Most notably, the Graph Convolutional Network (GCN) [**65**] introduced by Kipf and Welling in 2016 brought scalability and parallelizability to the space. In response to its simplicity and effectiveness, the GCN became a cornerstone of the GNN community nearly overnight. The later 2010s saw the rapid expansion of GNN architectures and applications, with variants like GraphSAGE [**45**], the Graph Attention Network (GAT) [**109**], and others introducing new mechanisms for aggregating neighborhood information and attention mechanisms for scaling the influence of each neighbor.

As the popularity of GNNs increased, the field began to diversify with specialized networks for different types of graphs (e.g., directed, heterophilic) and various tasks (e.g., node classification, graph classification, link prediction). At the same time, scalability to large graphs became a major focus, and techniques for mini-batch training, subgraph sampling, and distributed computing were developed. PyTorch was beginning to rise to prominence, and the now ubiquitous PyTorch Geometric (PyG) was born [**29**].

These days, GNNs are being applied to a wide range of real-world applications including social network analysis, recommendation systems, drug discovery, and more. A thriving field with equal opportunity for theoretical, industrial, and social impact.

2.1.4. Neural Message Passing

Ok, but what actually *is* a GNN? First, it isn't so much a new *kind* of neural network as it is a new *way* of applying neural networks. We still use gradient descent to update the weights, dense layers, sparse layers, ReLU activation functions, and batch or layer normalization; all the same tricks we're accustomed to using in deep learning. We just do so in a way that accommodates the demands of working with graph-structured data, specifically: satisfying permutation invariance. However, unlike in convolutional or recurrent neural networks (CNNs or RNNS), the computation being done fundamentally changes in response to the input.

In a GNN, the input itself defines the computation graph.

What this means is that we conform the computational architecture to the input in order to learn how a node relates to its neighbors, and we do so using a methodology called *neural message passing*. This neural message passing, where vector messages are exchanged between nodes in the graph and updated using neural networks, is the defining feature of a GNN [**44**]. These vector messages start out as the input features of each node $v_i \in \mathbb{R}^{1\times d}$ and become, through iterative message passing, node *representations* $h_i \in \mathbb{R}^{1\times d'}$, where d' is the dimension of the node representation and may differ from d.

At the highest level, neural message passing follows a simple pattern:

$$h_i^{(k+1)} = UPDATE^{(k)}\Big(h_i^{(k)}, AGG^k(h_j^{(k)}, \forall j \in \mathcal{N}_i)\Big) \tag{2.2}$$

where $h_i^{(k+1)}$ is the representation of node k after $k+1$ message passing steps, $\mathcal{N}_i$ is the one-hop neighborhood of node i, and $UPDATE$ and AGG are differentiable Update and Aggregation functions, respectively (typically neural networks). We use the superscripts k and $k+1$ to indicate the particular round of message passing, which is usually called a *layer* in a GNN, i.e., a k-layer GNN would include k rounds of message passing.

The "messages" in message passing are the embeddings on the neighboring nodes, and it is the task of the AGG function to aggregate them into a single message that will then be combined with the target node embedding h_i^k and transformed with the $UPDATE$ function to form a new embedding for the target node $h_i^{(k+1)}$. What this means is that, after k message passing steps, each node will contain information about its k-hop neighborhood. The optimal value of k will vary by application, but for now it is just important to recognize that message passing is a way of contextualizing each of the nodes, and the number of message passing steps defines the size of the neighborhood over which we'd like to contextualize.

To make the description in Eq. 2.2 more concrete, we can express it in a simplified form of the original GNN [**40**], borrowed from [**44**]:

$$h_i^{(k+1)} = \sigma\left(\mathbf{W}_{\text{self}}^{(k)} + \mathbf{W}_{\text{neighbor}}^{(k)} \sum_{j \in \mathcal{N}_i} h_j^{(k)} + \mathbf{b}^{(k)}\right) \tag{2.3}$$

where $\mathbf{W}_{\text{self}}^{(k)}$ and $\mathbf{W}_{\text{neighbor}}^{(k)}$ are learnable weight matrices, σ is a simple nonlinearity, like a sigmoid, and $\mathbf{b}^{(k)}$ is a bias term. This is a very straightforward operation: 1) transform the target node features into a representation, 2) sum the neighboring node features and transform that summed vector into a representation, and 3) sum the two together and pass that summed vector through a nonlinearity. The resultant, updated node representation for node i contains information about it's k-hop neighborhood and was obtained differentiably. So, if we wanted to perform, say, node-level classification, we would simply feed each of the transformed node representations a multi-layer perceptron or even a single linear layer to generate node-level predictions. This is how the learned node representations are used to make node-level predictions.

2.1.5. Neighborhood Normalization

Given the concrete example of a GNN in Eq. 2.3, a new problem becomes clearer: normalizing the final message generated by the neighborhood aggregation function. If $|\mathcal{N}_i| = 10$ and $|\mathcal{N}_j| = 1,000$, the summation of those neighboring messages is going to have an entirely different distribution. As neural networks struggle to gracefully handle distributional shift, we need to normalize the incoming messages. The simplest way to do this is to take an average instead of a sum:

$$h_i^{(k+1)} = \sigma\left(\mathbf{W}_{\text{self}}^{(k)} h_i^{(k)} + \mathbf{W}_{\text{neighbor}}^{(k)} \frac{\sum_{j \in \mathcal{N}_i} h_j^{(k)}}{|\mathcal{N}_i|} + \mathbf{b}^{(k)}\right) \tag{2.4}$$

There are many other approaches, including the Kipf normalization introduced in the original GCN paper [**65**], which divides the messages by $\sqrt{|\mathcal{N}_i||\mathcal{N}_i|}$ in order to minimize the effect of neighbors having high degrees, but it is beyond the scope of this thesis to enumerate them all.

2.1.6. Node vs. Graph Level Prediction

Now that we've seen the formulations in Eq. 2.2 and 2.3, the extension from node-level predictions to graph-level is clearer: we simply need to aggregate the information contained in *all* the node representations and map that bulk representation down onto some prediction. For regression, where we want to generate a scalar-valued prediction of a graph, often used in such tasks as molecular property prediction (more in Chapters 5 and 6), the formulation might look something like this:

$$y = \mathbf{W}_{graph}\left(\sum_{i=0}^{N} h_i\right) \tag{2.5}$$

Where $\mathbf{W}_{graph}$ is a learnable weight matrix mapping the summed representation onto a single scalar value.

While this accomplishes the task of graph-level prediction, it does so at the cost of expressivity. Aggregating every node in such an indiscriminate manner discards much of the natural hierarchical structure present in graphs. If we want to capture and exploit this hierarchical structure to make better predictions, we'll need a way to locate and aggregate the nodes into higher-level clusters. Doing this in a differentiable manner turns out to be a difficult challenge, and one we take material steps toward solving in Chapter 5. To prepare for the approach we introduce in Chapter 5, we'll review the task of progressively and intelligently coarsening a graph, also known as graph pooling.

2.1.7. Graph Pooling

In Euclidean domains like computer vision, convolution and pooling (or coarsening) operators work in unison to extract, transform and compress local information into rich, abstract features. Extending this pairing to non-Euclidean domains such as graphs, however, has proven to be a persistent challenge. While convolutional operators have been elegantly adapted to graph settings, the GCN being a prime example, the graph pooling literature has not seen similar progress. This is largely due to the lack of inherent spatial regularity and the unique challenges of coarsening a graph while preserving its topology.

Fundamentally, pooling operators facilitate dimensionality reduction and feature abstraction and enable neural networks to learn hierarchical representations of data. This is a desirable function when considering graph data with all its implicit, hierarchical information.

Figure 2.2 gives a colorful display of why pooling a graph is challenging. In addition to the variable sizes and shapes of the substructures in a graph, nodes have widely variable degrees, and the graphs themselves can have a widely variable number of nodes. Exacerbating an already challenging problem is how to traverse a non-Euclidean object like a graph. In CNNs, pooling kernels are stepped over the image in a linear fashion, much like

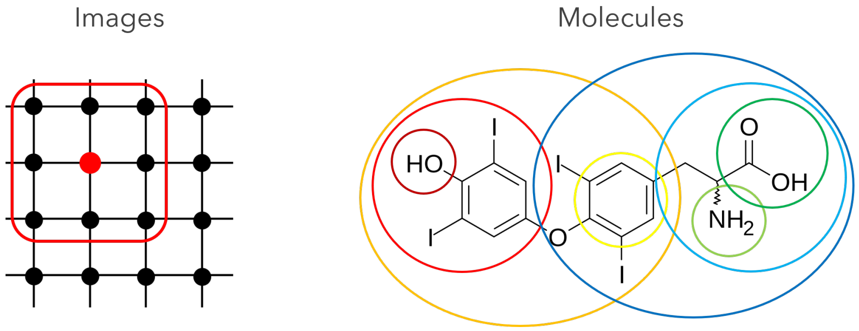

Figure 2.2. A colorful example to demonstrate the relative difficulty of defining a pooling operator on molecular graphs vs. images. On the left, we see the highly regular structure of an image. Pooling the one-hop neighborhood will yield predictable results for every neighborhood. On the right, we see the highly irregular structure of a molecule. The optimal pooling kernel is not at all clear, nor is how we would implement an analog of the "sliding" kernel action in images. Non-Euclidean data is more difficult to coarsen than Euclidean data.

mowing a lawn. But with cycles, bottlenecks, and dead ends at any position in the graph, applying the same protocol would not yield the desired result if applied to objects like a molecular graph. Finally, in the face of all this variability, the non-negotiable feature of any graph pooling algorithm is that it must maintain differentiability. Without this, we cannot use it as a drop-in layer in a GNN.

To date, a number of methods have been proposed, each tackling a particular facet of the problem [**33, 42, 116, 68, 25**]. We will discuss each in greater depth in Chapter 5. Here, we'll outline the broad contours of the problem.

Recall our definition of a graph $G(V, \mathcal{E})$ having N nodes $V = \{v_0, v_1, \ldots, v_N\}$ and edges $\mathcal{E}$. Graph pooling, or coarsening, is the process of mapping $|V| = N$ nodes and onto a new set of nodes $|V'| = M$, where $M \leq N$. To accomplish this, graph pooling operators possess some combination of the following three functions: selection, reduction, and connection (SRC) [**42**]. Selection is the process of grouping the input nodes, reduction is the process of aggregating and compressing those groups into new representations, and connection is the process of reconnecting the clustered nodes given the original edge set $\mathcal{E}$.

Given the discrete and connected nature of graphs, we would like the pooled or coarsened representation to have the following properties:

(1) **Complete:** We want to pool every node in the graph, not just select, for example, the Top-k nodes.

(2) **Dynamic:** Given the variable topologies of graphs, some graphs may warrant more pools than others, but it is not clear a priori how many a given graph will need. An ideal graph pooling algorithm would be able to adapt to the input graph and automatically learn the optimal number of pools.

(3) **Connected:** graphs represent locality and connectedness and we want the pooled representation to preserve this information. Pools should be composed of *only* connected nodes.

(4) **Discrete:** graphs represent discrete objects and relations, and we want our clusters to represent the same. Each node in V should be assigned to one and only one cluster.

As previously mentioned, a wide variety of learnable graph coarsening methods have been introduced to address these problems [**33, 42, 116, 68, 25**], and each comes with its own set of trade-offs. Some maintain connectivity but are not dynamic; others yield discrete clusters but need exogenous regularizers to enforce connectivity. Non-learnable, heuristic pooling methods often fail to capture the complex structural information in the graph, while learnable methods are often expensive and sensitive to the specific choice of hyperparameters. Learnable methods must also contend with maintaining differentiability: the central obstacle that produces the trade-offs listed above. To date, differentiable methods have had to make strong a priori assumptions about either the number or size of the learned pools, a severe disadvantage in domains where the optimal pool size is either unknown or variable across examples (most domains).

The approach we introduce in Chapter 5, called Topographic Pooling (TopoPool) is the first differentiable graph pooling operator to satisfy *all* these desiderata at once while maintaining differentiablity. We will save our discussion for Chapter 5, but we believe TopoPool is a material contribution to the graph pooling space and a useful addition to the GNN toolkit.

Next, we'll talk about the case in which the input graph, specifically the connectivity defined in $\mathcal{E}$, might not be trusted.

2.1.8. Graph Structure Learning

So far, we've assumed that the given graph $G(V, \mathcal{E})$ is the ground truth, that every edge is relevant, and that we should make use of them. But this is hardly the case in practice. Graphs are typically constructed using simple heuristics - k-nearest neighbors, distance thresholds, etc. - and often are filled with irrelevant edges that actually harm performance, which brings us back to the purpose of the word "relevant". A *relevant* edge is an edge that can be used to improve the prediction performance. An irrelevant edge is the opposite, one whose inclusion harms prediction performance. As with any model of any kind, be it a learned neural network or a scientific theory, we can trust that it captures some reality only to the extent that it can transmute the input features into reliable predictions.

This is because standard graph neural networks make the assumption that we've been making, and by default use every given edge to make their predictions. Graph structure learning adds a layer to the graph learning problem: not only do we want to learn how the nodes relate and to use that knowledge to improve prediction performance, but we also want to know *which types of neighbors are actually helpful in making predictions.* This brings with it a whole host of additional issues that we will cover at a high level here and deeper in Chapter 4.

So, the first question must be: why would the presence of an edge ever harm prediction performance? The answer lies in the assumptions that our GNN models make about the properties of the input data.

2.1.9. Homophily and Heterophily

Formally, the edge homophily ratio of a graph is a value the range [0, 1] that denotes the fraction of edges in the graph that join nodes with the same labels:

$$H(G, \{y_i; i \in \mathcal{V}\}) = \frac{1}{|\mathcal{E}|} \sum_{(i,j)\in\mathcal{E}} \mathbb{1}(y_i = y_j) \tag{2.6}$$

Where G is an input graph with nodes $v \in V$ having labels y. While homophily isn't an unrealistic assumption, with most graphs being constructed this way [**75**], a method for graph adaptation based on the assumptions of the downstream GNN is still desirable as many GNN models implicitly assume a high degree of homophily and perform poorly when this assumption is violated [**119**]. For example, in graphs with low homophily, such as the WebKB[*] datasets, GNN performance is often optimized by removing nearly every edge.

At this point, we may ask why we don't just use a model like the Graph Attention Network (GAT) [**109**], let that network learn to zero out the edge weights from neighbors that harm performance, and improve performance that way? It turns out, this isn't what happens.

Consider the following: take the Cora dataset [**74**] - an academic citation dataset in which nodes are papers and edges connect cited papers. If we modify the connectivity structure, $\mathcal{E}$, to add, for every node, three random edges to other nodes (papers that are *not* cited by the current paper), then train a GAT to perform semi-supervised node classification on that new graph, we get a classification accuracy of 65.9%. This is not very good. If, then, we *manually* set the edge weights (represented as attention coefficients in the GAT) for each of those newly added random edges to zero (note: the GAT is not able to learn this weighting on its own) the same GAT is able to achieve 79% accuracy. Better, but still not great. However, if we simply *remove* all the newly added random edges so the GAT does not consider them at all, the GAT can achieve 82.1% accuracy. This is in line with the expected best performance of the GAT on this dataset. The reason we see this behavior has to do with how the softmax function normalizes the incoming edge weights about some target nodes. We will go into detail about in Chapter 4, but what this tells us is that if we want to make use of the softmax function to normalize the weights on the incoming edges (we do, as this is exceptionally useful for node embedding) then it is more

[*] http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-11/www/wwkb/

effective to *drop* the noisy edges altogether, rather than allowing the network to learn to zero those edge weights. This is the task we call *structure learning.*

Structure learning as a task is distinct from node embedding, and to do it well requires disentangling the two. Our method, the Graph Learning Attention Mechanism (GLAM) introduced in Chapter 4, is the first to not employ any edge selection heuristics, exogenous structural regularizers, or otherwise modify the existing loss function to accommodate the structure learning task. Not only this, but GLAM can match state-of-the-art semi-supervised node classification accuracies while inducing an order of magnitude greater sparsity than existing graph learning methods.

Now, something we've glossed over until now is how we actually go about learning to retain or discard noisy, irrelevant edges in a graph. Edges are discrete, there either is or isn't an edge present, and, as such, aren't easy to add or remove differentiably. Solving this problem requires a clever way of rethinking the sampling process that pushes the non-differentiable operation out of the way so that gradients can flow through it. This is known as the reparameterization trick, and we refer the curious reader to [**55**] to learn more about this extraordinarily elegant technique.

——— * * * ———

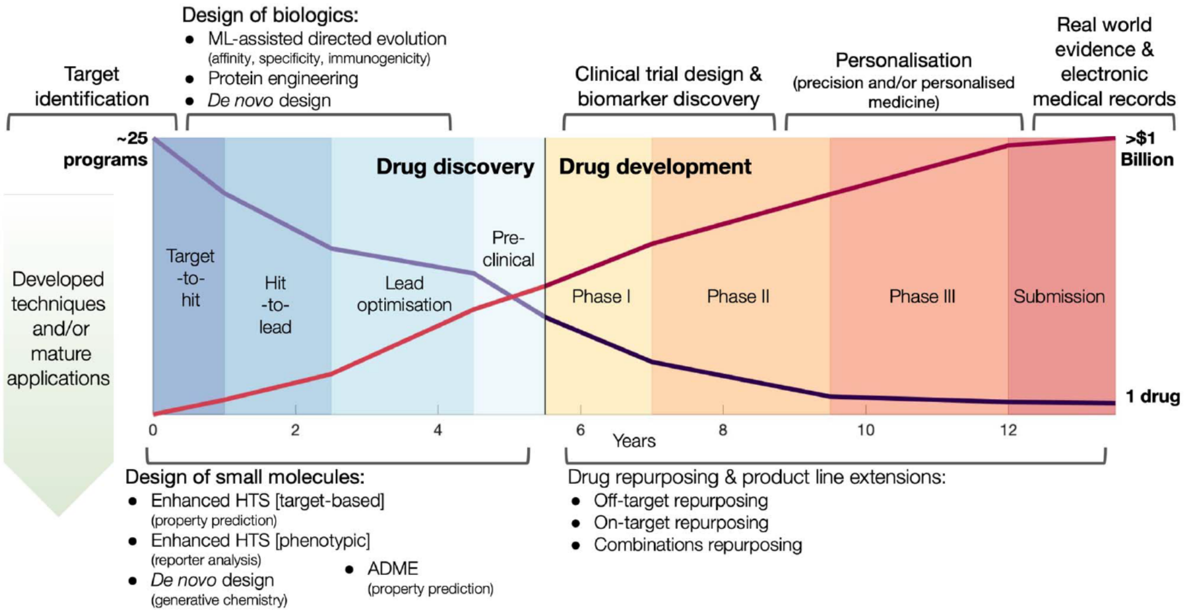

Figure 2.3. Figure borrowed from [**35**], with the financial, timeline, and success probability data taken from [**85**]. This figure is to show how astronomically expensive and time-consuming it is to bring a drug to market. Mean costs are >\$1.3B over 12 years. In our work, we target applications in the Discovery phase on the left, introducing new methods for Lead Optimization and Screening.

2.2. Machine Learning in Drug Discovery

Two of the works presented in this dissertation (Chapters 5 and 6) were designed for and inspired by applications in chemistry and drug discovery. In this section, we'll review how chemical compounds are represented and what sort of tasks we use machine learning for in this space, as well as provide a brief outline of the field to scope the magnitude of the challenges.

To begin, we refer the reader to Figure 2.3, borrowed from [**35**], which gives a sense of the scope of the problem. It takes ∼12 years and >\$1.3B to bring a single drug to market. Reducing this time and cost will not only save on expenses but more quickly bring life-saving medicines to people who need them.

The first thing to know about this process is what a drug is: a drug is a molecule, which is just a group of atoms bonded together to form an electrically neutral, physically stable structure. Molecules are the most fundamental unit of a chemical compound and the elements that take part in chemical reactions. In our body, these molecules attach to proteins and affect how physiological and biochemical processes play out. These processes are typically called pathways.

2.2.1. Molecules

Small-molecule drugs like aspirin, insulin, and antihistamines, make up 90% of the pharmaceutical drugs on the market [**98**]. On average, these compounds are made up of between 20-100 atoms.

In chemistry, molecules are naturally represented as graphs, Figure 2.4, where atoms are nodes and chemical bonds are edges. Each node (atom) is characterized by features such as:

(1) Atomic Number: Identifies the element of the atom.
(2) Chirality: Describes the three-dimensional arrangement of the atom, particularly in stereochemistry.
(3) Degree: The number of explicit bonds the atom has with other atoms.
(4) Formal Charge: The electric charge of the atom.
(5) Hybridization: The type of orbital hybridization (e.g., sp, sp2, sp3).

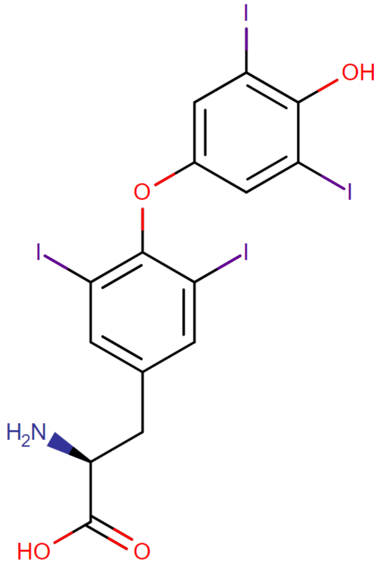

Figure 2.4. The common drug Levothyroxine, a synthetic T4 hormone used to treat hypothyroidism, represented as a graph. Each atom is represented as a node, and bonds between atoms are represented as edges. O = Oxygen, N = Nitrogen, I = Iodine, H = Hydrogen, and Carbon is so common in organic molecules that it is not indicated with its letter, C. Anywhere two edges come together without a printed letter, that atom is assumed to be Carbon. Single lines represent single bonds, while double lines represent double bonds. Additionally, the triangular bond indicates that that bond would come "out of the page" toward the reader.

(6) Implicit Valence: The number of implicit hydrogen atoms or other single-bonded atoms connected to the atom.

(7) Isotope: The specific isotope of the element (if applicable).

(8) Number of Radical Electrons: Electrons that are not paired.

(9) Aromaticity: Whether the atom is part of an aromatic system.

Similarly, edges (bonds between atoms) can have features like:

(1) Bond Type: This includes single, double, triple, or aromatic bonds, and is a fundamental descriptor of the bond's nature.

(2) Bond Stereochemistry: Information about the stereo configuration (e.g., cis/trans isomerism) of the bond, if applicable.

(3) Conjugation: Indicates whether the bond is part of a conjugated system, which is important for understanding electronic properties of the molecule.

(4) Ring Membership: Specifies if the bond is part of a ring structure in the molecule.

(5) Is the Bond in a Ring?: A binary feature indicating whether or not the bond is part of a ring.

(6) Bond Rotation: Whether the bond allows for free rotation or is locked due to double-bond character or ring constraints.

(7) Bond order: refers to the number of chemical bonds between a pair of atoms, and it indicates the stability and strength of a bond, as well as the arrangement of electrons in the bond.

As a brief note, while molecules can be naturally represented as a graph, we can also express them as a string, commonly called a Simplified Molecular-Input Line-Entry System, or SMILES string. The compound shown in Figure 2.4 can be equivalently represented with the string:

N[C@@H](CC1=CC(I)=C(OC2=CC(I)=C(O)C(I)=C2)C(I)=C1)C(O)=O

We will not go into the grammar of the SMILES representation, but refer the curious reader to the SMILES Wiki page.

2.2.2. Feature Encoding

Once we've identified the atom and bond-level features, they must be encoded into a format suitable for machine learning. This involves translating the chemical properties of atoms and bonds into numerical vectors (feature vectors) that can then be ingested by machine learning models. The most common way to do this is by using the RDKit package [**67**].

RDKit ingests a SMILES string and generates a molecular graph representation, expressed in the standard way with $G(V, \mathcal{E})$, where V is an $N \times d$ atom-feature matrix and $\mathcal{E}$ is the edge set containing the bond information.

There are many other properties one could consider, particularly as it relates to graph-level features like physiochemical properties, solvation properties, and attributes that define intrinsic chemical reactivity [**23**]. However, for our purposes, we will confine our attention to just atom and bond-level features, which can be more effectively leveraged by graph-based machine learning methods.

Now, for a drug compound to be effective, it first needs to physically get to the site in which it needs to work. That means it needs to be absorbed into the body, distributed through the body (to reach the target tissue), metabolized by the body (but not too fast!) and finally excreted (we don't want the drug in our system forever). Collectively, the factors of absorption, distribution, metabolism, and excretion are referred to as ADME and are a subset of properties known as *pharmacokinetic* properties. Think kinematics, like in physics, but in pharmacology space. How drugs move through the body.

2.2.3. Pharmacokinetics

In drug development, predicting pharmacokinetic ADME properties is crucial because a favorable ADME profile is required for any and every drug to be granted approval. For context, in the last decade, 90% of all drug failures were due to poor pharmacokinetic profiles [**14**]. As all relevant properties of a drug are determined by its structure, structure-aware graph machine learning can help in predicting how a drug will be absorbed, distributed, metabolized, and excreted in the body. An effective ADME property predictor helps enormously in filtering candidate compounds in the hit-to-lead phase (Figure 2.3) and streamlining the discovery process.

Initially (pre-1980s) ADME property prediction relied heavily on empirical observations and rule-based methods. Scientists used known properties of chemical compounds (derived from physical experimentation) and basic biophysical principles to predict how new compounds would behave. With the introduction of Lipinski's famous "Rule of Five" [**69**] in 1997, methods began to be developed to estimate drug-likeness based on molecular properties like molecular weight, lipophilicity, and hydrogen bond donors and acceptors.

Following the Rule of five methods, and riding the rising wave of computational capabilities, quantitative structure-activity relationship (QSAR) models marked the beginning of truly systematic computational approaches to ADME prediction, using statistical techniques to correlate the chemical structures of compounds with their pharmacokinetic properties. While more effective, these early QSAR models primarily utilized molecular descriptors (numerical values that describe the chemical properties of entire molecules) and linear regression or other simple statistical methods for prediction.

Eventually, as with many prediction tasks on structured data, the field converged on graph neural networks as the go-to solution to the ADME property prediction task. This transition revolutionized ADME property prediction because, unlike traditional machine learning models that relied on hand-engineered molecular descriptors, GNNs could learn directly from the graph structure of molecules, allowing for more accurate modeling of molecular interactions and properties.

This also brought with it the demand for greater interpretability. Without chemists hand-engineering relevant features, the predictions themselves, while better, were put under greater scrutiny. In the following years, many explainability and interpretability algorithms were generated, falling broadly under the instance-based or model-based categories [**117**]. We will not go into great detail here, but this is an issue we also address in our partition learning algorithm presented in Chapter 5. We've taken a slightly different tack than standard interpretability methods, which are typically post hoc and do not actually affect performance. Instead, our method uses an intrinsically interpretable algorithm to add interpretability *while improving performance.* For a complete explanation, see Chapter 5.

But getting back to ADME property prediction: why do these molecules have these properties? Why does one molecule behave one way in our body and another behave differently? It all has to do with how these molecules interact with proteins that drive all the processes of life.

2.2.4. Proteins

First, a quick pit stop for scale: small molecule drug compounds are made up of between 20-100 atoms, while proteins - biological entities built by our bodies - are composed of *tens to hundreds of thousands of atoms.* Indeed, they are composed of so many atoms that it's a bit silly to think of them in that way. Typically, we think of proteins as being made up of amino acids. Amino acids are smaller biological structures, humans have 21 different types, and a typical protein might be composed of between 50 and 2000 such amino acid blocks [**6**].

While we don't contribute methodologies specifically related to proteins in this dissertation, they are relevant and interesting enough to warrant a brief introduction.

Proteins play a crucial role in almost all biological processes in the body, and the interaction between proteins and small molecule drugs is a key aspect of modern therapeutics. As we've already mentioned, proteins are large, complex molecules made up of amino acids. The particular sequence of amino acids in a protein is what determines its structure and, just like with small molecules, its function.

Proteins perform a huge variety of functions in the body. They act as enzymes, catalyzing biochemical reactions necessary for life, and play roles in transport, communication, and immune response while also comprising the major structural elements of all cells [**52**].

For example, hemoglobin, a protein in red blood cells, transports oxygen throughout the body, while insulin, a hormone protein, helps regulate blood sugar levels. A particularly compelling and specialized class of protein are Class I Topoisomerases, shown in Figure 2.5, used to help untangle DNA in the nucleus. We will now embark upon a brief tangent - but these proteins are too impressive not to mention! For context, each of our

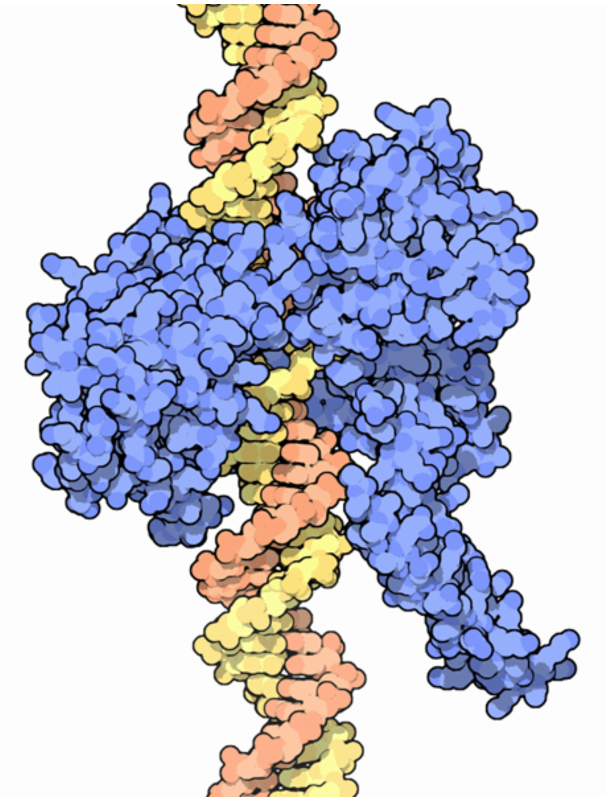

Figure 2.5. A Topoisomerase protein (blue) binds with a DNA helix.

cells packs two meters of DNA into a bundle on the order of microns (millionths of a meter) across. Inevitably, these strands get tangled, but in order to access the genetic information, they must be unwound. Enter: Topoisomerases.

Topoisomerases help to untangle strands of DNA in the nucleus by wrapping around the DNA at a damaged site, cutting one of the two strands, holding onto one strand while letting the helix relax (from over or under-winding), and finally reconnecting the broken strand, restoring the double helix [**3**]. One protein does all this. Ok, topoisomerase tangent: complete. The adeptness with which proteins solve astronomically (or, biologically!) hard problems in our bodies truly astonishing.

So, why do we care about proteins in drug discovery? Because proteins are the central actors in so called “pathways” in the body. A biological pathway is a series of actions among molecules in a cell that leads to a certain product or a change in the cell. These pathways are crucial for processes like metabolism, signal transduction, and gene regulation. They can be simple, involving a few steps, or highly complex, involving multiple layers of regulation and interaction. All of them rely on proteins to carry out these processes, and if we want to intervene, targeting the proteins is one way to do so.

Many diseases are caused by the dysregulation of these pathways. For example, cancer can result from the malfunction of signal transduction pathways that control cell growth and division. Targeting specific proteins in these pathways is a common strategy in drug development. For instance, drugs may inhibit overactive enzymes in a cancerous pathway.

In Figure 2.6, we show a cartoon of a pathway, with different states being arrived at through some actions. At the top, the healthy progression has been interrupted by some maladaptive action, resulting in a diseased state. At the bottom, we show how drug activity might put the pathway back on track so that it arrives at a healthy state instead.

As the function of proteins is determined by their structure, that is what we’re trying to change with small molecules. As we can see in Figure 2.5, proteins have a highly irregular, bumpy surface. Small molecules, if designed just right, can fit into one of those concave bumps and bond with the protein, changing its shape and thereby changing its function. These small, concave bumps are called pockets, and fitting a drug inside them is no easy task.

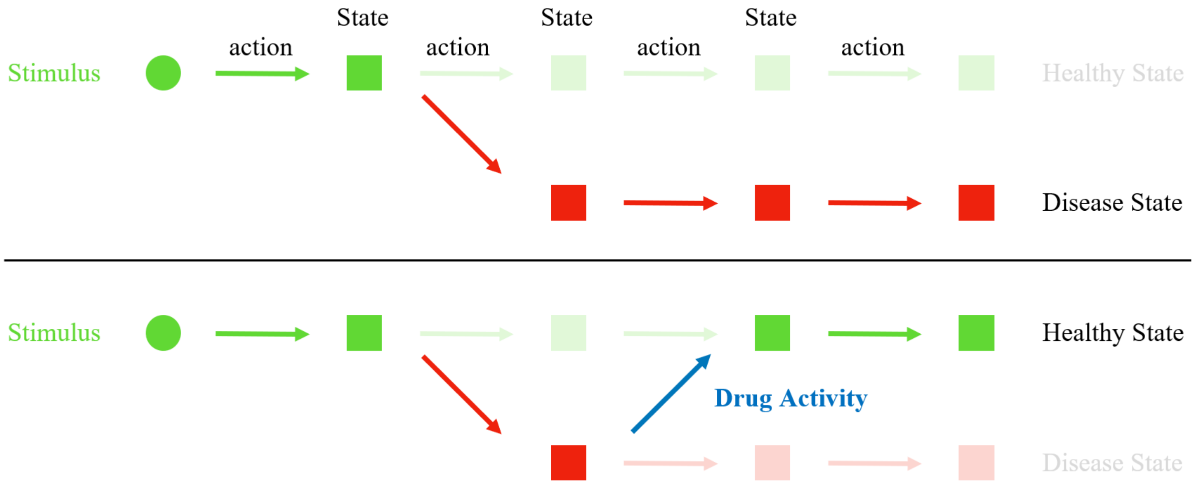

Figure 2.6. A cartoon depicting possible modifications to an abstract pathway. In the top, the healthy progression has been interrupted by some maladaptive action, resulting in a diseased state. In the bottom, we show how drug activity might put the pathway back on track so that it arrives at the healthy state instead.

2.2.5. Lock and Key

A common model used to describe how small molecules bind with proteins is known as the lock-and-key model. The protein pocket is a small, irregular, and very specific shape, like the series of pins in a lock. The small molecule, then, is like a key. If it has the right shape, it will nest itself inside the pocket and stay there, affecting how the protein functions.

While this is a simplified model, it is widely used to describe the challenging problem of protein-ligand binding (ligand is the technical term for a small molecule in this context).

2.2.6. Challenges of Learning in Chemical Space

Despite these advancements to machine learning methodologies we've just discussed, challenges remain. These challenges stem from both the complexity of biological systems and the limitations of current machine learning techniques.

Of particular note are limitations to the graph representation itself. While graph-based models have rapidly become state-of-the-art, they still face challenges in adequately capturing 3D molecular structures and interactions. This is primarily due to the difficulty of generating accurate conformers, or 3D shapes of the physical molecule.

Generalizing across chemical space also remains a challenge. Models trained on one set of compounds may not perform well on others, especially if the new compounds are structurally diverse from the training set. Among other biological reasons, this is driven largely by the concept of *activity cliffs* [**101**], which refers to the dramatically different behavior that can arise from only slight modifications to the structure of the chemical compound.

In terms of usability in a medicinal chemistry context, interpretability and explainability also remain an open problem. It's crucial to understand why a model makes a particular prediction, not only to understand it from a chemistry perspective but to improve upon and debug the models.

In this dissertation, we address the issues of generalization and interpretability by introducing a methodology for learning to extract and represent molecular *subgraphs* explicitly. Because discrete subgraphs, like functional groups appended to a backbone scaffold, are known to affect ADME property values in specific ways, learning to extract and represent these explicitly should help with generalization. Simultaneously, which subgraphs are extracted, paired with the weight assigned to them by the model, helps when it comes time to interpret the predictions.

——— * * * ———

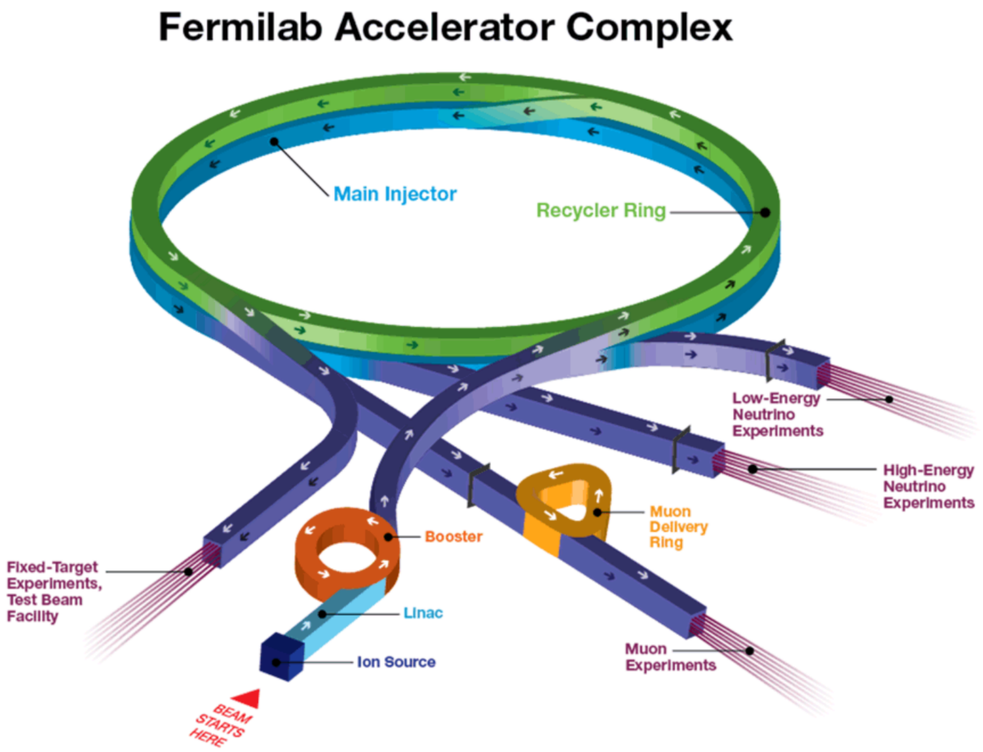
Fermilab Accelerator Complex
Main Injector
Recycler Ring
Low-Energy Neutrino Experiments
High-Energy Neutrino Experiments
Muon Delivery Ring
Booster
Fixed-Target Experiments, Test Beam Facility
Linac
Ion Source
Muon Experiments
BEAM STARTS HERE

Figure 2.7. A schematic of the Fermilab Accelerator Complex. In our work, we address problems relating to the two large rings, the Main Injector and Recycler Ring, as well as the Muon experiments.

2.3. Particle Accelerators

In Part III, we review select projects from our extensive research engagement with Fermilab: North America's particle accelerator complex.

As we will discuss, particle accelerators not only help us better understand how the world works, but their development has birthed such technologies as the Internet, MRI machines, and targeted cancer treatments. These are enormously interesting and important machines, and it has been an honor to get to work alongside the brilliant folks at Fermilab.

While our contributions in the machine learning space revolve around recurrent and convolutional machine learning methodologies, a rudimentary understanding of the accelerators under study is both helpful for comprehension and interesting in its own right. Plus, when else are we going to get to pack graph ML, computational chemistry, and particle accelerators all into one chapter? Let's get into it.

2.3.1. Why? How? What?

In the most basic terms, a particle accelerator does what it says on the tin: propels charged particles to high speeds, where *high* means *near the speed of light.* Slamming these particles into things (either colliding them with each other or directing them at a stationary target) creates conditions that wouldn't otherwise exist (or ones that haven't existed since the big bang) and lets physicists look under the hood at the rules governing the behavior of our universe. Over the last hundred years or so, we've used accelerators to build up what is called the Standard Model, our best theory to date about the fundamental particles and forces in the universe.

2.3.2. Fermilab

In North America, Fermilab[†], located just a few miles outside of Chicago in Batavia Illinois, is the premier accelerator complex and home to a wide array of machines and experiments. Figure 2.7[‡] presents a high-level diagram of the facility. What is not obvious from this Figure is the size of the facility: the blue and green paths corresponding to the

[†] Fermilab has a rich and exciting history, and I highly recommend anyone interested in learning more do so at their website: https://www.fnal.gov/

[‡] Figure 2.7 courtesy of Fermilab: https://www.fnal.gov/pub/science/particle-accelerators/accelerator-complex.html

Main Injector (MI) and Recycler Ring (RR) have circumferences of two miles. Arrows represent the direction of motion of the charged particles (sourced in the lower left). Buried deep underground and confined with strong magnetic fields, charged particles whip around the complex inside in steel tube called the beampipe. Along the way, they are accelerated, shaped, bunched, and redirected to the site of each experiment. Every second, particles will make around 100,000 trips around the Main Injector and Recycler [**53**].

The systems we are most interested in are 1) the Main Injector and the Recycler Ring, and 2) the Muon experiments. For more logistical than scientific reasons, the Main Injector and Recycler Ring share a tunnel. This leads to interesting and challenging problems that we've shown can be solved efficiently with machine learning.

2.3.3. Real-Time Edge AI for Distributed Systems (READS)

First and foremost, I would like to thank my collaborators, particularly Kyle Hazelwood, who provided countless hours of guidance, support and camaraderie as we turned an idea into a reality that promises to change how accelerator operators engage with these machines.

In Chapters 8 and 9, we will cover in detail the research in the following articles:

- K. Hazelwood†, R. Shi†, B.A. Schupbach†, **M. Thieme**†, M.R. Austin, M.A. Ibrahim, V.P. Nagaslaev, D.J. Nicklaus, A.L. Saewert, K. Seiya, R.M. Thurman-Keup, N.V. Tran, A. Narayanan, H. Liu, S. Memik, "Real-time Edge AI For Distributed Systems (READS): Progress On Beam Loss De-blending for the Fermilab Main Injector and Recycler," in *Conference IPAC'21.*

- K.J. Hazelwood†, **M. Thieme**†, J. Arnold, M.R. Austin, M.A. Ibrahim, V.P. Nagaslaev, A. Narayanan, D.J. Nicklaus, G. Pradhan, A.L. Saewert, B.A. Schupbach, K. Seiya, R.M. Thurman-Keup, N.V. Tran, D. Ulusel, H. Liu, S. Memik, R.

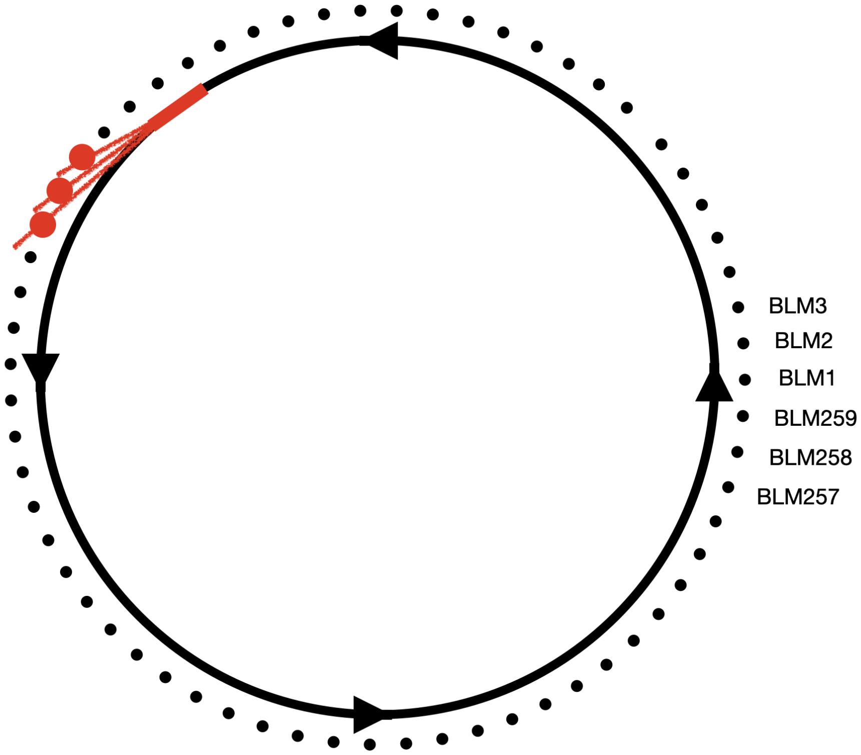

Figure 2.8. The solid circle represents an accelerator, with the arrows indicating the direction of travel of the particles. Dots around the outside represent (not to scale) the beam loss monitors spread evenly around the accelerator. When the beam scrapes against the edge of the beampipe, it will emit a spray of particles, shown here in red. The BLM's local to the event will detect and report the intensity of the radiative losses.

Shi, "Semantic Regression for Disentangling Beam Losses in the Fermilab Main Injector and Recycler" in *Conference NAPAC'22.*

- Shi, R., Ogrenci, S., Arnold, J. M., Berlioz, J. R., Hanlet, P., Hazelwood, K. J., Ibrahim, M. A., Liu, H., Nagaslaev, V. P., Narayanan, A., Nicklaus, D. J., Mitrevski, J., Pradhan, G., Saewert, A. L., Schupbach, B. A., Seiya, K., **Thieme, M.**, Thurman-Keup, R. M., and Tran, N. V. Ml-based real-time control at the edge: An approach using hls4ml. In the *31st Reconfigurable Architectures Workshop (RAW) 2024.*

†Equal Contribution

2.3.3.1. The Challenge. Excavating an underground tunnel two miles long is an expensive and time consuming task, but one that needed to be done to accommodate Fermilab's latest accelerator: the Main Injector. When the Main Injector had been built and the need arose for a place to store those accelerated particles, instead of digging a new two-mile loop for this new machine, they put it inside the existing tunnel (often referred to as an enclosure) that had already been dug for the Main Injector. This was a remarkable achievement and allowed for the completion of the project on budget and on schedule.

However, while these machines are almost unfathomably precise in their ability to guide a beam of particles traveling near the speed of light, they are not perfect. Occasionally, these particles will deviate from the center of the beampipe and scrape against the steel beampipe itself. This generates a spray of particles we refer to as "radiative beam losses", and we can detect them on an array of detectors known as Beam Loss Monitors (BLMs) that sit inside the tunnel next to the accelerators. See Figure 2.8 for a high-level overview. Investigating the readings on the BLMs allows operators of the machine to know when and where these deviations occur and take action accordingly.

The challenge comes from the shared nature of the enclosure: in periods of joint operation, when both MI and RR contain high intensity beam, radiative beam losses from MI and RR overlap on the enclosure's shared beam loss monitoring system, making it difficult to attribute those losses to a single machine. Incorrectly diagnosing the source results in unnecessary downtime that incurs both financial and experimental cost. It is our task to disentangle these overlapping beam loss profiles in order to pin down which machine needs to be adjusted or shut down. We do so with a novel neural approach taking the form of a continuous adaptation of the popular U-Net [**89**] architecture.

2.3.4. The Muon-to-Electron Experiment (Mu2e)

The second experiment we've contributed to at Fermilab is known as *Mu2e* [**2**]. I would like to recognize my brilliant counterparts at Fermilab, Aakaash Narayanan, and Vladimir Nagaslaev, for their invaluable guidance over the course of this project.

Work for this project was published in the following articles, the first two of which will be covered in detail in Chapter 10 and 11.

- A. Narayanan†, **M. Thieme**†, K.J. Hazelwood, M.A. Ibrahim, H. Liu, S. Memik, V. P. Nagaslaev, D.J. Nicklaus, P.S. Prieto, K. Seiya, R. Shi, B.A. Schupbach, R.M. Thurman-Keup, N.V. Tran, "Optimizing Mu2e Spill Regulation System Algorithms", in *Conference IPAC'21, Campinas, Brazil, 2021.*

- A. Narayanan†, J. Jiang†, **M. Thieme**†, J. Arnold, M. Austin, J.R. Berlioz, P. Hanlet, K.J. Hazelwood, M.A. Ibrahim, V. P. Nagaslaev, D.J. Nicklaus, G. Pradhan, P.S. Prieto, B.A. Schupbach, K. Seiya A. Saewert, R.M. Thurman-Keup, N.V. Tran, D. Ulusel , H. Liu, S. Memik, R. Shi, "Machine Learning for Slow Spill Regulation in the Fermilab Delivery Ring for Mu2e", in *Conference NAPAC'22.*

- Xu, C., YC. Hu, J., Jiang, J., Memik, S., Shi, R., Shuping, A. **M., Thieme**, M., and Liu, H. Beyond PID controllers: PPO with neuralized PID policy for proton beam intensity control in mu2e. In the *Machine Learning and the Physical Sciences Workshop, NeurIPS 2023.*

The Mu2e experiment at Fermilab is a high-precision physics experiment designed to explore the frontiers of the Standard Model. Its primary objective is to observe and study the relationship between electrons and muons (electrons' heavier cousin). Specifically, it is to study the rare *conversion* of a muon into an electron without the emission of neutrinos. This process, if observed, would be a clear indication of physics beyond the Standard Model, and help us better understand the particles (electrons) that quite literally keep the light of civilization burning.

†Equal Contribution

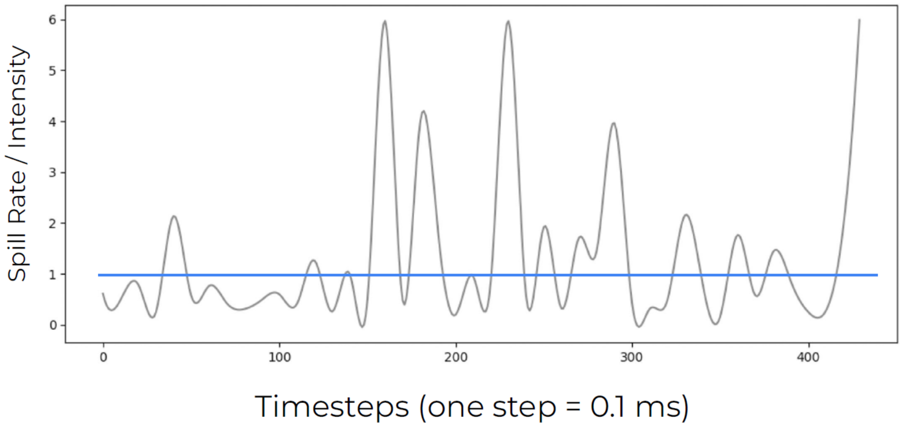

Figure 2.9. A single example spill, where the gray line indicates the extracted spill rate, or intensity, and the blue line reflects the ideal (but not achievable) extraction rate.

The Mu2e experiment intends to capture high-energy muons in Aluminum atoms and look for new physics in their decay to electrons. To help increase the signal strength, Mu2e demands pulses of muons arrive at the Aluminum target with strict requirements on the "rate uniformity", which just means the time variation in the intensity of the muon pulses. Because the signals we're looking for are so faint, this uniformity dictates the sensitivity of the experiment, and it is this uniformity that the research described in this dissertation is attempting to maximize.

To create the muons, proton pulses are slammed into a production target and muons are obtained from the secondaries, meaning that they are emitted as a byproduct of the protons impacting the production target. The proton pulses are created by the "slow extraction" of the bunched beam from the Delivery Ring, which is the process of cleaving off part of the proton beam circulating in the Delivery Ring and sending it down another beampipe to the experiment site. This extraction (or 'spill') of protons from the Delivery Ring is achieved using a process known as *third integer resonance extraction.* For our purposes, this is just the act of cleaving off part of the circulating beam we described above.

Our objective is to regulate the uniformity of the extracted spill - or increase its Spill Duty Factor (SDF) - by regulating this slow extraction process. The SDF is defined as follows:

$$SDF = \frac{1}{1+\sigma^2_{spill}} \tag{2.7}$$

where σ^2_{spill} is the variation in the *extraction rate*. For now, we can think of this extraction rate as *intensity*, as extraction rate per time is intensity. An example spill is shown in Figure 2.9, where the gray line indicates the extracted spill rate, or intensity, and the blue line reflects the ideal (but not achievable) extraction rate.

2.3.4.1. Extraction System. This extraction is governed by an extraction system composed of three main parts: the quadrupole magnet, the sextupole magnet, and the electrostatic septum. Figure 2.10 shows a visual representation of each component. Each magnet has a role: the quadrupole changes the shape of the circulating proton beam (more on this later), and the sextupole shapes and compresses the beam. The electrostatic septum is what actually cleaves the circulating protons off the primary beam and directs them to the downstream experiment. For our purposes, we assume that the sextupole and the electrostatic septum are static elements, and we will modify the current (and, transitively, the magnetic field strength) over the quadrupole. Functionally, we can think of the quadrupole as *squeezing* the circulating beam, changing its cross-sectional profile and thereby modulating how many protons are cleaved off by the electrostatic septum.

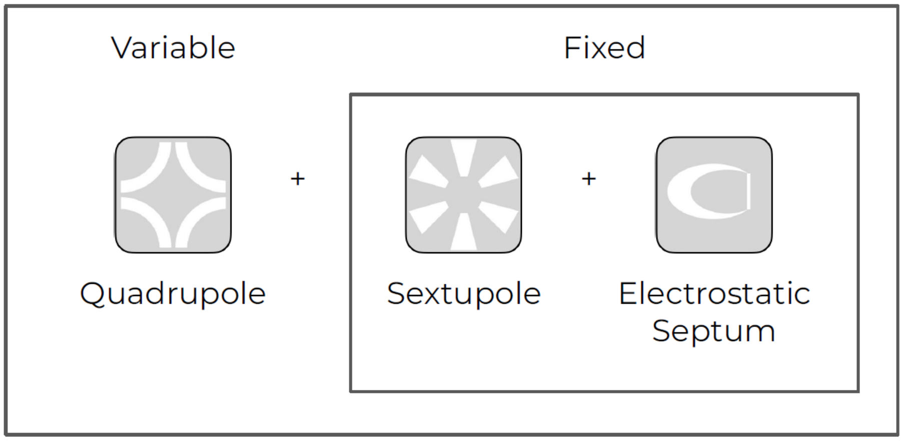

Figure 2.10. In our setting, we focus on modulating the quadrupole current to control the extraction rate. For our purposes, we consider the sextuple and electrostatic septum as fixed.

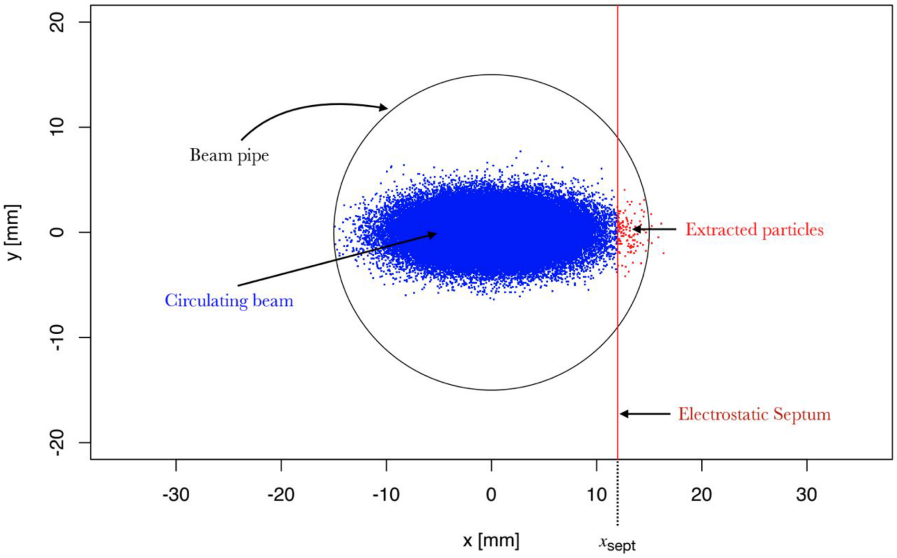

Figure 2.11. A snapshot of the beam in physical space at the extraction location. As the horizontal beam size increases, a slice of the circulating beam (that is past the position of the electrostatic septum) is extracted.

Figure 2.11[§] puts this into physical context.

[§] Figure courtesy of Aakaash Narayanan [**78**].

At present, regulating the extraction rate is accomplished with a standard 3-parameter PID controller. The inner workings of PIDs are a bit out of scope for this thesis, but suffice it to say, PID controllers are used across the sciences and engineering and are exceptionally effective and simple regulators. That being said, they are also linear and symmetric heuristic control systems, and that they are driven by constant parameters mean that they implicitly assume the response of the system is invariant across all operating regions. While there are ways of getting around some of these issues, e.g. stacking or cascading PIDs on top of one another, the heuristic nature of their operation remains. Due to the possibility of a nonlinear noise profile and variable response characteristics across operating regions, a *learnable* controller would be more desirable.

This is where we enter the fray.

2.3.4.2. Learning to Control. We would like to do one of two things to increase the uniformity of the extraction rate, 1) optimize the parameters of an existing PID controller, or 2) replace the PID controller entirely with a learnable controller.

Fortunately for us, the great folks at Fermilab had already solved the first problem and built a physical simulator of the entire system [**107**]. It was capable of simulating the random fluctuations in the intensity of the circulating proton beam and the effect of modulating the quadrupole current on the extraction rate. Our first contribution was to make this excellent simulator fully differentiable. More on that in Chapter 10. For an intuitive understanding of the computational setup, see Figure 2.12.

In Chapters 10 and 11, we will review in detail how we built our learning systems, how well they performed relative to the baseline PID, and some steps we might take to

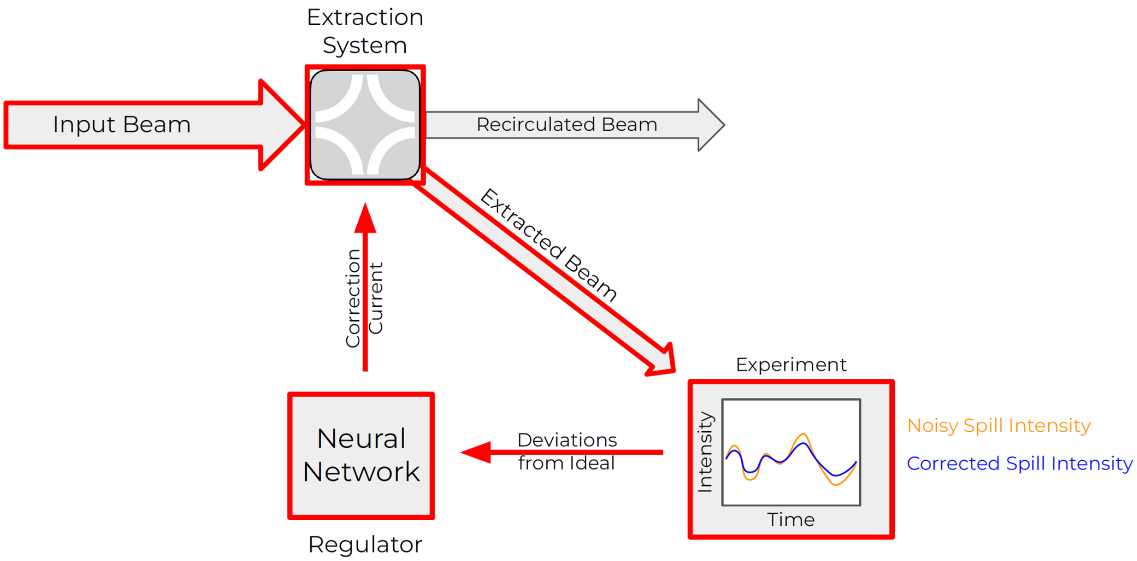

Figure 2.12. A diagram of the architecture that allowed for the optimization of existing PID parameters as well as the development of a fully learnable controller. Components shown in red were made to be differentiable, which allowed for the regulator (which is shown as a neural network here but could also be a PID controller) to be optimized using the standard machinery of gradient descent and backpropagation.

improve them further. The learning models themselves took the form of recurrent neural networks, adapted so as to accommodate our particular needs.

Part 2

Learning and Leveraging Graph Structure

3 Introduction to Part 2

The value of a scientific theory scales in proportion to the agreement between its predictions and reality. When predictions align with reality, we can be confident that the organizational structure set forth in the theory, i.e., which elements affect one another, as well as the nature of those relationships, mirrors some fundamental truth about the domain. In this thesis, we consider the training of machine learning models via gradient descent as an analog of this forward process of observation → theory → validation. In the context of machine learning, *observations* are simply the input data, and the *theory* is the learned mapping between those observations and the predictions made by the model. Then, we can think of gradient descent as the process of iteratively updating a model's theory, given the observations, to move its predictions closer to reality. In this way, as the predictions made by the model converge onto reality, we can be increasingly confident that the theory learned by the model mirrors some fundamental truth about the problem domain. There are, however, challenges with extracting and interpreting a theory from a model. In addition to the well-known challenge of inspecting the 'black box' of a deep neural network's weights, learned representations are affected by strong assumptions baked into the model's architecture. Based on the symmetries inherent to the problem domain, e.g., translation invariance, temporal invariance, or permutation invariance, we employ model architectures that confine the solution space to ones possessing those same symmetries, with, e.g., convolutional kernels, recurrent blocks, or message passing layers. While effective for improving generalization performance, these assumptions can affect how and what information flows backward from loss functions into the model. By mixing our assumptions into the learned abstractions, this process, in effect, *fogs the lens* that is the machine learning system, making the *theory* learned by the model more difficult to interpret.

Motivated by this problem, in Part 2 we look at methods for *clarifying the lens* of machine learning systems to improve the interpretability of the theory learned by our models as well as the generalization performance. We focus on graph-structured representations of data, both for their generality and because graph structures are well suited to extension into the natural sciences. Our work takes the form of architectures that disentangle unrelated tasks, algorithms that make fewer assumptions about structures in the data, and, broadly, methods that allow structure/partition-learning signals to flow more smoothly from losses to weights. Note that this does not mean doing away with all of our assumptions about the problem domain; without assumptions, learning isn't possible. Instead, it means rethinking the role that individual components play in our learning systems and working up from first principles to build graph-based algorithms that don't inherit unnecessary baggage.

3.1. Contributions to Graph Machine Learning

Part 2 covers three methods for leveraging graph structure to inform the scientific process.

The first is what we call the Graph Learning Attention Mechanism (GLAM) [**104**]. GLAM is designed to answer the question: which relationships actually matter? In science, a central task is to prune the spurious connections out of a theory. Maybe we thought that objects of type Y interact with those of type X but it turns out they actually don't, or at least that knowing about the behaviors of objects of type X isn't terribly informative when predicting the behavior of objects of type Y. This notion has been encoded into many GNN algorithms, particularly those based on attention like the GAT [**109**]. In attention-based GNNs, node aggregation functions implicitly rely on learned attention coefficients to scale the influence of incoming messages. However, as we'll show, mixing the distinct tasks of identifying useful edges (typically called structure learning) and generating useful node embeddings, is fraught with subtle issues. We show how disentangling these tasks allows attention-based GNNs to learn sparser, more effective structures. Not only is this more effective, computationally efficient, and grounded in first principles, it results in learned interaction structures that are sparser and more interpretable.

The second is what we call Topographic Pooling (TopoPool) [**102**]. TopoPool is designed to answer the question: in a graph, which nodes work together to affect some global, graph-level property in a predictable way? This is another central element of understanding complex systems because only rarely does a single element of a complex system dictate its properties and behavior. More often, it is groups of interacting elements that, through their interactions, influence the properties of the system. This was inspired by graph-level property prediction in the pharmaceutical space, where the interaction of groups of atoms in a molecule determines properties like reactivity, stability, biological activity, etc., but the relevance extends well beyond this domain. In materials science, properties of materials such as strength, conductivity, or heat resistance often depend on the interactions between their constituent particles; in the brain, behavior is also determined more by large networks of neurons that fire together than it is by any single neuron; and in ecosystems, which can be thought of as networks of species interacting with each other, discrete clusters within these networks are key to determining ecological balance, species interdependence, or the impact of environmental changes.

Finally, we introduce a method for generating *new* graphs having a desireable property distributions called Neural Atomic Replacement via Guided Molecular Adaptation (NAR) [**103**]. This methodology is designed to take the next step and ask: how can we *use* our understanding of some system - encoded in the weights of a neural network - to produce something new, having more desirable properties? We can think of this method as sitting at the intersection of science and engineering, as a means of transforming predictive models directly into generative ones. Why would we want to do this? In our case, we were motivated again by problems in drug discovery, namely lead optimization. It is often the case that a promising drug compound (a lead) is synthesized, tested, and found to be reasonably effective. However, it may be *more* effective if, say, it was less toxic, more absorbable, or metabolized more slowly. The question then becomes: how do we modify the molecule to improve these properties without unduly affecting all the properties that we already like about it? As the properties of a compound are dictated by its structure, we've opted to retain the (graph) structure and introduce a way to modify those properties without modifying the structure. We accomplish this by using predictive models in conjunction with the standard machinery of machine learning, backpropagation, to swap atom types *within* a given structure in such a way that it optimizes the predicted properties towards some targets of our choosing. We'll show that this is simple, effective, and capable of generating new, chemically valid compounds with uniformly better improved property values. We're also proud to say that one of these compounds generated by our method has been selected for synthesis and testing at AbbVie, a highly non-trivial milestone for any generative model in chemistry.

4 | Graph Structure Learning

Graph Neural Networks (GNNs) are powerful local aggregators, but their sensitivity to network structure leaves them vulnerable to noisy edges. In response, many GNNs include edge-weighting mechanisms that scale the contribution of each edge in the aggregation step. However, to account for neighborhoods of varying sizes, node-embedding mechanisms must normalize these edge weights across each neighborhood. As such, the impact of noisy edges cannot be fully eliminated without removing those edges altogether. Motivated by this issue, we introduce the Graph Learning Attention Mechanism (GLAM): a drop-in, differentiable structure learning layer for GNNs that separates the distinct tasks of structure learning and node embedding. In contrast to existing graph learning approaches, GLAM does not require the addition of exogenous structural regularizers or edge-selection heuristics to learn optimal graph structures. In experiments on citation and co-purchase datasets, we demonstrate that our approach can match state-of-the-art semi-supervised node classification accuracies while inducing an order of magnitude greater sparsity than existing graph learning methods.

4.1. Introduction

Local interactions govern the properties of nearly all complex systems, from protein folding and cellular proliferation to group dynamics and financial markets [**100, 26, 72, 83, 56**]. When modeling such systems, representing interactions graphically can improve model performance dramatically at both the local and global levels. Graph Neural Networks (GNNs) designed to operate on structured data have quickly become state of the art in a host of structured domains [**114**]. However, GNN models rely on the provided topologies representing meaningful relations, for example, the bonds between atoms in a molecule [**27**]. Additionally, to generate useful node embeddings, GNNs require permutation invariant neighborhood aggregation functions, many of which implicitly assume that neighborhoods satisfy certain homogeneity properties [**119**]. Therefore, if noisy edges are introduced, or if the neighborhood assumptions are not met, GNN performance can suffer.

To address both issues simultaneously, many GNNs include mechanisms for learning edge weights that scale the influence of neighboring features in the aggregation step. The Graph Attention Network (GAT) [**109**], for example, adapts the typical attention mechanism [**108**] to the graph setting, learning attention coefficients between adjacent nodes in the graph as opposed to between tokens in a sequence. We will argue in Section 4.2.1 that the demands of edge weighting (or structure learning) are in an inherent conflict with those of node embedding, and edge weighting mechanisms that are shared with node embedding mechanisms are not capable of eliminating the negative impact of noisy edges on their own.

In this work, we introduce a method for *separating* the distinct tasks of structure learning and node embedding in GNNs. Our method takes the form of a structure learning layer that can be dropped in front of existing GNN layers to learn task-informed graph structures that optimize performance on the downstream task. Our primary contributions are as follows:

(1) We introduce a principled framework for considering the inherent conflicts between structure learning and node embedding.
(2) Motivated by this framework, we introduce the Graph Learning Attention Mechanism (GLAM), a layer that, when used alongside GNNs, can separate the distinct tasks of structure learning and node embedding and improve GNN performance.
(3) We demonstrate the efficacy of the GLAM layer on real-world graph benchmarks, which show its capability for retaining high classification accuracies while inducing an order of magnitude greater sparsity than competing methods.

In contrast to existing structure learning methods [**70, 115, 21, 32, 96, 76**] GLAM does not employ any edge selection heuristics, exogenous structural regularizers or otherwise modify the existing loss function to accommodate the structure learning task. This makes it simpler to apply in existing GNN pipelines as carefully crafted and domain-specific objective functions do not need to be modified. To improve explainability and robustness, we learn the optimal graph at each layer. This not only improves performance and induces greater sparsity but allows us to observe the relationship between the data, the GNN aggregator, and the downstream task more clearly. Our approach is also scalable and generalizable to the inductive setting as it does not rely on optimizing a fixed adjacency matrix.

4.2. Preliminaries

As our method takes inspiration from the original GAT, we begin by reviewing the mechanism by which the GAT layer generates edge weights, as well as how those edge weights are used to aggregate neighborhood information. Understanding this mechanism is important to understanding our conceptual framework (Section 4.2.1) as well as the GLAM layer (Section 4.3).

Graph attention networks learn weighted attention scores e_{ij} for all edges between nodes i and $j, j \in \mathcal{N}_i$ where $\mathcal{N}_i$ is the one-hop neighborhood of node i. These attention scores represent the importance of the features on node j to node i and are computed in the following manner:

$$e_{ij} = \text{LeakyReLU}\Big(\vec{\mathbf{a}}^T[\mathbf{W}_{GAT}\vec{h_i} \parallel \mathbf{W}_{GAT}\vec{h_j}]\Big) \tag{4.1}$$

where $\vec{h_i} \in \mathbb{R}^F$ are node feature vectors, $\parallel$ is vector concatenation, $\mathbf{W}_{GAT} \in \mathbb{R}^{F' \times F}$ is a shared linear transformation for transforming input features into higher-level representations, and $\vec{\mathbf{a}} \in \mathbb{R}^{2F'}$ are the learnable attention weights that take the form of a single-layer feedforward neural network.

To ensure the attention scores are comparable across neighborhoods of varying size, they are normalized into attention coefficients α_{ij} using a softmax activation:

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{j \in \mathcal{N}_i} \exp(e_{ij})} \tag{4.2}$$

For stability and expressivity, the mechanism is extended to employ multi-head attention, and the outputs of the K heads in the final layer are aggregated by averaging:

$$\vec{h}'_i = \text{softmax}\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i}\alpha_{ij}^{k}\mathbf{W}_{GAT}^{k}\vec{h}_j\right) \tag{4.3}$$

In the next section, we explain why the normalization procedure in Eq. 4.2, while crucial for node embedding, is an impediment for structure learning.

4.2.1. Node Embedding vs. Structure Learning

At first glance, it may seem we could address the structure learning problem by simply thresholding the existing GAT attention coefficients α_{ij}. However, due to the need for neighborhood-wise normalization and permutation invariant aggregation, this would not be ideal. As a motivating example, consider the following: if we add three random edges per node to the standard Cora dataset [**74**], then train a GAT to perform semi-supervised node classification, we get a classification accuracy of 65.9%. If we manually set the edge weights for each of those newly added random edges to zero (note: the GAT is not able to learn this weighting on its own), the same GAT is able to achieve 79% accuracy. However, if we simply remove all the newly added random edges so the GAT does not consider them at all, the GAT can achieve 82.1% accuracy. In GNNs with permutation-invariant aggregation mechanisms, dropping noisy edges is more effective than learning to zero their edge weights.

To understand this result, it's important to understand why the GAT attention coefficients are calculated as the softmax of the attention scores. This softmax step serves two important purposes:

(1) It normalizes the attention scores into attention coefficients that sum to one, which ensures the sum of the neighboring representations (as defined in Eq. 4.3) is also normalized. This is a crucial function of any node embedding mechanism because it normalizes the distributional characteristics of $\vec{h}'_i$ not just across different neighborhoods but across neighborhoods of varying size. Without this, the downstream layers that ingest $\vec{h}'_i$ would need to account for widely variable magnitudes in the values of $\vec{h}'_i$, and performance would suffer.

(2) The softmax serves as an implicit, low-resolution structure learning device. To locate the maximum input element in a differentiable manner, softmax uses exponentials to exaggerate the difference between the maximum element and all of the rest. In graph attention networks, this means exaggerating the difference between the attention coefficient of the *single most important neighbor* vs. all the rest. This imbues the attention coefficients, and thus each node's neighborhood, with a soft sparsity that improves the learned node embeddings by minimizing the influence of all but the single most useful neighbor.

For these reasons, if our aim is to generate useful node embeddings for node-wise prediction, we should not do away with the softmax activation to normalize attention coefficients in each neighborhood. However, as our aim is to jointly learn useful node embeddings *and* the graph structure, this embedding mechanism alone is not sufficient. As discrete graph structure learning subjects local neighborhoods to a noisy evolution process as the network samples edges, we need to assess the value of each neighbor *independently*. When the value of a given neighbor is conditional on the present neighborhood, it is difficult to disentangle the relative value of one neighbor from another as the neighborhood evolves.

This is why the existing edge-weighting + node embedding paradigm is insufficient if our goal is to simultaneously learn node embedding and graph structure: the neighborhood-wise normalization (softmax over edge-weights) expresses each neighbor's importance *relative* to all the other nodes in the neighborhood, which is in direct conflict with the edge-wise independence requirements of structure learning.

To preserve the embedding advantages of GNNs while accommodating the conflicting demands of structure learning, we introduce the Graph learning Attention Mechanism (GLAM).

4.3. The Graph Learning Attention Mechanism (GLAM)

Like the GAT layer, the GLAM layer ingests the node features $\mathbf{h} \in \mathbb{R}^{N \times F}$ and the edge set $\mathcal{E}$, where N is the number of nodes, and F is the number of features per node. For each node i, we transform the node features h_i into higher order representations x_i using a shared linear transformation $\mathbf{W} \in \mathbb{R}^{F_S \times F}$:

$$x_i = \mathbf{W}(\vec{h_i}),\ x_i \in \mathbb{R}^{F_S} \tag{4.4}$$

For each edge between nodes i and j in the provided edge set $\mathcal{E}$, we then construct a representation for that edge by concatenating the node representations x_i and x_j. From here, our GLAM layer differs from the GAT layer. Using another shared linear layer $\mathbf{S} \in \mathbb{R}^{1\times F_S}$, we map these edge representations onto *structure learning* scores $\boldsymbol{\eta} \in \mathbb{R}^{|\mathcal{E}|\times 1}$, where the score η_{ij} for each edge becomes:

$$\eta_{ij} = \sigma\Big(\mathbf{S}\big[x_i \| x_j\big] + u\Big) \tag{4.5}$$

where $\|$ is vector concatenation, $u \in \mathbb{R}^1$ is i.i.d. noise drawn from a $\mathrm{U}(-0.5, 0.5)$ distribution centered about zero, and σ is a sigmoid activation allowing each η_{ij} to be interpreted as the probability of retaining the edge between nodes i and j. We note here that the noise term is not strictly required, but we found that it helped to smooth out the training.

Finally, we extend the GLAM layer to include K attention heads, and the final structure learning score η_{ij} for each edge becomes:

$$\eta_{ij} = \sigma\Big(\frac{1}{K}\sum_{k=1}^{K}\mathbf{S}^k\big[x_i \| x_j\big] + u\Big) \tag{4.6}$$

Next, we sample a discrete mask $M \in \{0,1\}^{|\mathcal{E}|}$ from the distribution parameterized by the structure learning scores $\boldsymbol{\eta}$. When applied to the given graph $\mathcal{E}$, we get a sparsified graph $M(\mathcal{E}) \rightarrow \mathcal{E}' \subseteq \mathcal{E}$. To sample the discrete values in M from the continuous probabilities in $\boldsymbol{\eta}$ differentiably, we use the Gumbel-Softmax reparameterization trick introduced by **[55]** and retain the edge between nodes i and j if $\eta_{ij} \geq 0.5$. This value was chosen because it splits the extreme values of η and was fixed in all our experiments.

This new graph $\mathcal{E}'$ is then used in place of $\mathcal{E}$ in the downstream GNN representation learner. If that downstream GNN were a GAT, using the equations from Section 4.2, the attention coefficients would become:

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{j \in \mathcal{N}'_i} \exp(e_{ij})}$$

$$\text{and} \tag{4.7}$$

$$\vec{h}'_i = \text{softmax}\left(\frac{1}{K} \sum_{k=1}^{K} \sum_{j \in \mathcal{N}'_i} \alpha^k_{ij} \mathbf{W}^k_{GAT} \vec{h}_j \right)$$

where the neighborhoods $\mathcal{N}'_i$ are defined by the non-masked edges in $\mathcal{E}'$ where $\boldsymbol{\eta} > 0.5$.

As each GNN model incorporates graph structure in its own way, we have ensured that GLAM produces a maximally general, differentiable mask on the given edges. This mask may be readily used to separate the structure learning tasks from node embedding regardless of the particular embedding mechanism.

For example, we incorporate the learned graph M into the GAT by trivially swapping the softmax over the attention coefficients with a sparse softmax that respects the masked edges (as shown in Eq. 4.7). Extending beyond the GAT, we could incorporate the learned graph into the widely used Graph Convolutional Network (GCN) [**65**], for example, by simply multiplying the adjacency matrix A by the mask M before the application of the renormalization trick. To apply M in arbitrary GNNs, we need only to insert it before the neighborhood aggregation step. Therefore, the GLAM layer may be utilized with arbitrary downstream GNNs without affecting the differentiability of the model.

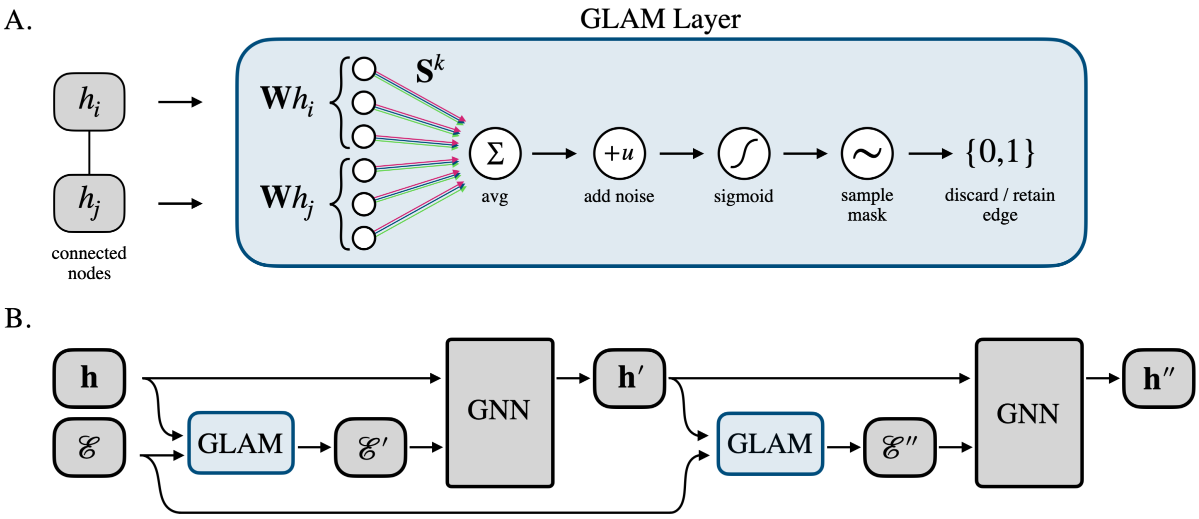

Figure 4.1. **A**: An illustration of the Graph Learning Attention Mechanism operating on a single pair of connected nodes with features $h_i, h_j \in \mathbb{R}^F$, using the shared weight matrix $\mathbf{W} \in \mathbb{R}^{F_S \times F}$ and multi-headed attention with $K = 3$ heads. **B**: A high-level overview of how the GLAM layer is used to learn optimal graph structures at each layer. The input is the original graph with node features $\mathbf{h} \in \mathbb{R}^{N \times F}$ and edge set $\mathcal{E}$. To make as few assumptions about the optimal graph structure as possible, we feed the original edge set $\mathcal{E}$ into each GLAM layer to reassess the utility of each edge at each layer. We note that this is optional, and chaining edge sets across layers is also possible.

4.4. Experiments

To investigate the efficacy of the GLAM layer, our experiments address the following questions:

- Q1: How does the GLAM layer affect the accuracy of a GNN on node classification tasks?
- Q2: What degree of sparsity (if any) is induced by the GLAM layer and how does that induced sparsity compare with competing methods?

Q3: As we use the GLAM layer to learn optimal graph structures layer-wise, how does that induced sparsity vary across layers?

In all experiments, we report performance in semi-supervised node classification tasks and induced sparsity on real-world graph datasets. As it is well known that GNN performance is degraded by heterophilic graphs (in which adjacent nodes tend to have different labels), we consider only those graphs where the edges bring material value for GNNs, i.e., homophilous graphs. Formally, the edge homophily ratio of a graph is a value in the range [0, 1] that denotes the fraction of edges in the graph that join nodes with the same labels:

$$H(G, \{y_i; i \in \mathcal{V}\}) = \frac{1}{|\mathcal{E}|} \sum_{(i,j)\in\mathcal{E}} \mathbb{1}(y_i = y_j) \tag{4.8}$$

Where G is an input graph with nodes $v \in V$ having labels y. While homophily isn't an unrealistic assumption, with most graphs being constructed this way [**75**], a method for graph adaptation based on the assumptions of the downstream GNN is still desirable as many GNN models implicitly assume a high degree of homophily and perform poorly when this assumption is violated [**119**]. In graphs with low homophily, such as the WebKB* datasets, GNN performance is often optimized by removing nearly every edge. For this reason, to get a better understanding of the GLAM layer's efficacy, we confine our study to the homophilic graphs detailed in Table 5.1.

Cora, PubMed, and Citeseer ([**74**], [**94**], [**37**]) are citation datasets with a relatively low average degree. Nodes correspond to documents (academic papers) and edges represent

*http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-11/www/wwkb/

	Type	Nodes	Edges	Features	Classes	$H(G)$
Cora	Citation Network	2,708	13,264	1,433	7	0.83
PubMed	Citation Network	19,717	108,365	500	3	0.79
Citeseer	Citation Network	3,307	12,431	3,703	6	0.71
Amazon Photo	Co-Purchase	7,487	245,812	745	8	0.84
Amazon Computers	Co-Purchase	13,385	505,474	767	10	0.79

Table 4.1. Homophilic graph datasets used in our experiments.

citations between these documents. Node features are the bag of words representing the document, and the task is to classify each document by its topic. Amazon Photo and Amazon Computers [**97**] are co-purchase datasets with a much higher average degree, where each node corresponds to a product and two nodes are linked if those products are frequently bought together. The node features are also a bag of words representations but of the reviews of each product. Similarly, the task is to classify each product into its product category.

Two methods exist which adapt the graph attention mechanism to learn task-informed, sparsified subgraphs: Graph Stochastic Attention (GSAT) [**76**] and Sparse Graph Attention Networks (SGAT) [**115**] (described in greater detail in Section 4.5). As GSAT was not evaluated on our datasets, and SGAT more closely resembles GLAM in design and functionality (inducing an edge-sparsified graph), we confine our direct comparison to SGAT. To ensure fairness and clarity in our comparisons, we use the same canonical splits in each dataset: 20 nodes per class for training, 500 nodes for validation and 1,000 for testing.

4.4.1. Training Methodology

In all our experiments, we train a two-layer GAT, using the GLAM layer to learn optimal structures for each GAT layer. Crucially, to demonstrate how the GLAM layer may be 'dropped in' to an existing GNN model with little to no modification to the training scheme, we use the known optimal hyperparameters and training methodology for the GAT layers (as reported in [**109**]) without modification. This covers all components of the model, including loss functions, regularization, optimizers, and layer sizes. The only hyperparameters we modify are those associated with the inserted GLAM layers, and, as mentioned in Section 4.3, the learned mask is enforced using a sparse softmax activation in the GAT layers that respects the mask M.

As the Co-Purchase datasets were not tested in the original GAT paper, we began optimization with the same hyperparameters as Cora and found that increasing the hidden dimension from 8 to 32 led to optimal performance. In all cases, we use cross entropy as our loss function and the Adam optimizer [**61**] to perform gradient descent. In all experiments, we report the average classification performance and induced sparsity over 10 independent trials (Table 4.2). All hyperparameters are available on GitHub.

Additionally, to make as few assumptions about the optimal graph as possible, we use the GLAM layer to independently assess the utility of each edge at each layer, as shown in Figure 4.1B, and report the induced sparsity at each layer. This is in contrast to existing methods which learn a single graph at the first layer that is then reused for all the downstream layers. As we will discuss more in Section 4.4.2, learning the optimal graph at each layer not only improves performance and induces greater sparsity but allows us to observe the relationship between the data, the GNN aggregator, and the downstream task more clearly.

We note here that the use of exogenous regularizers and edge-selection heuristics, as employed by other structure learning methods, is to help the structure learner better adapt to the demands of the downstream GNN. With the GLAM layer, we enable this same sort of adaptation by simply making these layers more sensitive than the downstream GNN layers. To do so, we relax the regularization (weight decay) and amplify the learning signal (increase the learning rate) for just the GLAM layers. Doing so for only the GLAM layers allows them to adapt to the GNN representation learner without the need for additional terms in the loss function. These layer-wise changes are slight, and optimal values vary by dataset. All layer-specific learning rates and regularization coefficients are outlined in the hyperparameter configuration files at https://github.com/MattsonThieme/graph-learning-attention.

Finally, as self-loops have disproportionate utility in any node-wise prediction task, the GLAM layers only attends over those edges that adjoin separate nodes, retaining all self-loops by default.

4.4.2. Classification Accuracy and Induced Sparsity

On semi-supervised node classification datasets, the GLAM layer enables the GAT to reach similar accuracies while inducing an order of magnitude greater sparsity than existing methods. This increased sparsification is notable as we do not explicitly enforce sparsity constraints or penalize retained edges.

Dataset	**Cora**	**PubMed**	**Citeseer**	**Photo**	**Computers**
MLP	49.5%	68.1%	46.9%	70.5%	53.1%
GAT	82.4%	78.2%	70.9%	89.1%	81.5%
SGAT	83.0%	78.3%	71.5%	89.9%	81.8%
GLAM	82.4%	78.6%	70.5%	90.3%	83.0%
% Edges Removed (layer 1 / layer 2)					
SGAT	2.0, -	2.2, -	1.2, -	42.3, -	63.6, -
GLAM	22.3, 2.2	30.0, 0.3	46.3, 1.4	63.1, 13.2	55.0, 8.7

Table 4.2. Top: Semi-supervised node classification accuracies are listed in percent. Bottom: Percentage of edges removed at the first / second layer. For SGAT, a single graph is learned at the first layer and reused it for all following layers. In our experiments, the GLAM layer is used to learn an optimal graph at each layer.

To answer explicitly the questions posed in Section 4.4:

A1: The GLAM layer improves the classification performance of an unmodified GAT on semi-supervised node classification tasks.

A2: The GLAM layer induces an order of magnitude greater sparsity than competing methods (while preserving classification accuracies).

A3: The induced sparsity is greatest in the first layer and drops off from there.

As the GLAM method is fundamentally about inducing task-informed subgraphs, and classification accuracies are similar to those yielded by SGAT, we confine our analysis to the sparsification aspect of the results. Complete results can be found in Table 4.2.

On each of the citation datasets, the GLAM layer is competitive with SGAT on prediction performance while inducing over an order of magnitude greater sparsity. This degree of sparsity has not yet been induced in these datasets while preserving SOTA classification performance. As such, we view these results as indicating, in addition to the efficacy of the GLAM layer, that there may be more redundant edges in these datasets than previously believed. Part of the advantage of the GLAM layer is that we can train it using only the signal from the downstream task. As such, the induced graph is a reflection purely of the relationship between the data, the GNN aggregation scheme, and the downstream task. Following this, we note that the first GLAM layer trained on these datasets removes closer to $1 - H(G)$ percent of the edges. There is not an exact correspondence, but the proximity between these two quantities is likely a reflection of the GLAM layers learning that the downstream GAT performs best on neighborhoods with higher homophily. The second GLAM layer removes on the order of 1% of the edges, which is similar to SGAT. We hypothesize that more edges are retained in the second and final layer due to the node representations already containing information from their 1-hop neighborhood, and there being additional value in aggregating information from the 2-hop neighborhood.

On the co-purchase datasets, which have much larger average degrees, the GLAM layers remove substantially more edges in the first layer but fewer in the second. We emphasize here that as there are no structural regularizers, GLAM is not encouraged to learn the sparsest graph possible, but rather the graph which optimizes downstream task performance. As we can see with the Computers dataset, retaining a few more edges in the first layer and a few fewer in the second resulted in substantially higher classification accuracies.

While our method does not always achieve state-of-the-art classification performance, we do achieve similar performance with far greater sparsification and without changes to the canonical loss function. As the GLAM layer is differentiable and can be readily integrated with arbitrary downstream GNNs, we believe it could be widely useful and is worthy of further study.

4.5. Related Work

The structure learning literature includes many effective methods for inducing task-informed, sparsified subgraphs. However, each of them relies on either edge-selection heuristics (such as top-k selection) or exogenous structural regularizers (such as penalties on retained edges) to stabilize the structure learning process. In contrast, our approach requires none of these, making it more readily deployable in existing GNN pipelines and obviating the need to interfere with carefully crafted objective functions or training methodologies.

Approaches such as Neural Relational Inference [**64**] and Graphs for Time Series [**96**] introduce methods for learning probabilistic graph models optimized for various time-series forecasting tasks. However, while both induce sparse subgraphs, they rely on exogenous regularization (penalties on retained edges) and suffer from scalability issues as both learn fixed adjacency matrices.

More closely related to our approach are the Graph Stochastic Attention, GSAT [**76**] and Stochastic Graph Attention, SGAT [**115**] models, which also adapt the graph attention layer to learn sparse subgraphs. SGAT attaches a binary gate to each edge in the given graph and then learns a single adjacency matrix that is shared across all layers. While effective, this adjacency matrix makes it difficult to scale, and they employ an L_0 norm in the loss function to penalize retained edges.

GSAT [**76**], based on the graph information bottleneck principle [**113**] is designed to be an interpretable graph learner and uses a similar sparse attention mechanism to assess the utility of each edge. However, in spite of not using sparsity constraints such as L_0 norms, they do add additional structural regularization terms into the loss function. Additionally, learnable weights in GSAT are shared between the GNN used to generate the binary mask and the GNN used for downstream node embedding. As we reviewed in Section 4.2.1, the demands of structure learning conflict with those of node embedding, so sharing weights across these networks may reduce performance on some tasks.

4.6. Discussion

We presented a principled framework for considering the graph structure learning problem in the context of graph neural networks, as well as the Graph Learning Attention Mechanism (GLAM) a novel structure learning layer motivated by this framework. In contrast to existing structure learning approaches, GLAM does not require exogenous structural regularizers nor does it utilize edge-selection heuristics. In experiments on citation and co-purchase datasets, the GLAM layer allows an unmodified GAT to match state-of-the-art performance while inducing an order of magnitude greater sparsity than other graph learning approaches.

There are several improvements that could be made to the method, especially as it relates to stabilizing the structure learning layers without augmenting canonical loss functions. Future research could be directed at additional methods for stabilizing the structure learning layers in ways that do not rely on additions to existing loss functions. These may include changes as simple as decaying the learning rate or early-stopping in a layer-wise manner. We can also explore the GLAM layer as a means of assessing the capacity for a novel GNN aggregation mechanism to productively aggregate information on heterophilic neighborhoods. For example, we would expect that an aggregation mechanism like the one presented in the CPGNN model [**119**] would yield a higher number of retained edges on heterophilic datasets than the GAT or GCN.

Finally, as our approach yields graph structures uncorrupted by the influence of exogenous heuristics, we also see the potential for its use in the design and analysis of novel GNN architectures. In the GLAM layer, edges are retained or discarded based on whether the downstream GNN can make productive use of them. By interpreting the distributions of retained edges at each layer, designers could, for example, better understand how well some aggregation scheme integrates information from heterophilic vs. homophilic neighborhoods. In a similar way, GLAM may be used as a principled method for assessing the value of depth in GNNs, a persistent issue due to the over-smoothing problem [**82**]. If a GNN layer were to no longer benefit from the incorporation of structural information, i.e. the number of retained edges yielded by its preceding GLAM layer was close to zero, then the GNN may be too deep, and adding additional layers may introduce complexity without increasing performance.

In the next chapter, we shift our focus to a novel algorithm that retains the provided connectivity structure and learns to cluster the nodes into meaningful substructures.

5 | Graph Partition Learning

Within molecules and proteins, discrete substructures affect high-level properties and behavior in distinct ways. As such, explicitly locating and accounting for these substructures is a central problem when learning molecular or protein representations. Typically represented as graphs, this task falls under the umbrella of graph pooling or segmentation. Given the highly variable size, number, and topology of these substructures, an ideal pooling algorithm would adapt on a graph-by-graph basis and use local context to locate optimal pools. However, this poses a challenge where differentiability is concerned, and each of the *learnable* graph pooling methods proposed to date must make strong a priori assumptions in regard to the number or size of the learned pools. As such, demand remains for a graph pooling algorithm that can maintain differentiability while retaining adaptability in the size and number of learned pools. To meet this demand, we introduce the Topographical Pooling Layer (TopoPool): a differentiable, hierarchical graph pooling layer that learns an arbitrary number of varying-sized pools without making any a priori assumptions about their number or size. Additionally, it naturally uncovers only connected substructures, increasing the interpretability of the learned pools and obviating the need for exogenous regularizers to enforce connectedness. We evaluate TopoPool on diverse molecular and protein property prediction tasks, where we achieve competitive performance against existing methods. Taken together, TopoPool represents a novel addition to the graph pooling toolbox and is particularly relevant to areas such as drug design, where locating and optimizing discrete, connected molecular substructures is of central importance.

5.1. Introduction

Graph neural networks (GNNs) have become the de facto models for learning representations of graph-structured data and have revolutionized structural biology [**95**], drug design [**38**], and molecular property prediction [**114**]. To facilitate graph-level predictions, where node and edge features must be collapsed into a single graph-level feature, a wide variety of learnable graph coarsening methods have been introduced [**33, 42, 116, 68, 25**] each with their own set of trade-offs. Heuristic pooling methods often fail to capture the complex structural information in the graph, while learnable methods are often expensive and sensitive to the specific choice of parameters. Learnable methods must also contend with maintaining differentiability. To do so, methods to date have had to make strong a priori assumptions about either the number or size of the learned pools, a severe disadvantage in domains where the optimal pool size is either unknown or variable across examples. The pharmacokinetic properties of molecules, for example, depend on discrete, connected molecular substructures that vary greatly in size, shape, and relative position within the molecule [**34**]. In this setting, an ideal graph pooling algorithm would be able to dynamically adapt to the input graphs and learn both the size and the number of clusters on a graph-by-graph basis.

In this work, we introduce the Topographic Pooling layer (TopoPool). TopoPool is the first differentiable graph pooling layer that 1) dynamically adapts to the input graphs and learns the optimal number and size of pools on a graph-by-graph basis. TopoPool also naturally locates only *connected* substructures, obviating the need for exogenous regularizers and loss terms required by other learnable methods to enforce the notion that nearby nodes should be pooled together. To our knowledge, as of the time of writing, no other graph pooling approaches are adaptive in this way. Each either sets an upper limit on the number of clusters [**116, 57**] samples a pre-defined number/ratio of ranked nodes or edges [**88, 25, 57, 33, 8**], and requires exogenous regularizers to enforce connectivity in the learned pools. Our contributions are as follows:

- We introduce TopoPool, the first hierarchical graph pooling algorithm that pools entire graphs without making assumptions about the number or size of the learned pools.
- In experiments on real-world molecular datasets, where graph pooling is particularly challenging, we demonstrate competitive performance against the existing approaches.
- We provide an efficient PyTorch implementation of the TopoPool layer that can be easily integrated into existing GNN pipelines.

Our choice to focus on molecular datasets was motivated by the field of medicinal chemistry, where the presence and relevance of pharmacophores [**43, 91**] serves as a prime example of the importance of preserving connected substructures in graph-based molecular representations. Pharmacophores are something like molecular phonemes: canonical ensembles of atoms used to construct larger molecules. The particular structure and position of the pharmacophores within a molecule is critical for determining a molecule's biological activity [**87**]. Thus, understanding how these substructures affect the pharmacokinetic properties of a molecule is crucial for drug discovery, design, and optimization [**92**].

5.2. Preliminaries

We denote a graph G with nodes $V \in \{v_0, \ldots, v_N\}$ and edge set $\mathcal{E}$ as $G(V, \mathcal{E})$. Graph pooling, or coarsening, is the process of mapping $|V| = N$ nodes onto a new set of nodes $|V'| = M$, where $M \leq N$. To accomplish this, graph pooling operators possess some combination of the following three functions: Selection, Reduction, and Connection (SRC) [**42**]. Selection is the process of grouping the input nodes, Reduction is the process of aggregating and compressing those groups into new representations, and Connection is the process of reconnecting the clustered nodes given the original edge set $\mathcal{E}$. The algorithmic contributions of TopoPool primarily lie in the *Selection* phase, but we also perform Reduction and Connection to form a fully self-contained graph pooling operator.

Being a differentiable graph pooling layer, TopoPool is designed to be used alongside GNN layers. However, TopoPool is agnostic to the particular GNN aggregation mechanism, and we can abstract this portion away, denoting an arbitrary GNN layer as $\text{GNN}(V, \mathcal{E}) : x \in \mathbb{R}^{N \times d} \to \mathbb{R}^{N \times d'}$, where d' is the dimension of the transformed node representation. As outlined in the following section, it is these transformed node representations, alongside the edge set, that the TopoPool layer ingests.

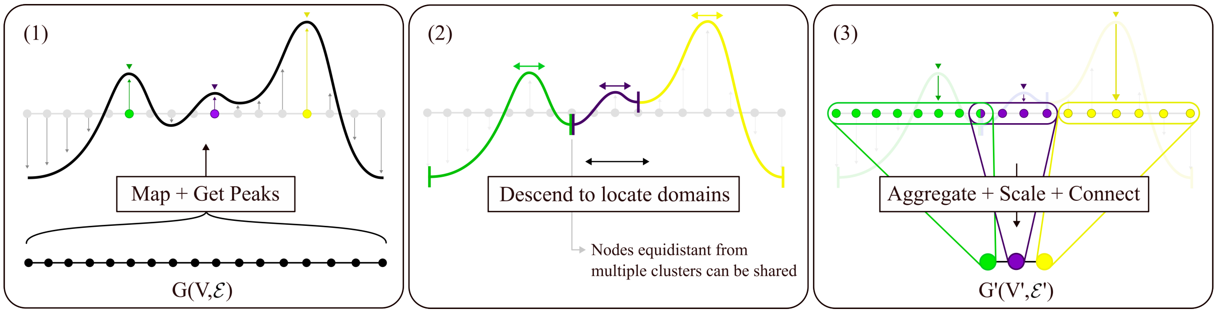
(1)
Map + Get Peaks
G(V,$\mathcal{E}$)
(2)
Descend to locate domains
Nodes equidistant from multiple clusters can be shared
(3)
Aggregate + Scale + Connect
G'(V',$\mathcal{E}$')

Figure 5.1. A high-level overview of the TopoPool algorithm. In panel (1), we generate the scores, which, in conjunction with the graph structure, define the 'topography' of the graph. Panel (2) shows how, once we've found the peaks in the topography, the pools are located by descending from the peaks until we reach a trough. This allows the layer to locate pools of variable size that are necessarily connected. Finally, panel (3) shows how the pooled clusters are reduced to the coarsened graph G'. Note that we visualize the algorithm here on a 1D chain graph for clarity only. The TopoPool algorithm can be applied on graphs with arbitrary topologies.

5.3. The Topographical Pooling Layer (TopoPool)

The TopoPool layer is a differentiable, hierarchical graph pooling algorithm for locating and aggregating an arbitrary number of discrete, connected substructures in an input graph. It was motivated by the desire to locate and extract pharmacophores from molecular graphs and inspired by the clean boundaries described in topographic maps. In this section, we describe how the TopoPool layer differentiably coarsens a graph without making any assumptions about the number or size of the clusters.

TopoPool takes as input a vector of node representations $h \in \mathbb{R}^{N\times d}$ and the edge set $\mathcal{E}$, and yields a new graph with nodes V' with features $h' \in \mathbb{R}^{K\times d}$ and edges $\mathcal{E}'$, where K is the number of learned clusters.

Select: The first step in the TopoPool algorithm is to generate the topography (Figure 5.1, (1)). Using a single linear layer $f_\theta(\cdot) \in \mathbb{R}^{d\times 1}$, TopoPool maps the node representations

$h \in \mathbb{R}^{N \times d}$ onto a scalar valued graph signal $f_\theta(h) = s \in \mathbb{R}^{N \times 1}$. These scalar values are what define the graph signal that, in conjunction with the network structure, we treat as a topography over the graph. Once s is generated, we locate all the peak nodes with a simple message passing operation, returning `True` in a binary mask for all $s_i : s_i > s_j, \forall j \in \mathcal{N}_i$, where $\mathcal{N}_i$ is the one-hop neighborhood of v_i. We denote the set of all such local peaks as P, where $|P| = K, K \leq N$, and each peak node $v_k \in P$ defines the source of its own unique cluster. Then, from each source node v_k, we *expand and descend*, growing each cluster outward while the neighboring node scores $s_j \leq s_k$ (Figure 5.1, (2)). We refer to the new clustered nodes by their peak node of origin, k, and accumulate constituent members of cluster k in a set C_k. In this descent process, nodes are assigned to clusters on a first-come, first-served basis. However, if some node is equidistant from multiple peaks and therefore reached in the same iteration of the descent phase, it is assigned to both clusters, as shown in Figure 5.1, panel (2). Experimentally, we found that allowing multiple clusters to share nodes does not impact performance. Therefore, for simplicity, we do not penalize or avoid the double counting of these shared nodes.

Reduce: Once clusters have been located and every node assigned, members of each cluster k are aggregated (Figure 5.1, (3)) with a max pool over their features, yielding a representation $h_k = \max_i(h_{i \in C_k})$. Each cluster representation h_k is then scaled by the score s_k on its peak node v_k. This step is important for a few reasons: 1) it allows gradients to flow back to the linear layer $f_\theta(\cdot)$; 2) it improves prediction performance by scaling the influence of each cluster; and 3) it adds a degree of interpretability be clearly showing the relative differences in importance between the learned clusters. To ensure consistency across graphs with varying numbers of clusters, we normalize these peak scores s_k over each graph with a softmax, such that the final representation for the k^{th} pool becomes:

$$h_k = \frac{e^{s_k}}{\sum\limits_{i \in P} e^{s_i}} \cdot \max_i(h_{i \in C_k}), \quad h_k \in \mathbb{R}^{1 \times d} \tag{5.1}$$

The node representations for the final pooled graph are the concatenation of all the pooled representations: $h' \in \mathbb{R}^{K \times d}$.

Connect: To connect the learned clusters and generate the new edge set $\mathcal{E}'$, we simply draw an undirected edge between two clusters if there existed at least one edge between their constituent nodes in the input edge set $\mathcal{E}$.

A note on scalability: The TopoPool layer is inherently scalable as the complexity of the *expand* portion algorithm is the same as BFS. The cost for this step is $\mathcal{O}(N + E)$, where N and E are the number of nodes and edges in each graph, respectively. We note, however, that this is the worst case, where only a single cluster is learned. In most cases, the actual runtime will be $\mathcal{O}(C_{MAX} + E)$, where C_{MAX} is the size of the largest learned cluster, and typically $C_{MAX} << N$. This runtime is on par with benchmark graph pooling methods (Section 5.4.2) and ours runs in comparable time.

5.4. Experiments

In our experiments, we aim to answer the following questions:

Q1 - Is TopoPool competitive with existing learnable and heuristic graph pooling operators?

Q2 - Can the TopoPool layer be easily integrated into existing GNN pipelines?

Q3 - Are the learned clusters interpretable and consistent across examples?

Following the example of [**68, 116, 25, 88**], we benchmark our model on molecular and protein property prediction tasks. High-level descriptions of the datasets can be found in Table 5.1.

High-level descriptions of the datasets can be found in Table 5.1.

Two of our datasets, PPBR and Caco-2, come from the Therapeutic Data Commons (TDC) [**50**], a cross-modality repository of therapeutic data designed to evaluate AI capabilities across the stages of discovery. Molecules in each dataset are stored as SMILES strings and converted to a graph structure before training using the RDKit library. As edges in molecular graphs represent bonds, they are all undirected. Node features in the TDC datasets are a concatenation of 1-hot vectors expressing the atom type, degree, formal charge, chiral tag, and aromaticity; 39 features in all per node. For both datasets, we report the mean absolute error (MAE) on the test set.

Additionally, as the real-world drug discovery process yields increasingly diverse and increasingly novel compounds over time [**28**], our objective must be to learn a model that generalizes not only to new molecules but to molecules composed of unseen substructures. Therefore, to make our evaluations as realistic as possible, we use the canonical scaffold splits (defined in the TDC databases for all tasks) that force the training and testing sets to have maximally dissimilar molecular structures.

PPBR [**1**] stands for Plasma Protein Binding Rate, and the PPBR dataset labels drug molecules by the percentage of that drug bound to plasma proteins in the blood. This rate inversely affects the efficiency with which a drug is delivered to a site of action

Dataset	Task	Graphs	Nodes	Edges	Features	Classes
PPBR	Regression	1,614	28.8	91.7	39	1
Caco-2	Regression	906	29.3	92.4	39	1
MUTAG	Classification	188	17.9	39.6	7	2
PROTEINS	Classification	1,113	39.1	145.6	3	2
ENZYMES	Classification	600	32.6	124.3	3	6

Table 5.1. Molecular property prediction datasets used in our experiments. The Nodes and Edges columns reflect the average number of nodes and edges per graph.

and is a measure of *distribution*, or how a drug moves between the various tissues of the body.

Caco-2 [111] labels drug molecules by the rate of the drug passing through Caco-2 cells (a human colon epithelial cancer cell line) and approximates the rate at which the drug permeates through intestinal tissue. Fundamentally, this expresses a measure of *absorption* of a drug molecule.

MUTAG [24] is a widely used chemical dataset in which each molecule is labeled as either mutagenic or non-mutagenic, representing whether the compound has a mutagenic effect on the germ cells of the bacterium Salmonella typhimurium.

PROTEINS [16] is a dataset of protein graphs labeled as either enzymes or non-enzymes. In contrast to the other datasets, nodes in PROTEINS represent amino acids and two nodes are connected (again via undirected edges) if the amino acids are less than 6 Angstroms apart.

ENZYMESs [16] Like the proteins dataset, the ENZYMES dataset consists of protein structures. Unlike the other tasks, the ENZYMES dataset is a multiclass classification problem. Each protein belongs to one of six enzyme classes based on the chemical reactions they catalyze. Given its complexity and real-world relevance, the ENZYMES dataset is often used to evaluate the performance of graph-based algorithms, particularly GNNs. The multiclass nature of the classification task also makes it a useful dataset for testing the ability of the TopoPool layer to handle more complex tasks.

5.4.1. Model Configuration

While the TopoPool layer is capable of hierarchical pooling, we demonstrate it here as a single pooling layer following graph convolution layers. We do this for simplicity's sake as adding another pooling layer added complexity without materially improving performance. The base model, as well as all training and hyperparameters, are identical in all our experiments, with only the final pooling layer being swapped out. The GNN layers in our model are based on the GCN architecture [**65**], as it is widely used, efficient and performed well in our experiments. However, we note that the TopoPool layer can be applied in conjunction with any GNN aggregation layer and additionally report unoptimized performance results with GAT [**109**] and GraphSAGE [**45**] as the backbone in Table 5.2. Our model architecture is straightforward, with three graph convolution layers, a single graph pooling layer, and three linear readout layers to map the learned representation onto the final prediction. As it is well known that incorporating global information helps in pharmacokinetic prediction models [**28**] we concatenate node features in each layer with an average pool over the batch dimension. We found that the addition of LayerNorm [**7**] after each convolutional layer helped to stabilize training by stabilizing the gradients flowing to the linear layer $f_\theta(\cdot)$. After each convolutional layer, we concatenate batch-wise max and average pools over the node features. These concatenated features act like skip connections and are summed together to form the input to the final readout layers.

Training is performed with the Adam Optimizer [**62**], learning rate 5e-3, learning rate decay, and patience of 200 epochs. We use a batch size of 128 for all experiments and randomly shuffle the training and validation sets. Each model configuration was tested over ten independent trials and we report both the average and standard deviation of the performance over these trials in Table 5.2. All training and model details are available in our GitHub*.

5.4.2. Benchmark Methods

Graph Pooling methods can be broadly categorized into global or hierarchical pooling. Global methods produce graph-level features by aggregating node-level, and occasionally edge-level, features across the entire graph, usually via a sum, mean, or max aggregation. On the other hand, hierarchical pooling methods implement some version of the SRC framework [**42**], reducing a graph to one with fewer nodes and edges rather than collapsing it into a single graph-level feature and discarding the edges.

To date, all graph pooling algorithms, including learnable algorithms, depend on some heuristics to specify the size of the clusters, number of clusters, or cluster thresholds [**68, 33, 116, 25, 42**]. Unfortunately, this is antithetical to need to locate clusters of unknown size and shape, such as those corresponding to pharmacophores in a drug molecule. In our experiments, we compare TopoPool against a range of heuristic and learnable graph pooling operators outlined below.

*https://github.com/MattsonThieme/TopoPool

No Pool refers to a simple GNN model in which the final graph-level representation is obtained with a simple max pool over all the learned node representations.

TopK [**33**] The authors propose a graph pooling method based on the U-Net architecture [**89**], a popular model in the field of semantic segmentation. The method constructs hierarchical graphs and captures the corresponding nodes' representations at different levels. Downsampling is performed by the TopK pooling layer, which drops nodes based on a learnable projection score and a hyperparameter that specifies the ratio of nodes to retain.

SAG [**68**] Similar to TopK, the SAG pooling layer identifies the most informative nodes by assigning an importance score to each node in a graph using self-attention, then selecting the nodes with the highest scores to form a coarsened graph. Here again, the value of k is a hyperparameter that must be specified a priori.

DiffPool [**116**] is a differentiable graph coarsening method that performs a soft cluster assignment of the nodes to a predetermined number of clusters. While DiffPool is capable of using only a subset of those predetermined number of nodes, the upper limit must still be specified a priori. Additionally, as there are no architectural properties enforcing the notion that nearby nodes be clustered together, they also implement a supplementary link-prediction objective to encourage this behavior.

EdgePool [**25**] introduces a sparse, learnable pooling method for graphs based on edge contraction. They first compute a score for each edge, then sort all edges by their score, successively choosing the edges with the highest scores whose two nodes have not yet been part of a contracted edge. While effective, the authors note the disadvantage

Base	Pooling	Dataset				
		ENZYMES(↑)	PROTEINS(↑)	MUTAG(↑)	Caco2(↓)	PPBR(↓)
GCN	Global	0.410(0.094)	0.727(0.055)	0.732(0.095)	0.444(0.083)	8.92(0.33)
	TopK	0.395(0.081)	0.746(0.029)	0.689(0.128)	0.407(0.097)	8.88(0.32)
	SAG	0.405(0.059)	0.727(0.032)	0.705(0.100)	0.412(0.070)	8.85(0.31)
	DiffPool	0.410(0.079)	0.737(0.023)	0.705(0.155)	0.389(0.044)	8.86(0.53)
	EdgePool	0.403(0.091)	**0.762(0.039)**	0.768(0.042)	0.395(0.061)	8.78(0.35)
	ASAPool	0.392(0.052)	0.750(0.037)	0.774(0.094)	0.455(0.139)	8.78(0.61)
	TopoPool	**0.427(0.080)**	0.759(0.024)	**0.779(0.131)**	**0.382(0.046)**	**8.63(0.43)**
GAT	***TopoPool***	*0.433(0.032)*	*0.771(0.037)*	*0.753(0.120)*	*0.450(0.055)*	*8.70(0.34)*
SAGE	***TopoPool***	*0.438(0.051)*	*0.747(0.041)*	*0.811(0.120)*	*0.376(0.031)*	*8.96(0.21)*

Table 5.2. Performance on the held out test set, best is shown in bold and second best is underlined. For ENZYMES, PROTEINS and MUTAG, we report the classification accuracy on the test set, and for Caco2 and PPBR we report the MAE. Reported values reflect the average and standard deviation in the performance over 10 independent training runs. Performance results with GAT and GraphSAGE as the GNN backbone are shown in italics as hyperparameters for these model configurations were not optimized for the given endpoints. We present them to show that TopoPool can be used with arbitrary GNN layers, as well as to demonstrate the reasonable performance TopoPool can achieve even before hyperparameter optimization.

that each step roughly halves the number of nodes in the graph, and that this cannot be modified by the user.

ASAPool [88] introduces a novel GNN-based attention network to generate importance scores for each node in a given graph. Similar to DiffPool, it then learns a soft cluster assignment for nodes at each layer and uses these assignments to locate and pool subgraphs. However, while this addresses some issues in the graph pooling literature, it still samples substructures using a variant of top-k selection, where the pooling ratio k must specified a priori.

5.5. Results and Discussion

Table 5.2 contains performance results for the benchmark methods and TopoPool on our datasets. Values for GAT/SAGE + TopoPool are shown in italics as these were not optimized for the given configuration. We present them simply to demonstrate the capacity to implement TopoPool with any GNN layer, as well as the ability for TopoPool to achieve reasonable performance even before hyperparameter optimization.

To answer the questions set forth in Section 5.4:

A1 - Given the performance in Table 5.2, TopoPool is very competitive with the existing learnable and heuristic graph pooling algorithms, achieving superior performance on all but one dataset.

A2 - TopoPool can be easily integrated into existing GNN pipelines, and we report good performance using a GAT and GraphSAGE backbone even before hyperparameter optimization in these configurations.

A3 - As we discuss below and show in Figure 5.2, the learned clusters are highly interpretable while also being consistent across examples.

Performance: On all but one dataset, TopoPool achieves superior performance to every benchmark method, and on PROTEINS it achieves only a slightly lower accuracy than EdgePool. Additionally, standard deviations in the performance are in line with all the other methods. Also worth noting is that methods like TopK, which retain only the K highest ranked nodes, often lead to even poorer performance than global pooling, implying that there is utility in retaining information from every node.

Training dynamics: Like all learnable graph pooling methods, the exact pools extracted will depend on the particular initialization. However, we do observe in Figure

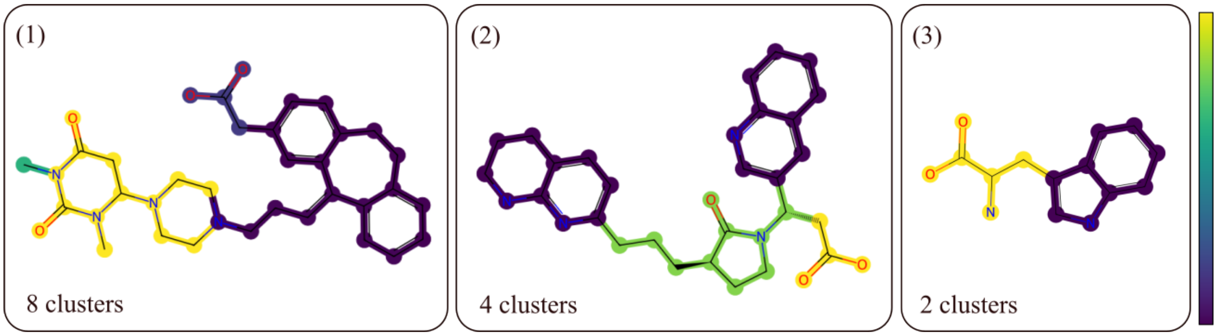

Figure 5.2. Example clusters learned on the Caco2 Permeability endpoint. Each cluster is colored according to the score s_k on its peak node v_k, which reflects the relative importance of each cluster to the given endpoint. We note the clearly delineated clusters that cleanly subsume discrete molecular structures, as well as the consistency of the pools across molecules.

5.2 that the *relative* ranking of the substructures, for example, the carboxyl group (the trigonal planar functional group colored yellow in (2) and (3)) vs. the aromatic rings (the carbon rings colored purple in all three molecules) is preserved across most of the independent trials.

In regards to training time, our implementation uses advanced indexing in PyTorch for all but the *expand and descend* operation, resulting in training times that are comparable with those of DiffPool and EdgePool. The number of clusters learned is also consistent across independent trials, with the model gradually converging on a similar number as training progresses.

Cluster Interpretability: We can see that clusters are clearly delineated, subsume discrete molecular structures, and similar structures are scored similarly across molecules. We note that the two aromatic rings on either side of the molecule in (2), while scored and colored similarly, are distinct clusters. Also of note is that the relative importance of the carboxyl group is conserved across (most, not all) molecules in the Caco2 Permeability dataset. The relative differences in the scores of similar structures across examples is not necessarily of concern, as these scores are relative *within* a given molecule, and the structural context for each molecule will be different.

Consistency across examples: As we can see in Figure 5.2, the aromatic rings are, for the most part, scored lower than the substructures containing peripheral oxygens. This is consistent across examples in the test set for this endpoint, and we observe similar behaviors across Caco2 and PPBR (the two molecular datasets).

Areas for improvement: Learning a variable number of nodes of variable size presents challenging normalization problems, particularly as it relates to how the node representations within a given cluster are aggregated. We tried a number of methods with varying success across datasets. We arrived at the MAX aggregation, as noted in Equation (1), as it achieved good average results across our datasets. However, we note that this is a design choice that can and should be adjusted based on the unique demands of each dataset. Additionally, when considering molecules, there are additional symmetries that the algorithm may be able to exploit. As it stands, the TopoPool algorithm originates a unique cluster from each local maxima in the node scores s. However, due to common symmetries in many molecules, this often results in structures with bilateral symmetry being split into two symmetrical pools. It may improve performance to, for example, merge symmetric, adjacent pools into a single pool. Finally, the learned scores are sensitive to initialization. However, the learned clusters appear to be less so. As our model has a readout layer that maps the clustered representations onto a single value, the scores on each learned cluster have to adapt alongside the weights in the readout layers and will remain sensitive to initialization. This is a limitation of the method and a problem worthy of further study.

5.6. Conclusions

Here, we presented TopoPool, the first learnable, hierarchical graph pooling layer capable of coarsening an entire graph without making assumptions about the size or number of clusters. It also innately uncovers connected substructures, improving interpretability while obviating the need for exogenous forcing functions or regularizers to ensure connectedness within the pools. We demonstrated its efficacy on real-world molecular and protein property prediction datasets, where it outperformed existing graph pooling algorithms on all but one. We additionally showed that even un-optimized implementations can achieve decent performance. Finally, we provide an efficient, open-source implementation of the algorithm built with PyTorch Geometric [**30**]. Given its differentiability, novelty and efficacy, we believe that TopoPool represents a useful addition to the GNN toolkit and a step towards improving molecular and protein representations.

In the next chapter, we introduce a novel method for transforming pre-trained GNN-based molecular property predictors into molecular generators.

6 | Graph Generation via Adaptation

In drug discovery, lead optimization is the process of making slight changes to the structure of a candidate molecule (the lead compound) in order to improve its properties[*]. However, as a compound's structure is what determines its function, every structural change affects every property. A structural change that improves one property may harm another, and it isn't clear a priori which changes will produce the desired result. At present, this problem is approached with a combination of chemical intuition, expert knowledge, and a suite of virtual screening tools operating on cheminformatics libraries that can swap functional groups, atoms or bonds where physically plausible, exploring nearby compounds in chemical space. However, even with differentiable (neural) property predictors, because the enumeration procedure is not differentiable, optimizing for multiple properties in this model requires a guess-and-check approach. In this work, we take the first step to making this task end-to-end differentiable, transforming molecular property predictors into molecular manipulators. Specifically, we focus on a constrained variant of bioisosteric replacement, where we simply swap atom types within a given, fixed molecular structure. By absorbing the vectorized atomic features of a given input compound into a property

[*] I would like to thank Kevin Cusack for the productive, early discussions at AbbVie's MedChem Summit that set this project in motion.

predictor, we can optimize for them directly and locate new compounds with better properties and the same structure. Challenges remain, but we demonstrate that the method is capable of differentiably optimizing compounds towards multiple desired properties while preserving the original structure, a first in the bioisosteric transformation literature.

6.1. Methodology

Our goal is simple: to optimize a given lead compound towards multiple endpoints (properties) without changing its structure. Given this strong structural constraint, the task amounts to finding the optimal atomic distribution within that structure. We solve this problem using a novel methodology that allows us to transform existing molecular property predictors into graph generators without the need for retraining of any kind. This is a simple method for building a generative model from a predictive one.

6.2. Differentiable Property Predictors

These days, the best molecular property predictors are end-to-end differentiable. Our approach allows us to leverage them to intelligently navigate chemical space using the best guides available.

For maximum generality, we assume we already have a pre-trained molecular property predictor f_θ, parameterized by parameters θ, that maps molecular graphs onto a set of k properties[†]. Ordinarily, f_θ is a graph neural network, but it doesn't necessarily need to be. The only requirement on f_θ is that it be end-to-end differentiable and perform graph-level prediction, i.e., it maps entire graphs onto graph-level properties.

[†] In our experiments, this was a property predictor developed at AbbVie, but our methodology applies equally using any open source model so long as it is end-to-end differentiable.

We denote the graph corresponding to a single molecule with $G(V, \mathcal{E}) : V \in \mathbb{R}^{N \times d}$, where N is the number of atoms in the molecule, d the number of features per atom, and $\mathcal{E}$ the set of bonds joining adjacent atoms. The predictor f_θ takes this graph as input and maps it onto each of k molecular properties as follows:

$$f_\theta\big(G(V, \mathcal{E})\big) = y, \;\; y \in \mathbb{R}^{k \times 1} \tag{6.1}$$

6.3. Neural Atomic Replacement

Here, we introduce our methodology, termed Neural Atomic Replacement (NAR). Bioisosteric Replacement is simply the task of swapping functional groups or, in our case, atoms in a molecule, with the goal of improving some molecular properties. Our method is the first learnable, differentiable solution to this atom-swapping problem.

To use this trained model to intelligently modify an input graph, we are going to recast the input features $G(V, \mathcal{E})$ - specifically the node features $V \in \mathbb{R}^{N \times d}$ - as *parameters* of a new, composite model containing both the trained model f_θ and the molecular graph $G(V, \mathcal{E})$. We denote our composite model, $h_{f_\theta, V, \mathcal{E}}(\cdot)$. The composite model h is parameterized by both the trained parameters θ *and* the features of an input compound V. The output of h then becomes:

$$h_{f_\theta, V, \mathcal{E}}(\cdot) = f_\theta\big(G(V, \mathcal{E})\big) = y, y \in \mathbb{R}^{k \times 1} \tag{6.2}$$

Note that Unlike f, h does not take any input (it is implicitly encoded into the model) but still produces a vector of predicted property values $y \in \mathbb{R}^{k \times 1}$.

6.3.1. Optimization

Now that we've built our composite model, we can use the standard machinery of backpropagation to optimize the input graph G.

To do so, we need targets. However, these targets won't be the standard labels provided in $\hat{y}$, but rather the *desired* values for each property. When designing a drug compound, chemists often have a desired property profile in mind. In order for drugs to be effective, they first need to be absorbed, distributed, metabolized, and excreted (ADME) at certain rates. These ADME properties are the properties that the predictor f_θ yields. When optimizing the input graph using the composite model, we denote the desired values for each property with $\hat{y}_{opt}$. From here, updating the features of the input graph becomes trivial: we perform a single forward step with h to generate predicted values y, calculate the loss between y and the *desired* values $\hat{y}_{opt}$ using a loss function $\mathcal{L}(\cdot)$. Then we calculate the gradients $\frac{\partial \mathcal{L}}{\partial V}$ which are used to update only the parameters V in $h_{f_\theta,V,\mathcal{E}}$.

This solves the problem we're after: it modifies the qualities of the *input graph* (drug molecule) in such a way as to push the *predicted* property values towards each of the desired values.

However, given the physical constraints of organic chemistry, this process alone is not sufficient to yield a usable generative model. We must also respect the space of viable atoms, i.e., which atoms are usable in an organic drug compound while maintaining the differentiability of the model.

6.3.2. Fragmenting Atomic Space

One detail we have not mentioned yet is what each of the node features in V represents. In the case of molecular graphs, we make use of the widely used RDKit framework to preprocess the input molecules and generate their graphical representation. For each atom in the molecule, nine features are generated; these features are:

(1) Atomic Number: Identifies the element of the atom.

(2) Chirality: Describes the three-dimensional arrangement of the atom, particularly in stereochemistry.

(3) Degree: The number of explicit bonds the atom has with other atoms.

(4) Formal Charge: The electric charge of the atom.

(5) Hybridization: The type of orbital hybridization (e.g., sp, sp2, sp3).

(6) Implicit Valence: The number of implicit hydrogen atoms or other single-bonded atoms connected to the atom.

(7) Isotope: The specific isotope of the element (if applicable).

(8) Number of Radical Electrons: Electrons that are not paired.

(9) Aromaticity: Whether the atom is part of an aromatic system.

The feature we are most concerned with is the first, atomic number. This defines the type of atom at each position and is the only feature that we update in the optimization process. This is a limitation and will be discussed in a later section but, as our results demonstrate, even changing only this value is sufficient to generate novel compounds with properties closer to the desired properties set forth in $\hat{y}_{opt}$.

The challenge is as follows: using the backprop machinery described above, we will iteratively make small changes to the values in V, gradually shifting the compound in ‘atom-type space’ towards a compound with more desirable properties. For example, if an atom starts out as a Carbon but a Nitrogen there would yield better properties, we would gradually shift that value from 6.0 to 7.0, the atomic numbers for Carbon and Nitrogen, respectively.

However, atoms can only have one integer atomic number or another, not something in between. So, we need a way to bridge the gap between atom types in feature space without quantizing and breaking the differentiability. To do so, we introduce the concept of feature-space fragmentation.

As an additional note, only some atom types are usable in organic chemistry. From all the *possible* atom types, we select only C, N, O, F, S, and Cl, or atomic numbers 6, 7, 8, 9, 16, and 17[‡]. To span the gulf between the first cluster of atom types 6 - 16 and the second 16 - 17, while maintaining differentiability and allowing backpropagation to effectively optimize this feature, we introduce the atomic-space fragmentation scheme in Figure 6.1.

First, we don’t allow the features in V to take on values between these two blocks, shown by the “eliminate gaps” transition in Figure 6.1. Second, we localize the possible values around each of the allowed atom types, shown in the “localize atoms” transition.

[‡] This is mostly due to synthetic constraints.

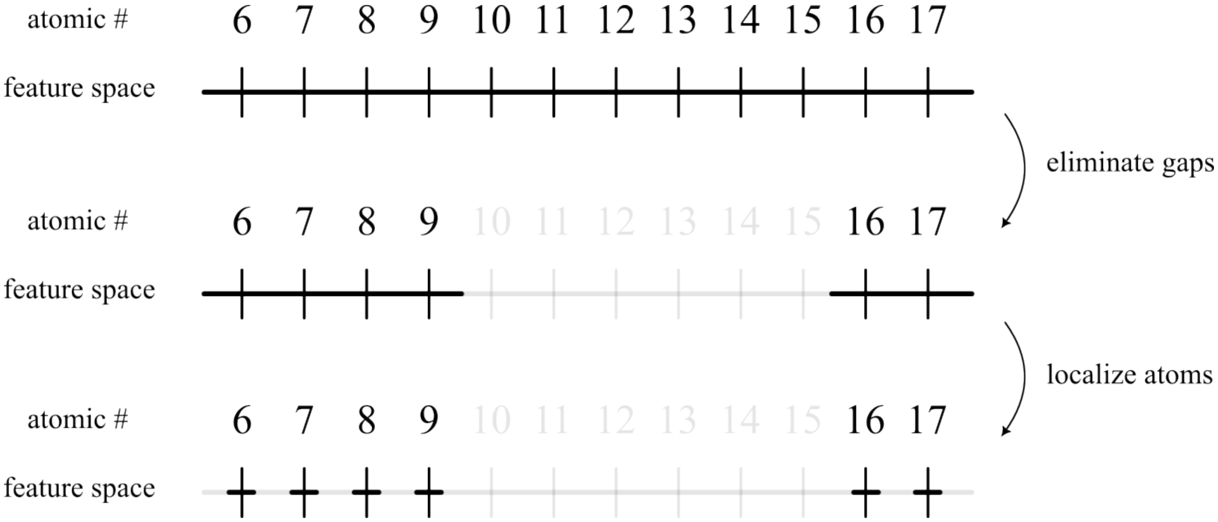

Figure 6.1. The atomic-space fragmentation scheme visualized. At the top, we show the entire space spanning both clusters of allowable atoms. Our first step is to eliminate the space between the two clusters. Then, to further improve the efficiency and performance of the generative scheme, we also localize the allowable feature space around each atom. For example, if we provide an allowable window of width 0.5, the first two allowable windows would be [5.75, 6.25] and [6.75, 7.25]. This window size is a hyperparameter and may vary based on the application.

The width of each of these windows is a hyperparameter, but we have found that a width of 0.5 centered at each atom type yields good performance.

Now, when we are optimizing the atomic numbers in V using the method defined above, we will only allow those features to take on values in the black regions in Figure 6.1, which brings us to our next methodological development: Feature Value Snapping.

6.3.3. Feature Value Snapping

Feature values in V are iteratively modified using the standard machinery of backpropagation. If such a feature value falls into one of the disallowed regions (shown in gray in Figure 6.1, we perform what we term *Feature Value Snapping.* Assume, for example, that we have two windows 1) [5.75, 6.25] corresponding to Carbon, and 2) [6.75, 7.25] corresponding to Nitrogen, and that a feature value in V was adjusted from 6.25 to 6.35 using a learning rate of 0.1. Instead of keeping the value 6.35, which is not in an allowed region, we would snap it up to the bottom of the next window by updating its value to 6.75, placing it into the allowable window for atomic number 7. This procedure accelerates the optimization process by allowing the model to spend less time in the spaces between atom types, and we also find that it encourages the model to learn more optimal solutions.

6.4. Experiments and Results

In our experiments, we set out to answer the following questions:

Q1 - Can our method generate novel and valid compounds with identical structures to the seed compound?

Q2 - If so, do those compounds have more desirable properties than their seed compound? As defined by the targets we set.

Q3 - Further, can our method locate compounds that improve upon multiple competing properties simultaneously?

To find out, we've gathered a random sample of ∼1,000 compounds from the Therapeutic Data Commons [**50**]. For each of these seed compounds, we generate a *single* new compound using our method. This new compound has an identical structure, (possibly) different atoms at each position, and is optimized towards one, two, three, or four ADME properties simultaneously. We've chosen one assay from each of the four ADME categories - Absorption, Distribution, Metabolism, and Excretion - to make the experiments as representative as possible.

Joint Targets	A	D	M	E
	Caco2	VDSS	Fu Mic	Clearance
A	+78.2%	-	-	-
A&D	+70.6%	+23.6%	-	-
A&D&M	+71.2%	+20.2%	+54.5%	-
A&D&M&E	-16.3%	+57.1%	+6.81%	+70.1%

Table 6.1. Median improvements to each of five endpoints (graph properties) our method was able to achieve for property A by itself, A and B, A and B and C, etc. A value of +10% means that the predicted property value of the generated compound was 10% closer to the desired value than the predicted property value of the seed compound. Higher is better, and the uniformly positive performance shows that our method is able to robustly generate new compounds with identical structures but better properties than their seed compounds. Further, rows 2-4 show settings in which we are attempting to optimize for multiple properties simultaneously. This table shows that our method is capable of generating new compounds that improve upon multiple competing properties at once.

Our results are shown in Table 6.1, which presents the performance of the modified compounds when optimizing toward one, two, three, or four of the ADME endpoints. We report the median performance to avoid results being skewed by outliers, which are common in this space. Results are expressed in terms of *improvement* in the predicted property value. For example, a value of +10% means that the predicted property value of the generated compound was 10% closer to the desired value than the predicted property value of the seed compound. This is what we are aiming for and what we achieve.

To answer the questions set forth above:

A1 - Yes. our method generated a novel and valid compound for each of the seed compounds tested.

A2 - Yes. Predicted property values for the generated compounds uniformly improved upon the seed compounds. The only scenario that saw performance degradation was when optimizing for all five endpoints simultaneously, where the Absorption metric decreased. This is due to a well-known competitive interaction between absorption and clearance that we will discuss in Sec. 6.7.

A3 - Yes. Up to three endpoints, our method was capable of improving upon all three simultaneously.

Table 6.1 clearly demonstrates the utility of the method, and we are proud to say that this has been used at AbbVie in the lead optimization phase and generated a compound sufficiently compelling as to be physically synthesized and tested. A significant milestone for any generative method.

6.5. Methodological Details

Given a trained property predictor f_θ, our method has relatively few hyperparameters. They include:

- Learning rate: the size of the changes made to each feature value per iteration
- Number of iterations: how long we run the adaptation/optimization procedure
- Endpoints: the particular properties we're aiming to optimize
- Feature-space fragmentation: how (and if) we decide to fragment the feature space to improve generative efficiency

In our experiments, we choose a learning rate of 0.1. While typically too large a learning rate to train a neural network, we've found that this larger rate allows the method to explore further in feature space and find more optimal solutions.

We choose to optimize for 150 iterations, as this balanced performance with runtime. We note, however, that performance increases the longer we run.

The endpoints we chose were for demonstration purposes only. Individual projects will have their own set of properties they're optimizing for and our method can accommodate these without issue.

In regards to feature space fragmentation, we chose a window of 0.5 because it worked well and was easy to implement. Smaller windows may further improve performance and this is an important direction for future research. There is also the possibility of snapping to the middle of the next window, as opposed to snapping simply to the next edge. We will explore these options in future experiments.

We also note that at the end of the training, modified feature values in V are not integers, but atom types are. As such, we round them to the nearest integer before performing the final inference and assessing the quality of that generated compound.

6.6. Challenges

Many challenges remain, particularly as it relates to incorporating chemical rules into the method. Due to valency constraints, not just any atom can be placed in any location in an atom. At present, we handle this on an atom-by-atom basis, checking the valency of the proposed atom at that position and only allowing it to be swapped into the molecule if that constraint is satisfied. This is inefficient, and a more mature implementation would take this into account in the generation phase.

Atom type is also inherently a categorical value, and our treatment as a continuous value is only to maintain differentiability in this example. Future work could be directed at ways of handling categorical features in a more typical manner.

There is also the challenge of computational efficiency. Similar to diffusion, we need multiple iterations to modify the compound and converge onto a solution. This is an unavoidable property of the method as-is but does not represent a serious detriment because of how the method is intended to be used: to optimize select compounds one at a time. As this is not intended to be used to generate large numbers of compounds for screening, this limitation may be less significant than in diffusion.

Finally, as the property predictor f_θ is fixed, our method is deterministic given a single input molecule. One way around this that we are exploring is to simply add noise to the generation process. Adding noise on a preset schedule could help the model explore more possibilities early on in the optimization while also allowing for the identification of multiple modified compounds for each seed compound.

6.7. Discussion

Beyond just performant, our method is also targetable, meaning that we can trivially target only a given subgraph or functional group by masking the gradients applied to the input features. This is useful in the many cases where chemists know that some part of a compound is important for their project and want to it to remain fixed. In a similar vein, selectively fragmenting the feature space allows us to explore only particular chemical spaces by restricting which atom types we consider. This is useful when accounting for the global demands of organic chemistry or the local demands of particular retrosynthesis capabilities.

The fourth row of Table 6.1 shows clearly the expected competition between absorption and excretion. Generally, if a drug is absorbed more efficiently into the bloodstream, it is available in the body for a longer duration before being excreted. Conversely, if a drug is rapidly excreted, its absorption into the bloodstream may be reduced.

Finally, we are proud to say that, through our collaboration with AbbVie, this method has been used in lead optimization for an active drug discovery project and generated an optimized compound that has been physically synthesized and tested. This is a significant achievement, inducting our method into the small cohort of machine learning approaches that have yielded compounds worthy of synthesis.

Part 3

Accelerating Discovery: ML for High Energy Physics

7 | Introduction to Part 3

Pivoting now from learning on clusters of atoms, or molecules, to *sub*atomic particles and the machines that manipulate them, we review a selection of projects in which we've adapted machine learning tools to inform and control.

7.1. Contributions to Machine Learning for High Energy Physics

In the spirit of treating AI as a tool for accelerating scientific discovery, this chapter highlights select works from our extensive collaboration with Fermilab towards integrating real-time machine learning systems into the accelerator complex. The work revolves around two main projects:

(1) Real-Time Loss De-Blending in the Main Injector and Recycler

(2) Slow Spill Regulation in the Muon-to-Electron (Mu2e) Conversion Experiment

We first cover the two papers published in the Loss De-Blending project:

- K. Hazelwood*, R. Shi*, B.A. Schupbach*, **M. Thieme***, M.R. Austin, M.A. Ibrahim, V.P. Nagaslaev, D.J. Nicklaus, A.L. Saewert, K. Seiya, R.M. Thurman-Keup, N.V. Tran, A. Narayanan, H. Liu, S. Memik, "Real-time Edge AI For Distributed Systems (READS): Progress On Beam Loss De-blending for the Fermilab Main Injector and Recycler," in *Conference IPAC'21.*

*Equal Contribution

- K.J. Hazelwood*, **M. Thieme***, J. Arnold, M.R. Austin, M.A. Ibrahim, V.P. Nagaslaev, A. Narayanan, D.J. Nicklaus, G. Pradhan, A.L. Saewert, B.A. Schupbach, K. Seiya, R.M. Thurman-Keup, N.V. Tran, D. Ulusel, H. Liu, S. Memik, R. Shi, “Semantic Regression for Disentangling Beam Losses in the Fermilab Main Injector and Recycler” in *Conference NAPAC’22.*

And then the three papers published towards the Slow Spill Regulation project:

- A. Narayanan*, J. Jiang*, **M. Thieme***, J. Arnold, M. Austin, J.R. Berlioz, P. Hanlet, K.J. Hazelwood, M.A. Ibrahim, V. P. Nagaslaev, D.J. Nicklaus, G. Pradhan, P.S. Prieto, B.A. Schupbach, K. Seiya A. Saewert, R.M. Thurman-Keup, N.V. Tran, D. Ulusel , H. Liu, S. Memik, R. Shi, "Machine Learning for Slow Spill Regulation in the Fermilab Delivery Ring for Mu2e", in *Conference NAPAC’22.*
- A. Narayanan*, **M. Thieme***, K.J. Hazelwood, M.A. Ibrahim, H. Liu, S. Memik, V. P. Nagaslaev, D.J. Nicklaus, P.S. Prieto, K. Seiya, R. Shi, B.A. Schupbach, R.M. Thurman-Keup, N.V. Tran, "Optimizing Mu2e Spill Regulation System Algorithms", in *Conference IPAC’21, Campinas, Brazil, 2021.*

Additional works not covered in this dissertation include:

- Xu, C., YC. Hu, J., Jiang, J., Memik, S., Shi, R., Shuping, A. **M., Thieme**, M., and Liu, H. Beyond PID controllers: PPO with neuralized PID policy for proton beam intensity control in mu2e. In the *Machine Learning and the Physical Sciences Workshop, NeurIPS 2023.*
- Shi, R., Ogrenci, S., Arnold, J. M., Berlioz, J. R., Hanlet, P., Hazelwood, K. J., Ibrahim, M. A., Liu, H., Nagaslaev, V. P., Narayanan, A., Nicklaus, D. J., Mitrevski, J., Pradhan, G., Saewert, A. L., Schupbach, B. A., Seiya, K., **Thieme, M.**, Thurman-Keup, R. M., and Tran, N. V. Ml-based real-time control at the edge: An approach using hls4ml. In the *31st Reconfigurable Architectures Workshop (RAW) 2024.*

*Equal Contribution

8 | Accelerator Loss De-Blending: A Proof of Concept

The Fermilab Main Injector enclosure houses two accelerators, the Main Injector and Recycler. During normal operation, high-intensity proton beams exist simultaneously in both. The two accelerators share the same beam loss monitors (BLM) and monitoring system. Beam losses in the Main Injector enclosure are monitored for tuning the accelerators and machine protection. Losses are currently attributed to a specific machine based on timing. However, this method alone is insufficient and often inaccurate, resulting in more difficult machine tuning and unnecessary machine downtime. Machine experts can often distinguish the correct source of beam loss. This suggests a machine learning (ML) model may be producible to help de-blend losses between machines. Work is underway as part of the Fermilab Real-time Edge AI for Distributed Systems Project (READS) to develop a ML empowered system that collects streamed BLM data and additional machine readings to infer in real-time, which machine generated beam loss.

8.1. READS Overview

The Real-time Edge AI for Distributed Systems (READS) project is a collaboration between the Fermilab Accelerator Division and Northwestern University. The project has two objectives; first to implement Machine Learning ML into the future Delivery Ring slow spill regulation system [**79**] for the Mu2e experiment [**10, 107**], and second to create a real-time beam loss de-blending system for the Main Injector MI accelerator enclosure also utilizing ML [**93**].

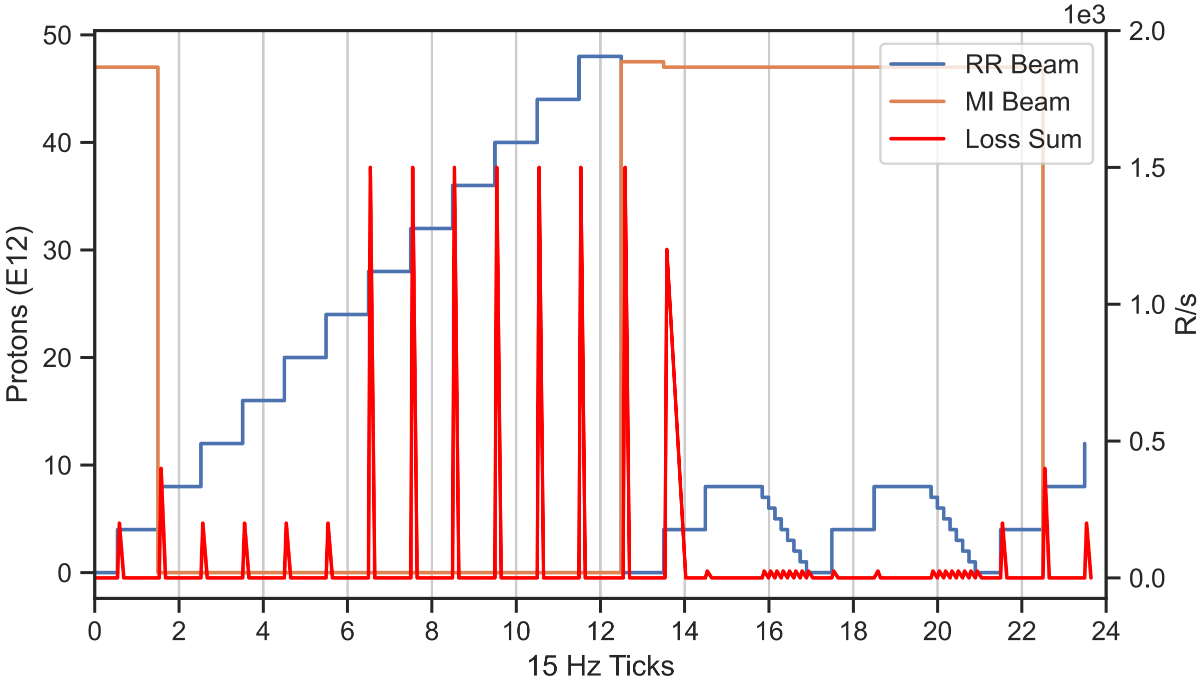

Figure 8.1. Example illustration of overlapping beam events and losses in the MI and RR accelerators.

8.2. Beam Loss De-blending

The Main Injector and Recycler Ring (RR) accelerators share a tunnel and one beam loss monitor (BLM) system. The 8GeV permanent magnet Recycler was originally built as an anti-proton storage ring for the Tevatron collider [**54**]. Anti-proton losses in Recycler were insignificant compared to the 8GeV to 120 GeV proton losses from Main Injector; there was less need to monitor ionization beam losses from Recycler. When the Tevatron was decommissioned, Recycler was re-purposed as a proton stacker for Main Injector 120 GeV NuMI beam operation [**5**] as well as for 8GeV Muon g-2 experiment beam delivery [**41**]. Currently, normal operation of the accelerator complex has high intensity beams in both Main Injector and Recycler simultaneously. Beam losses from both machines are now a large concern. The origin of loss on any of the 259 operational BLMs can be difficult to attribute to any one machine. However, experts can often attribute losses to either Main Injector or Recycler based on time (Fig.8.1), machine state, and location (Fig.8.2).

Using streamed distributed readings and real-time ML inference hardware, this project aims to replicate and improve upon the machine expert's ability to de-blend losses between machines.

8.2.1. Pirate Card Development

In order to satisfy the data requirements for this project, a parasitic VME bus reader card, commonly referred to as a Pirate Card, is being developed and integrated into the existing MI BLM system. Each of the 7 BLM nodes distributed around the 2.2 mile Main Injector complex consists of a VME Crate Processor, Control Card, Timing Card, and an array of digitizers [**12**]. The sole responsibility of the Pirate Cards is to intercept the BLM values of each digitizer throughout the beam cycle without disturbing normal operations of the system. The digitized BLM values will be packaged up with other relevant machine data like beam intensities, cycle events, and MI momentum. This datagram will be sent

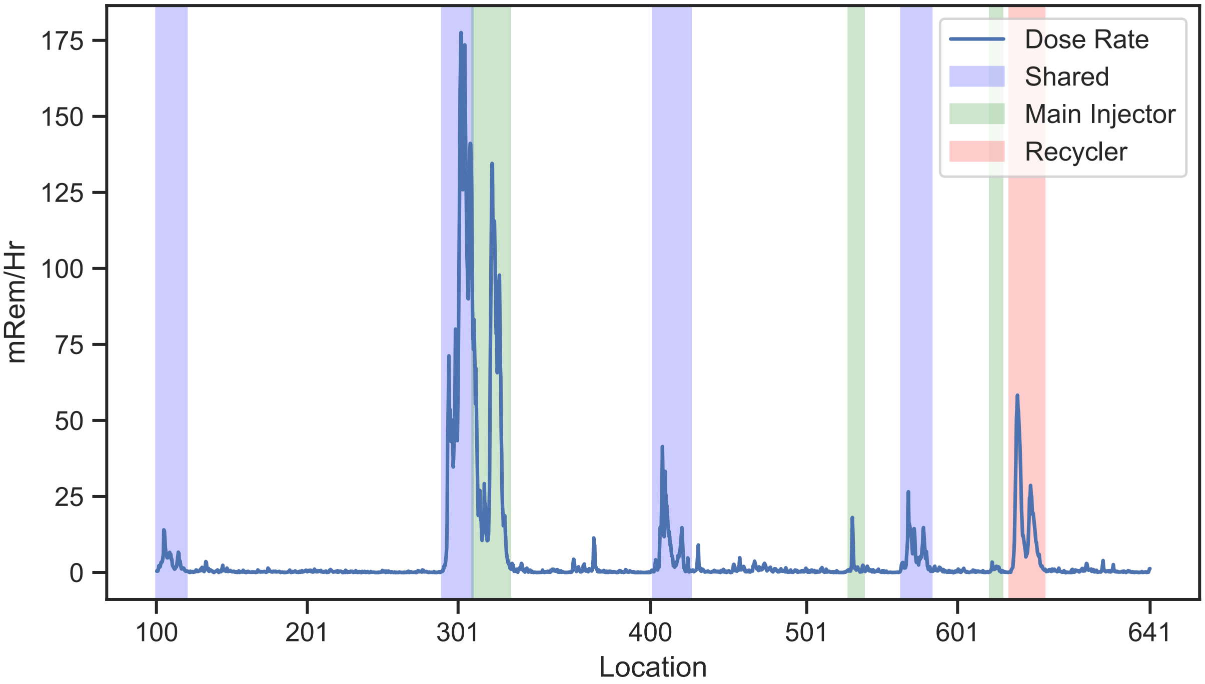

Figure 8.2. Location dependency of MI and RR beam loss as seen from tunnel residual dose rates.

over the network for ML model training data as well as for the eventual FPGA ML model implementation. The Pirate Cards have been designed and are currently being manufactured. Delivery of the cards is expected by late spring 2021; in time to collect higher frequency training data to be used in training a final ML model.

8.3. Datasets

A **Sample Dataset** is being continuously generated using actual accelerator operations data. The data in the Sample Dataset is the same in structure and shape as the data that is expected to be used for training the final ML models, albeit at a much reduced frequency (15Hz) compared to data rates expected from the Pirate Cards. Data rates for the Sample Dataset are constrained by limitations of the existing BLM system and ACNET controls network. An algorithm has been created to label the BLM losses by machine when possible. This data will help inform the extent and structure of data coming from the Pirate Cards. Initial ML models are being designed and trained from this dataset.

The **Training Dataset** will be created using data streamed from the aforementioned Pirate Cards. The expected Training Dataset rate is 333Hz which corresponds to the current rate at which the BLM system nodes poll their digitizers for new BLM sums. Normal operation of the accelerators does not allow for much opportunity to record BLM readings when beam exist in only one machine. For this reason, fairly involved studies have been requested in late spring 2021, just before the Fermilab accelerator complex has it's annual maintenance shutdown. The study will involve manipulating the beam event time line to purposefully keep events for Recycler and Main Injector from overlapping, thus only having beam in one machine at a time. This will allow for the proper attribution and labeling of beam losses to one machine using the same algorithm used for the Sample Dataset. To ensure that a broad range of loss conditions are captured for the Training Dataset, moderate beam losses will be generated at all locations in both machines using various miss-configurations of the machines.

8.4. Model Architecture

The De-Blending Network (DBLN) is comprised of three parts: a BLM network, a State Network, and an Aggregation Network (Fig.8.3). At each point in time, the DBLN maps observations of the last n BLM loss signatures $l_n \in \mathbb{R}^{n\times 259}$ and machine data $e_n \in \mathbb{R}^{n\times 9}$ to class-conditional probabilities over individual accelerators *per BLM*:

$p(a_i|l_n, e_n) \in \mathbb{R}^{259\times 2}$. Note here the overloading of the term "loss": when referring to the BLM losses we use l, and when referring to the mathematical quantity related to the ML model performance we use ℓ.

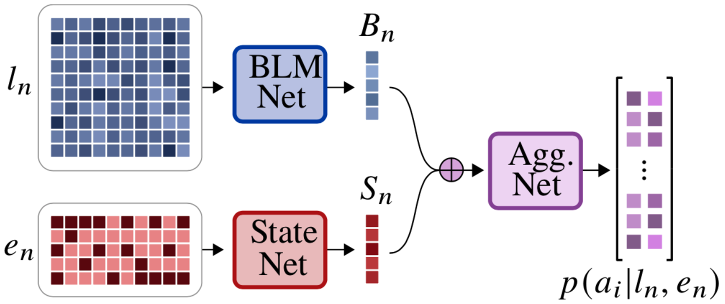

Figure 8.3. DBLN model architecture, a rudimentary model to prove out the concept.

The **BLM Network** is a convolutional neural network (CNN) with two max-pooled convolutional layers followed by two linear layers. The BLM network learns a mapping between the raw BLM loss data $l_n \in \mathbb{R}^{n\times 259} \to B_n \in \mathbb{R}^k$. This vector B_n is then ingested by the Aggregator where it is conditioned on a representation of the machine state generated by the State Network.

The **State Network** is a two-layer fully-connected multilayer perceptron (MLP) that learns a mapping between the last n state observations $e_n \in \mathbb{R}^{n\times 9} \to S_n \in \mathbb{R}^k$. The output S_n serves as a conditioning mechanism for the representation of the BLM loss signature B_n.

The **Aggregator Network** is a three layer fully-connected MLP that learns to map $B_n \oplus S_n \in \mathbb{R}^k$, where $\oplus$ is the elementwise sum, to class-conditional probabilities over $p(a_i|l_n, s_n) \in \mathbb{R}^{259\times 2}$. We choose the elementwise sum instead of concatenation to make the model more compact.

To train the model, we use Binary Cross-Entropy Loss and the Adam Optimizer with a learning rate = 0.001. The final model has 1.3M trainable parameters.

8.5. Preliminary Results

Initial results using the Sample Dataset show promising performance. Figure 8.4 details training accuracy and loss over the first 1000 batches using the past $n = 2$ observations and batch size = 32.

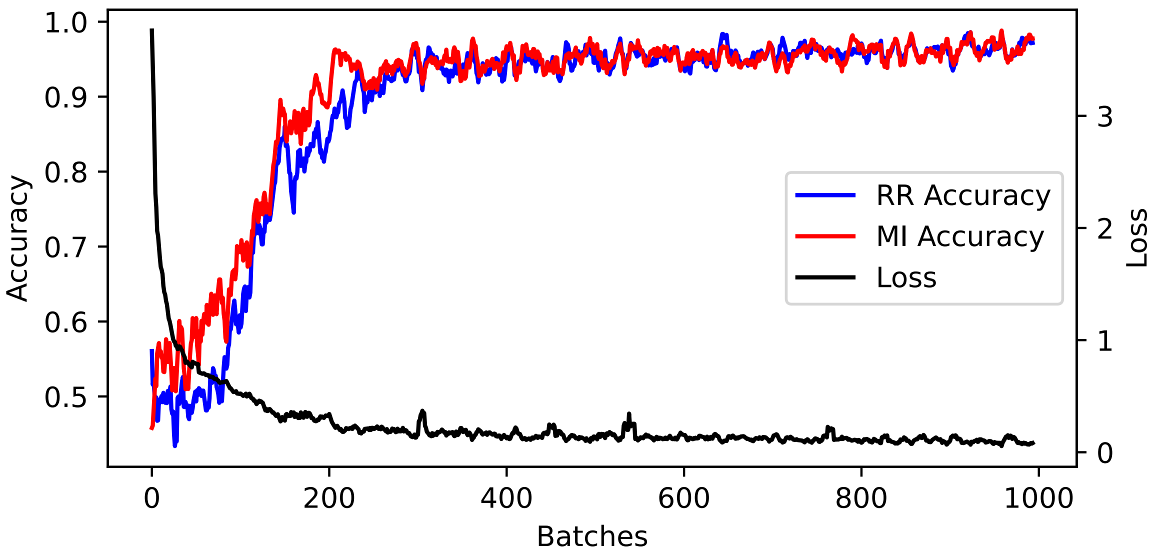

Figure 8.4. RR and MI training accuracy over 1000 batches.

Figure 8.5 (A) shows the measured beam intensities in MI and Recycler over 24, 15Hz ticks. Section (B) shows the normalized BLM measurements at each tick. Sections (C) and (D) show the output of our model scaled by the BLM loss intensity (significance). From these plots, we can see that our model is appropriately classifying the losses in each region. As beam is extracted from Recycler to MI; our model recognizes the change in the loss signature and switches the inferred label from RR to MI in turn.

Overall, high validation accuracies (95% and 96% for MI and RR, respectively) are evidence that our model is learning meaningful mappings between BLM loss profiles and their machine of origin.

8.5.1. Model Confidence

Of particular interest is the model's behavior on the BLM losses that cannot be attributed to a single machine, i.e., loss profiles acquired when MI and RR operate simultaneously. Each row in the output $p(a_i|l_n, s_n) \in \mathbb{R}^{259\times2}$, corresponds to $[p(\mathrm{MI}), p(\mathrm{RR})]$. Probabilities ≥ 0.5 are treated as positive identifications and < 0.5 as negative identifications. Performing inference on these data with unknown labels yields the confusion matrix in Table 8.1, where MI/RR (+) and MI/RR (-) represent positive and negative identifications, respectively.

Table 8.1. Model Confusion Matrix

	MI (+)	**MI (-)**
RR (+)	1%	2%
RR (-)	77%	19%

Presently, the model is disproportionately recognizing these unknown losses as originating from MI and *not* RR. Experimentation is underway to better understand the model's behavior on these data with unknown labels.

8.6. Model Implementation

The ML model will be implemented as an IP core on the Intel Arria-10 SOC. The board contains an FPGA side and a hard processor subsystem (HPS) side, which have

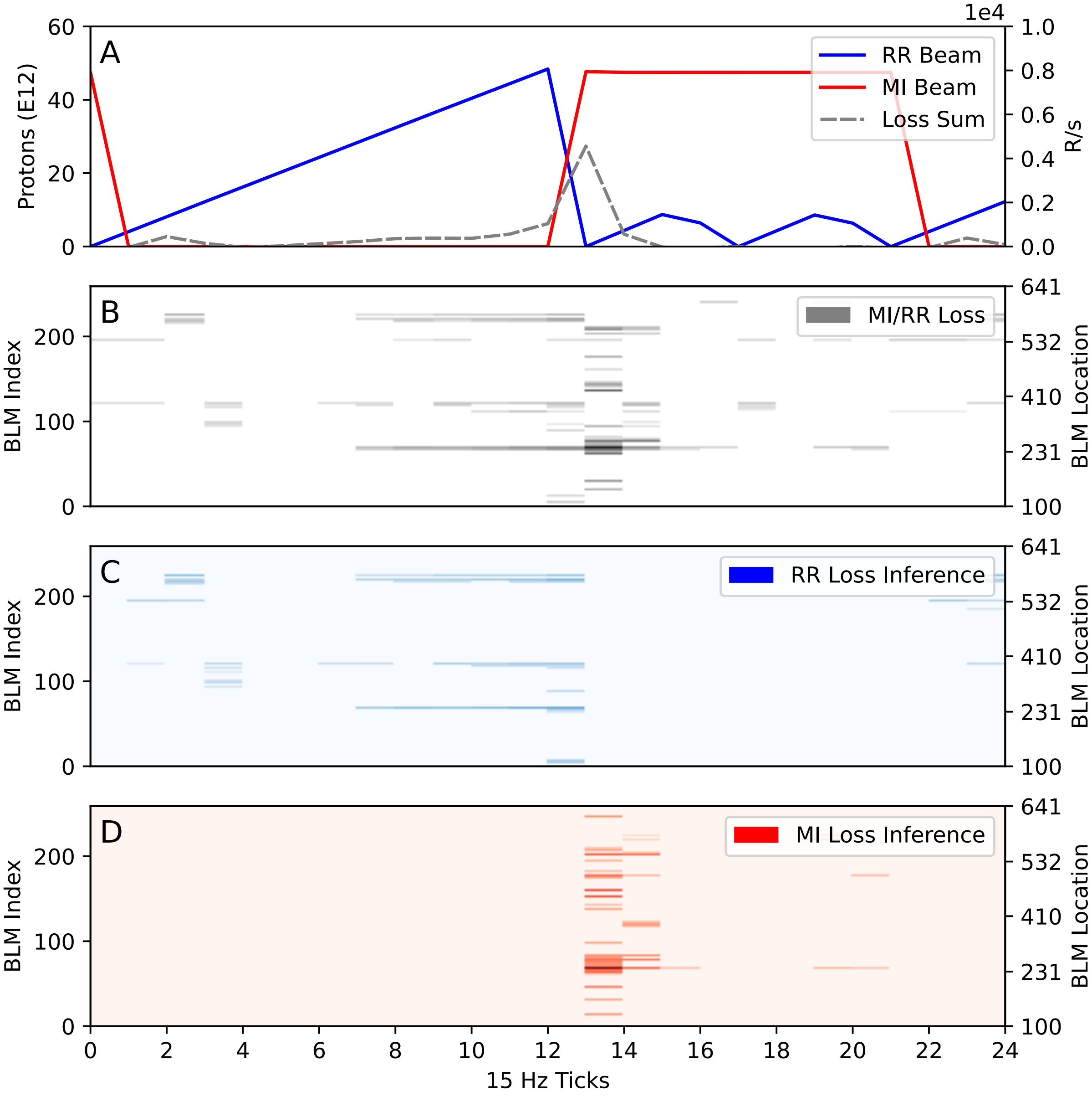

Figure 8.5. Model inference on a single beam extraction from RR to MI using Sample Dataset.

fast communication through HPS-FPGA bridges between them. The FPGA side can be used to implement the DBLN network for processing the data. The HPS side has

Ethernet and can do complicated calculations. The HPS will also be useful for updating the implemented DBLN network by partial reconfiguration.

The hls4ml+Quartus tool flow will be used to generate an initial design for the ML IP. Based on the model development described in previous sections, the saved model will act as an input to hls4ml and will generate an implementation in C++, which uses HLS Compiler for hardware design. The Quartus backend of hls4ml will be used as a starting point, with additional customization done later. Finally, The DBLN network will be implemented as an IP core and connected to HPS using the Platform designer.

During the design phase of the IP, there must be trade-offs between the expected latency and the limited resources. Various methods can be adopted to optimize the implementation by exploiting pipelines and parallelism. For example, unrolling the potential loops, modifying the initial interval and adding registers between each layer can all be used to help achieve the desired data processing pipeline. Multiple parallel kernels will be used, this will likely translate into parallel data paths. To fully utilize the limited resources on the FPGA, with respect to constraints imposed by other functionality that is co-located on it, we need to carefully select a proper precision of the data representation and consider reuse of the implemented kernel and BRAM buffers.

8.7. Summary

The READS Main Injector accelerator enclosure beam loss de-blending project is progressing well. A Sample Dataset has been generated using MI/RR readings and a very promising preliminary ML model has been created from the data. To meet the project's data needs, a custom BLM node VME bus reader card, commonly referred to as a Pirate Card, has been designed and is being manufactured.

9 | Accelerator Loss De-Blending: Efficient and Effective

Once we proved the concept, we set out to refine the method and design a model capable of disentangling loss profiles without using any machine state data.

Fermilab's Main Injector enclosure houses two accelerators: the Main Injector (MI) and the Recycler (RR). In periods of joint operation, when both machines contain high-intensity beam, radiative beam losses from MI and RR overlap on the enclosure's beam loss monitoring (BLM) system, making it difficult to attribute those losses to a single machine. Incorrect diagnoses result in unnecessary downtime that incurs both financial and experimental cost. In this work, we introduce a novel neural approach for automatically disentangling each machine's contributions to those measured losses. Using a continuous adaptation of the popular UNet architecture in conjunction with a novel data augmentation scheme, our model accurately infers the machine of origin on a per-BLM basis in periods of joint and independent operation. Crucially, by extracting beam loss information at varying receptive fields, the method is capable of learning both local and global machine signatures and producing high-quality inferences using only raw BLM loss measurements.

9.1. READS Overview

The Real-time Edge AI for Distributed Systems (READS) project is a collaboration between the Fermilab Accelerator Division and Northwestern University. The project has two main goals: 1) to create a Machine Learning (ML) system for real-time beam loss de-blending in the Main Injector (MI) accelerator enclosure [**93**], and 2) to create a separate ML system for slow spill regulation in the Delivery Ring [**79**] used in the Mu2e experiment [**10, 107**]. In this paper, we extend our previous work [**46**] and introduce a novel approach to beam loss de-blending inspired by semantic segmentation models originally developed for biomedical imaging [**89**].

9.1.1. Beam Loss De-blending

The MI and RR accelerators share a tunnel and one beam loss monitoring (BLM) system. When originally constructed, the 8 GeV permanent magnet Recycler was used as an anti-proton storage ring for the Tevatron collider [**54**]. As the 8 GeV anti-proton losses from RR were relatively insignificant compared with the 120 GeV proton losses from MI, there was little need to monitor ionization beam losses from RR. However, when the Tevatron was decommissioned, RR was re-purposed as a proton stacker for MI 120 GeV NuMI beam operation [**5**] as well as for 8 GeV Muon g-2 experiment beam delivery [**41**]. As a consequence, normal operation of the accelerator complex sees high-intensity beams in both Main Injector and RR simultaneously, and beam losses from both machines are now a significant concern. However, while the origin of radiative losses measured on any of the 259 operational BLMs can be difficult to attribute to a single machine, experts can often attribute losses to either MI or RR based on timing, machine state, and physical location within the ring.

Using streamed, distributed BLM readings and real-time ML inference hardware, this project aims to replicate and then improve upon the machine expert's ability to de-blend or disentangle each machine's contribution to the measured losses.

9.2. Preliminaries

BLMs are spaced (approximately) evenly within the tunnel and report the incident flux in mR/s. Because this flux is generated when beam is lost from the accelerators, i.e. when beam scrapes the edges of the beampipe and generates a spray of particles that then exit the accelerator, we often refer to it as 'loss'. When we discuss the 'BLM loss profile,' we are referring to the pattern of flux measurements over the BLMs at a given time. This is not to be confused with 'loss' in machine learning, which refers to the penalty incurred for prediction errors. In this paper, when we refer to 'loss' or 'BLM loss', it is these flux measurements that we refer to.

9.3. Training on BLM Loss Profiles

Following recent progress in Pirate Card development [**13**], which now allows for the collection of high frequency (333 Hz) data in real-time directly from the BLMs, we have constructed a training dataset using actual accelerator operations data.

A single training example is composed of a single BLM loss profile, also called a 'tick', collected at some time i, and is represented as a 1D vector $x_i \in \mathbb{R}^{259}$ in which each of the 259 elements represents the flux over each of the 259 BLMs at time i. To date, we have collected and processed data from 12M such ticks at 33 Hz, 9M of which are used for training, 1M for validation, and 2M for testing.

We take a supervised learning approach to attributing BLM losses to one machine or another, so in order to generate a training signal, we also need targets for each example. Targets for a given tick are defined by the operational states of MI and RR at that time. Specifically, whether or not each accelerator is carrying beam. For a given tick $x_i \in \mathbb{R}^{259}$ we construct a target vector $y_i \in [0,1]^{259 \times k}$, where the number of classes $k = 2$ and the value of the label reflects whether MI or RR are carrying beam.

When neither machine is in operation, values measured on the BLMs could not have originated in either machine and must be background noise. Labels $y_{i,j}$ for each BLM$_j$ in this setting would be assigned values $[0., 0.]$ as the probabilities that the loss originated in either machine are zero. It is important to include these during training to ensure the model is able to distinguish the signatures of each machine from background noise. Next, in settings in which only a single machine is in operation at once, referred to here as 'single operation', the losses measured on all BLMs could only have originated in that machine. In this case, depending on whether MI or RR are the single machine in operation, the labels for each BLM$_j$ at that tick will be either $[1., 0.]$ or $[0., 1.]$, for MI or RR, respectively.

In the next setting, when both machines are in operation simultaneously, referred to here as 'joint operation', we do not have a clean way to attribute the losses to one machine or the other. Indeed, this is the difficulty motivating the project. Data collected during periods of joint operation are handled in one of two ways: 1) We hold these data out during training and use them only for testing, relying on expert machine operators to assess the quality of the predictions. This is what we will discuss in our Results section. 2) We can construct synthesized data that approximate, to the best of our ability, the relative percentages of the loss originating in MI or RR. In the second case, synthetic data and labels can be created by summing loss profiles from periods of single-operation of each machine (controlled for machine states) and, for each label, normalizing the target probabilities for MI and RR to sum to 1. As training has not been completed on synthetic data, we confine our discussion to results obtained using method 1.

9.4. Semantic Regression

Here, we motivate the architectural choices made in designing our model as well as detail how our semantic *regression* model differs from the popular UNet **[89]** architecture for semantic *segmentation*.

Most generally, for each input example x_i, we are looking for an estimator $f(\cdot)$ parameterized by learnable parameters θ mapping 1D BLM loss profiles onto predicted class probabilities for each BLM:

$$f_\theta : x_i \in \mathbb{R}^{259} \to \hat{y_i} \in [0, 1]^{259 \times k} \tag{9.1}$$

where, for each index j corresponding to BLM$_j$ at tick i, $\hat{y}_{i,j} \in [0, 1]^{1\times 2}$ reflects the probability that the loss measured on BLM$_j$ at tick i originated in either of the $k = 2$ machines (MI or RR). This task is similar to one known as semantic segmentation. The object of semantic segmentation - considered primarily in computer vision applications **[77]** - is to classify each pixel in the input into one of k categories. For example, in biomedical imaging, we may want to know whether some pixel contains normal or abnormal tissue or, in autonomous driving settings, we can imagine the utility of knowing whether some pixel represents road or sidewalk.

The UNet architecture has seen wide-ranging success in such settings. At a high level, it is composed of cascaded convolution operations between two distinct halves. In the first half, known as the contracting path, we apply repeated convolution operations, each time increasing the channel dimension while the original spatial dimensions are downsampled via pooling. In the second half, known as the expanding path, we invert the operations of the first, de-convolving at each layer and decreasing the channel dimension until we arrive at an output with the same spatial dimension as the original input image but with k channels corresponding to k classes. To better incorporate information from varying receptive fields and avoid information loss, feature maps are passed layer-wise from the contracting path to their analog in the expanding path. Finally, softmax is applied over the channel dimension of the final layer, and class labels are obtained.

While similar to the original UNet, our model differs in two important ways: 1) our input $x_i \in \mathbb{R}^{259}$ is 1D, so we replace 2D convolutions with 1D convolutions, and 2) we are not trying to predict a *single class* per BLM, but *probabilities for each class*, each of which may scale independently between 0 and 1, so we replace UNet's final softmax layer with a sigmoid. Losses are calculated using the MSE between predictions and labels: $\ell = \text{MSE}(\hat{y}_i, y_i)$.

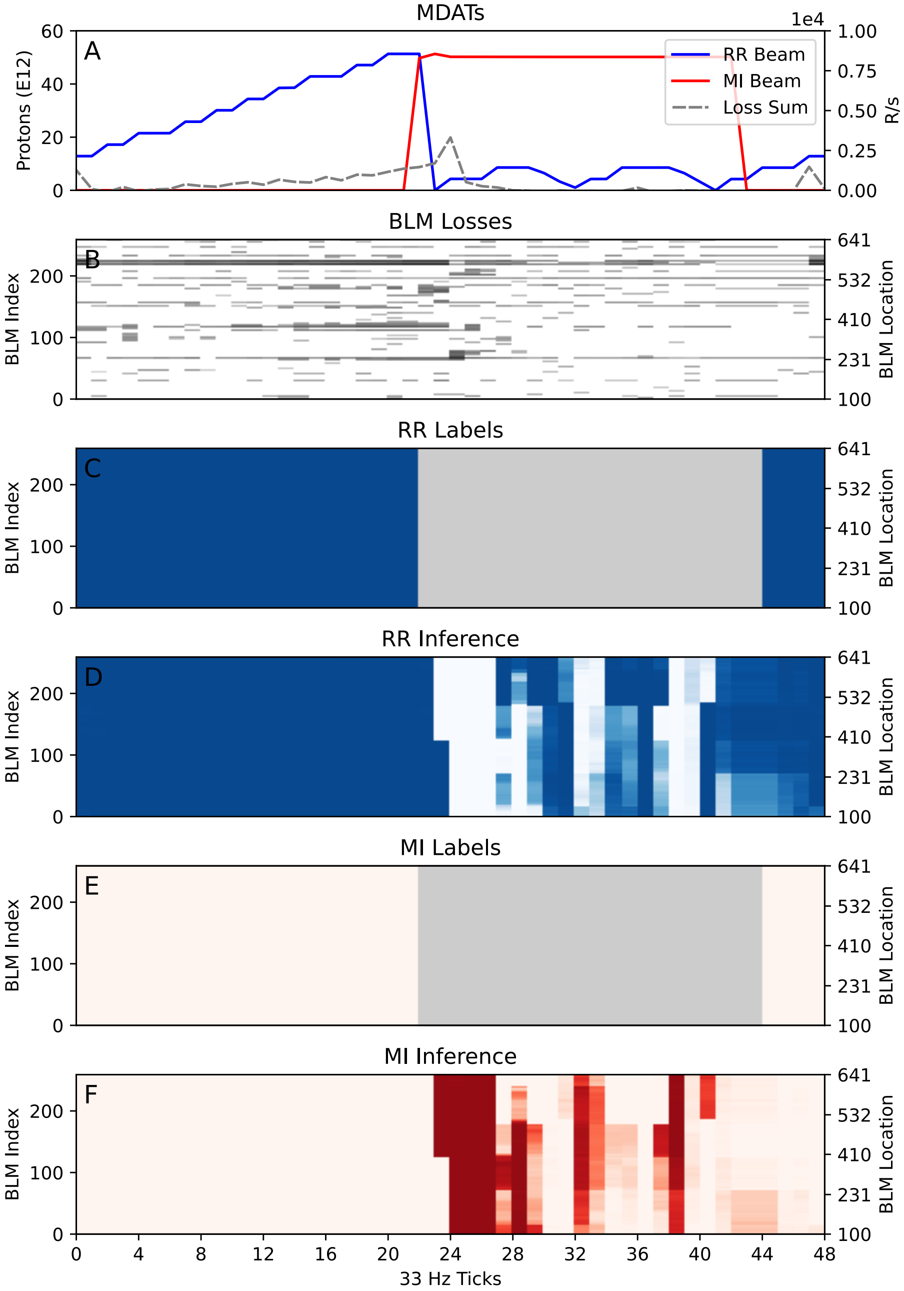

Figure 9.1. Inferences made on BLM losses during a period of joint operation. A: Beam intensities (R/s) in RR and MI over 48 ticks. B: BLM loss profiles - darker = more loss. C and E: Labels for RR and MI, where gray indicates that the machine of origin is unknown. D and F: UNet model inferences for RR and MI, respectively. Intensity corresponds to the inferred probability that the losses on a particular BLM at a particular loss originated in RR or MI.

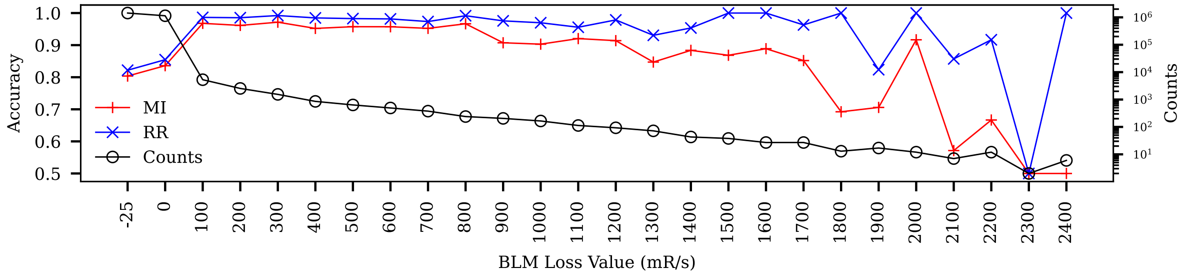

Figure 9.2. Accuracy is high in the primary region of interest, between 10 and 2500 mR/s. Counts reflects the number of observations with loss in that range. Note that noise in the accuracy is driven by the declining counts with a given loss value.

9.5. Results

Figure 9.2 details prediction accuracy per-label in periods of single operation (MI *or* RR *or* neither, i.e. labels [1., 0.], [0., 1.] or [0., 0.]) vs. BLM loss value. We report accuracy here as it is a more interpretable quantity than MSE, using a 20% threshold for the accuracy calculation, i.e. if a prediction is further or closer than 0.2 away from the label, we consider the inference incorrect or correct, respectively. This figure shows that our model is able to learn robust representations of each machine's unique signature using only BLM loss profiles. It also shows that prediction performance is highest on BLM loss values of interest, namely those above the background noise (10 mR/s) and below the existing abort threshold (2500 mR/s). While accuracy appears to fall above 1800 mR/s, this is primarily due to a paucity of data in this range, of which the counts in log-scale are visible on the right axis.

9.5.1. Inference on Losses of Unknown Origin

Of central interest in Figure 9.1 are panels D and F, which contain inferences made during periods of Since some portion of the beam is not captured during the coalescing process, this beam continues around in the machine until the end of the cycle, where it is finally lost at the RR collimators and abort line around BLM index 210. ingests), and C/D displays the labels, where the gray indicates regions in which we cannot assign a unique label. These inferences agree well with known behavior in MI and RR, and we call the readers attention to two features of these inferences that demonstrate the model's ability to generalize beyond the labeled training data.

First, in panel D, the heavy band of RR inferences starting around BLM index 200 are in agreement with known behaviors in RR. The two RR bumps between ticks 24 and 42 are in the beam to the Muon campus and reflect a process that involves coalescing 53 MHz RF bunched beams into larger 2.5 MHz RF bunches. During this process, some beam is lost, and ultimately the beam is extracted around BLM index 190. Since some portion of the beam is not captured during the coalescing process, this beam continues around in the machine until the end of cycle where it is finally lost at the RR collimators and abort line around BLM index 210. This region after BLM index 200 is where we would expect large RR contributions to the loss, and the model agrees across both RR bumps.

Second, in panel F, ticks 32 and 38 happen to lie in the few milliseconds between events where there is no beam in RR. Our model's high confidence that the losses in these ticks originated in MI is likely due to the line interpolation and lower frequency data rate. Our labeling system cannot label this type of event correctly every time on 33 Hz data.

However, it is likely there are samples where it is labeled correctly because the sample happened to occur at an opportune time in which losses and beam events overlapped. What we can gather from panel F is that our model was able to learn these profiles and correctly impute that, given the loss profiles at ticks 32 and 38, the loss likely originated in MI.

9.6. Future Work

Towards the goal of real-time inference, this model is being adapted for implementation on FPGA. We are also exploring additional methods for improving the generalization and accuracy of the model in periods of joint operation, including the generation of representative synthetic data, the incorporation of temporal information, and further statistical analysis of synergistic effects observed during periods of joint operation.

10 | Mu2e: Differentiable Simulators for PID Optimization

Transitioning now to our work on the *Mu2e* project, we discuss the first approach taken to solve the problem: optimizing PID gains using a novel, differentiable physics simulator and the standard ML machinery of gradient descent and backpropagation.

A slow extraction system is being developed for the Fermilab's Delivery Ring to deliver protons to the Mu2e experiment. During the extraction, the beam on target experiences small intensity variations owing to many factors. Various adaptive learning algorithms will be employed for beam regulation to achieve the required spill quality. We discuss here the preliminary results of the slow and fast regulation algorithms validation through the computer simulations before their implementation in the FPGA. Particle tracking with sextupole resonance was used to determine the fine shape of the spill profile. Fast semi-analytical simulation schemes and Machine Learning models were used to optimize the fast regulation loop.

10.1. Resonant Extraction at Delivery Ring

Slow extraction is a well-established technique to deliver continuous beams to the experiments. Nevertheless, the spill quality - or uniformity - remains to be one of the biggest challenges as the beam intensity and complexity of the experiments are growing. Slow extraction in the Fermilab Delivery Ring (DR) [**106**] for the Mu2e experiment [**4**] is achieved by exciting the 3rd integer resonance and by driving (squeezing) the machine tune to the exact resonance value (2/3). The resonance strength is controlled by the two circuits of sextupole magnets, and the squeeze is driven by the dedicated circuit of 3-tune ramping quadrupoles (QX). To satisfy those requirements, the Spill Regulation System (SRS) is being developed.

10.1.1. Spill Regulation System

The SRS will regulate extraction through 2 primary elements: the QX circuit and the RF Knock-Out (RFKO) system. Each regulation element is controlled by separate but concurrent control loops. The SRS has been designed to mitigate several sources of ripple in the spill profile.

The SRS system architecture will consist of the System-On-Module (SoM) and a carrier board. The SoM is a FPGA mezzanine card that hosts the Intel Arria10 SoC. Arria10 SoC features a second-generation dual-core ARM Cortex-A9 MPCore processor-based hard processor system (HPS). One ARM Core will be designated for the front-end software application, which provides an interface between the FPGA controller and the control system. The second ARM core of the HPS can be dedicated to calculating cycle-to-cycle feedforward corrections or facilitating machine learning algorithms.

Furthermore, the SoM uses bottom FMC connectors to mount onto a carrier board, which, in turn, is contained within a rack-wide chassis. The FMC connectors provide two PCIe x8 Gen3 LVDS lanes to interface with the components on the carrier board. The carrier board hosts the peripherals, which will receive clock signals, timing signals, and spill monitoring signals. It will be critical to coordinate the SRS processes within the machine cycle and within each spill interval.

10.1.2. Functional overview

The overarching goal of this work is to develop and study algorithms that would facilitate signal processing in the SRS. There are three main components of the SRS: slow spill profile regulation, fast random ripple regulation and the harmonic ripple content tracker. Here we will discuss the first two of them.

The characteristics of the proton beam in the DR could slowly change with time due to slow drifts in the various accelerator components. The slow regulation controller will be tracking the slow changes in the spill profile producing corrections to the QX current ramp needed to achieve the uniform spill rate. This slow regulation is done adaptively over many spills.

The fast regulation system response, on the other hand, would be supplemented on top of the slow regulation in order to correct instantaneous ripples in the spill intensity. We assume here that this fast noise (ripples) has a random nature or otherwise is a semi-random component of regular harmonic noise that the harmonic controller is not able to suppress. These fast fluctuations can be large. In the SRS, this is handled by the fast PID loop controller.

10.1.3. Slow Regulation Simulation

The algorithms for the slow regulation were tested on a simulation by tracking 132,000 particles in a lattice with a single sextupole. The input for the simulation is the machine's horizontal tune value at every turn. The simulated beam's kinetic energy was 8 GeV (with no momentum dispersion) and a normalized rms emittance of 2.6 π mm-mrad. The time step for the simulation is one turn, and the total time for one iteration is one full spill. For our purposes, we start with a linear tune curve from $\nu_x = 9.650$ to $\nu_x = 9.666$ during the first spill. The simulation tracks the position and angle of each particle at the end of each turn, and any particle whose position is found beyond the electrostatic septum plane is counted as extracted.

For computational purposes, the total number of turns was divided into bins of 250 turns. The number of particles extracted (n_{ext}) is counted bin-wise and is compared with the ideal number of particles that ought to be extracted n_{ideal}. If n_{ext} is greater (or lesser) than n_{ideal}, then the tune distance from resonance for that respective bin is increased (or decreased) by k%. The new modified tune values for each of the bins are then stored and fed as feedforward input to the next spill. This is iteratively repeated until the ideal spill rate (with ±5% error tolerance) is achieved for all the bins. Various $k\%$ values were tested, and the results are discussed in the poster. A typical result is presented in Fig. 10.1. The algorithm was successfully able to find the ideal tune curve within a few hundred iterations. In real life, one iteration amounts to a spill duration of 43 ms, thus the algorithm should correct the spill rate within few minutes.

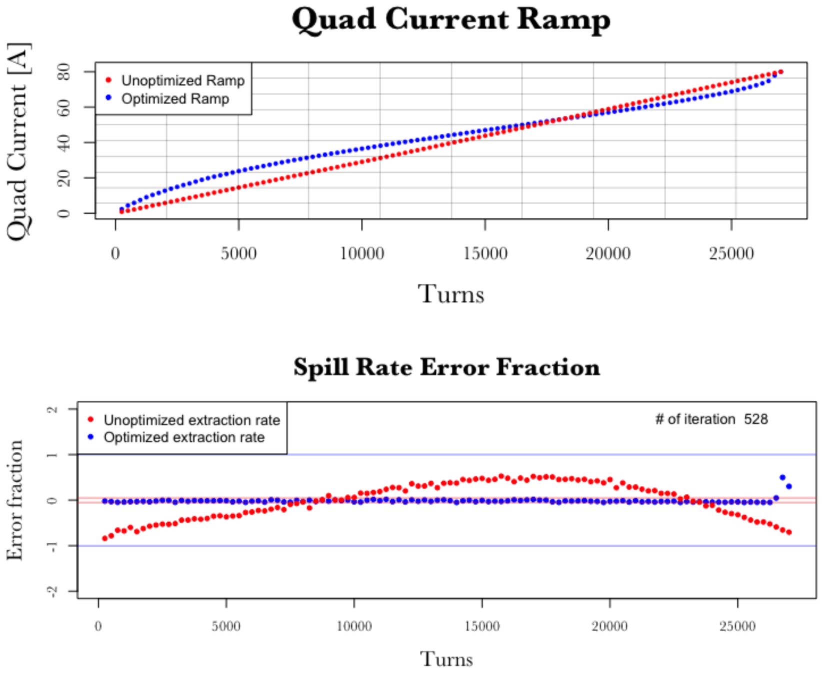

Figure 10.1. Slow regulation algorithm finding the ideal tune ramp for uniform extraction.

10.1.4. Fast Regulation with PID Controller

As noted earlier, the noise source that demands fast regulation is random (or semi-random) and induces variations in the spill rate within one spill. This is met with a PID loop that would send a control signal superimposed on top of the ideal ramp curve in order to curtail the ripples in extraction. The primary knobs for the PID controller are the three gain values, G_p, G_i, and G_d, where the control signal u is given by

$$u = G_p \cdot e(t) + G_i \cdot \int e(t)dt + G_d \cdot \frac{de(t)}{dt} \tag{10.1}$$

where e is the error in spill rate. If the PID controller is ideal, the control signal would be equal in magnitude but opposite in sign to the noise profile present within the spill. However, the tuning of the PID controller can sometimes be tricky as the complete loop includes many machine elements and the beam response. While manual optimization techniques exist, we construct an end-to-end machine learning (ML) differentiable simulator parameterized by the PID gains that allows us to optimize them directly from simulated data. In addition to full-spill optimization, the ML system is capable of finding optimal PID gain values in arbitrary subdomains of the spill.

In order to generate the training data for the ML to tune the PID gains, it would be computationally expensive to carry out particle tracking. Instead, the machine ripples and the regulation response can be very adequately reproduced with a simplified analytical model of extraction.

10.2. Analytical Modelling of Extraction with Fast Regulation

To simulate the fast regulation, we assume that the ideal tune ramp to give the perfect extraction is already in place (because in real life, the slow regulation loop will have already provided that). The analytical model approximates the ideal quad current ramp to be a logarithmic function of time, on top of which the fast regulation performance is simulated. The goal of the physics simulator is to test the performance of the PID loop in terms of spill quality quantified by the spill duty factor (SDF), defined as $1/(1+\sigma^2_{\text{ext. rate}})$. The spill rate in the physics simulation is normalized to an expectation value of 1. The time step of the simulation is done at 10 kHz as the SRS has been designed to have a total Gain-Bandwidth of 10 kHz [**51**].

10.2.1. Analytical Modelling - Fast Regulation Simulation

The ripples in the extraction rate are generated in the physics simulator using a randomwalk log-normal distribution. The PID loop (loaded with the 3 gain values) reads the rippled extraction rate and computes the control signal to be sent to the quads in order to suppress the ripples. Since this PID response is superimposed with the quad current, the semi-analytical model is involved in calculation of the fast extraction rate changes. Also the fast quad modulation is passed through a Butterworth low-pass filter to simulate the steel beam pipe shielding quadrupole's magnetic field variations higher than 1 kHz. When a particle finds itself outside the stable region in phase-space, it will take a finite time to transit to the septum to get extracted. The nature and extent of the delay were studied in earlier simulations, and an appropriate time delay transit function was constructed. The low-pass filtered PID response is convolved with this transit time delay function. The delayed and low-pass filtered control signal is then superimposed with the idealized logarithmic current ramp, and the total number of particles extracted at every time step is computed along with the extraction rate, which is the output of the physics simulator. The ML simulator uses this output to find optimal gain values for the PID loop to maximize performance of the fast regulation loop.

10.3. Learning with a Differentiable Simulator

PID gain values determine how well the PID is able to control the extraction rate. While manual optimization techniques exist, we construct an end-to-end differentiable simulator parameterized by the PID gains that allow us to optimize them directly from simulated data. In addition to full-spill optimization, the system is capable of finding optimal PID gain values in arbitrary subdomains of the spill.

10.3.1. A Hybrid Machine Learning Simulator

At the base of our ML simulator is a neural network that maps a constant scalar input $\in \mathbb{R}^1$ to a vector of PID gains $G \in \mathbb{R}^3$. With each training iteration, PID gains generated by the neural network are ingested by the PID simulator to produce a corrected spill $c \in \mathbb{R}^{430}$ (10 points per millisecond in the 43 ms spill). Each corrected spill has an associated SDF value, which is a volatility metric that varies in the range (0,1]. The loss function ℓ is defined as $\ell = (1 - \text{SDF})^2$, the squared error between the actual and target SDF. Gradients with respect to the PID gains $\partial\ell/\partial G$ are calculated directly from data and are backpropagated through the simulator to update the neural network weights and minimize the loss (i.e., maximize the PID performance).

We refer to this approach as a *Hybrid* ML Simulator because only those functions which must be differentiable are made so. This allows functions such as noise generation and tune ramps to be pre-computed and excluded from the more computationally expensive gradient calculation and backpropagation steps.

10.3.2. Training Procedure: PID Gains Tuning

An SDF of 1 corresponds to perfect regulation (zero volatility in the corrected spill). The SDF is thus used as the objective function with a target of 1 (perfect regulation). To optimize the PID gains, we minimize the loss function ℓ. The procedure for training the neural network to generate optimal PID gains is as follows: (i) the neural network is

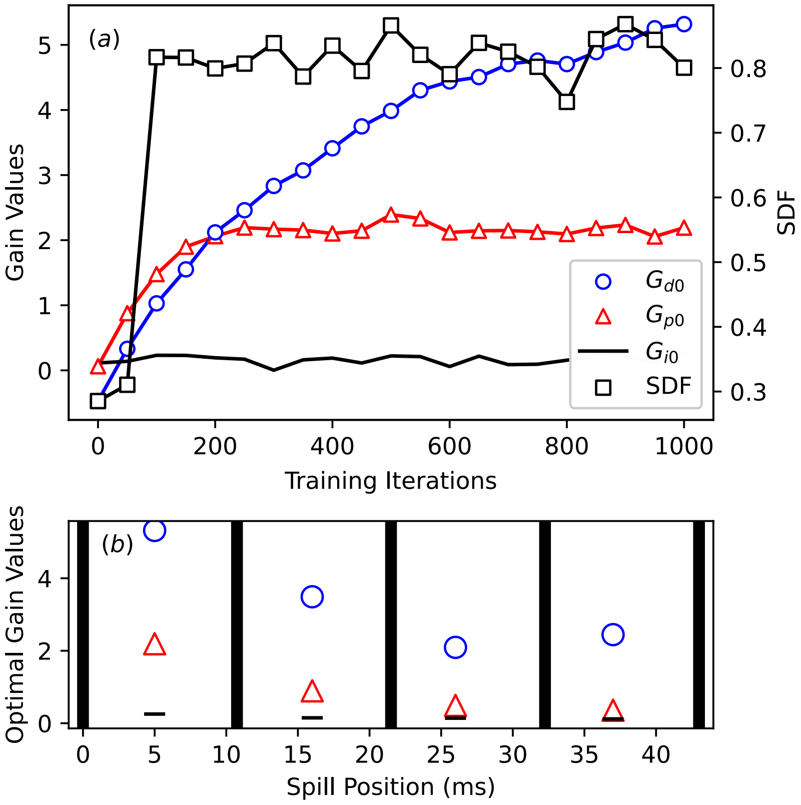

Figure 10.2. Top (a) Evolution of the PID gains in domain-0 (leftmost subdomain of the bottom plot) over the full spill, as well as the SDF. Bottom (b) Four subdomains of the spill are segmented by vertical bars. Optimal gain values within each subdomain are shown on the vertical axis.

initialized randomly and generates a set of PID gains which are fed alongside a random noise signal into the PID simulator, (ii) the PID simulator uses these gains to produce a corrected spill c, (iii) the SDF value of this spill is compared to the target and the gradients of the loss function ℓ are calculated, (iii) gradients with respect to the PID gains $\partial\ell/\partial G$ are backpropagated **[90]** through the simulator and the neural network to minimize the error, (iv) the updated neural network generates a new set of PID gain values and the process repeats. Note that each iteration uses independent noise profiles and minimizes the error over an *entire spill*. All differentiable functions are built with PyTorch [**84**], and we use the Adam Optimizer [**63**] with learning rate 0.01 for training.

10.3.3. Spill Segmentation

Since the ideal quad current ramp is non-linear in time, the extraction rate is sensitive to varied degrees in different parts of the spill. To characterize this variation, our system is capable of independently optimizing PID gains within arbitrary subdomains of the spill. Fig. 10.2(a) shows how the system evolves towards optimal PID gain values for 4 subdomains plotted along with the full spill's SDF value. Fig. 10.2(b) shows optimal PID gain values within four equally sized subdomains of the spill.

10.3.4. Regulation Performance

Our system is able to identify PID gains that realize high SDF values on representative input noise distributions (noise has an average SDF of 0.5). After 1000 training iterations, we can achieve corrected spills with median SDF values of 0.74, and by splitting the spill into four subdomains, median SDF values of 0.83, representing 48% and 66% reductions in the noise, respectively. If achieved in reality, this would well satisfy the experimental requirements of Mu2e. The regulation is especially effective on the high spikes, which are the main concerns for the detector performance.

11 Mu2e: Learnable Controllers in Noisy Environments

Next, we tackle the problem of control directly, avoiding the PID altogether and attempting to learn to generate the control signal with a recurrent sequence model.

As we have already covered, a third-integer resonant slow extraction system is being developed for the Fermilab's Delivery Ring to deliver protons to the Mu2e experiment. During a slow extraction process, the beam on target is liable to experience small intensity variations due to many factors. Owing to the experiment's strict requirements in the quality of the spill, an advanced Spill Regulation System (SRS) is currently under design. The SRS primarily consists of three components - slow regulation, fast regulation, and harmonic content tracker. In this chapter, we shall present the investigations of using Machine Learning (ML) in the fast regulation system, including further optimizations of PID controller gains for the fast regulation, prospects of an ML agent completely replacing the PID controller using supervised learning schemes such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) ML models, the simulated impact and limitation of machine response characteristics on the effectiveness of both PID and

ML regulation of the spill. We also present here early results of Reinforcement Learning efforts, including continuous-action actor-critic methods and soft actor-critic methods, to regulate the spill rate.

11.1. Resonant Extraction for Mu2e

Resonant extraction is a beam physics technique employed to extract a slice of beam turn by turn in a circular accelerator by exciting third integer resonance using dedicated sextupole magnets.

Slow extraction at Fermilab is to be done at the Delivery Ring (DR) using a circuit of 6 harmonic sextupoles and 3 fast quadrupoles, driving the horizontal tune from 9.650 to 9.666 to extract 10^{12} protons over the course of 43 ms ($\approx$ 25000 turns) to be sent to the muon production target. Any protons left in the DR after 43 ms would be aborted. The ideal spill quality would be to extract 3×10^7 protons every turn, but we expect the spill quality to be heavily affected by irregularities due to noises that could arise from various accelerator components and other factors.

The *Mu2e* experiment imposes strict requirements on the spill quality since the detectors are sensitive to the intensity variations of particles coming off of the muon production target. We thus need a robust regulation system in place to ensure high and steady spill quality.

11.2. Regulation System

The spill regulation system (SRS) comprises slow regulation, fast regulation, and harmonic content canceller. The present design bandwidth of the regulation system is 10

KHz, i.e., 430 data points within one spill. A brief functional overview of the SRS can be found at [**79**].

11.2.1. Fast Regulation System

The fast regulation in the SRS concerns the control of spill quality *within* one spill to curtail any semi-random noise that might arise and also suppress any high frequency 60 Hz harmonic content that the harmonic content suppressor system is not able to suppress. The fast regulation would be supplemented on top of the slow regulation through a fast feedback loop. This loop will send a control signal update at every time step within one spill, and this signal will superpose to the (already preloaded) quad current ramp of the tune ramping quadrupoles to reduce the instantaneous noises in the spill rate.

One way to implement the fast regulation is through PID feedback control, which is a fairly robust and proven technique. However, given the non-linearity of the resonance process, low spill time, and the strict requirements from the Mu2e experiment on the spill quality, we also investigate the possible use of machine learning to enhance the fast regulation.

If the ideal spill rate is normalized to 1, the quality of the spill is defined by the spill duty factor (SDF),

$$\text{SDF} = \frac{1}{1+\sigma^2_{\text{spill}}} \tag{11.1}$$

where σ_{spill} is the standard deviation of the spill rate. An ideal spill would bear a constant spill rate value of 1, giving us an SDF of 1. The goal of the SRS for *Mu2e* is to obtain an SDF of 0.6 or higher.

11.3. Analytical Modelling of Extraction with Fast Regulation

Modeling of the extraction is necessary to validate various regulation schemes in simulations. Particle tracking could be very computationally expensive, we thus developed a simplified analytical model of the spill process to verify the efficiency of different regulation schemes.

11.3.1. Physics Simulator

We built a physics simulator in Python (using the analytical model) to simulate the spill process and generate the required spill data to train various machine learning models. The details of the physics simulator can be found in our earlier work [**79**].

To train a machine learning model to regulate the extracted spill intensity, we need a way to simulate the effects of the models' regulation signals on the extracted spill. This, then, requires that the physics simulator itself must also be fully differentiable, as gradients must be able to flow back from the spill signal through the physics simulator and into the ML model. To accomplish this, we leveraged differentiable operations in the widely used ML framework PyTorch [**84**] to produce a fully-differentiable clone of the existing simulator. Every operation in the original simulator, including the low-pass filter for limiting the effective interaction rate of the control system, has been replaced with its differentiable counterpart in PyTorch. This allows us to propagate gradients through the simulator and update the weights of the machine learning model. In the following section, we motivate the use of ML in this context and discuss why our approach is not only effective but theoretically sound.

11.4. ML for Spill Regulation

Our objective is to construct a machine learning system that maps measured spill values (deviations from the ideal spill) onto quadrupole currents that bring the spill intensity back to the target intensity level, thereby smoothing the noise inherent in resonant extraction. Historically, this problem was solved using proportional–integral–derivative controllers (PIDs) which are control loop mechanisms designed to maintain a system in a steady state in the presence of external perturbations. While simple, effective, and well understood, PIDs are a linear and symmetric heuristic control system with constant parameters, meaning they are designed to operate in domains in which the response of the system is invariant across all operating regions. And while approaches such as gain scheduling [**118**] exist to address this issue, any PID-based regulation system will still be heuristic. As we cannot presume the exact noise distribution and possible nonlinearities in the extraction system, a control system that is 1) nonlinear and 2) capable of adapting to the idiosyncratic response of the extraction system, is warranted. As modern neural networks represent a class of arbitrary nonlinear function approximators, they are a natural solution for extending resonant extraction control systems into the nonlinear regime.

Broadly, the machine learning problem can be formulated as follows. Given an input spill $x \in \mathbb{R}^{430}$, we are looking for a model $f(\cdot)$, parameterized by trainable parameters θ, such that:

$$f_\theta : x \in \mathbb{R}^{430} \rightarrow q \in \mathbb{R}^{430} \tag{11.2}$$

where q is a sequence of quadrupole correction currents. Note that we do not index the input spills because each spill is generated procedurally and is entirely random. As mentioned before, the effect of these quadrupole correction currents must be simulated using a differentiable simulator we denote $SIM(\cdot)$, which is a differentiable function mapping $q \in \mathbb{R}^{430} \rightarrow s \in \mathbb{R}^{430}$, where s is the final corrected spill. Note that while $SIM(\cdot)$ is differentiable, it does not contain any trainable parameters. The final spill profile is then $s = SIM\big(f_\theta(x), \lambda\big)$, where x is the input noise profile and λ is the low-pass filter value that effectively puts an upper limit on the interaction frequency of the regulation system. We include it here because it has a strong impact on regulation performance and is tunable via the use of different materials for the beampipe.

11.4.1. Supervised Learning

In supervised learning, our goal is to learn a model that minimizes the difference between the models' output and the supervision label. As our ultimate goal is to minimize the noise inherent in the spill, we use as our supervision signal the difference between the SDF of the corrected spill (generated by our model) and the SDF of an ideal spill (noiseless spill intensity with a dimensionless value of 1). The training loss ℓ then becomes:

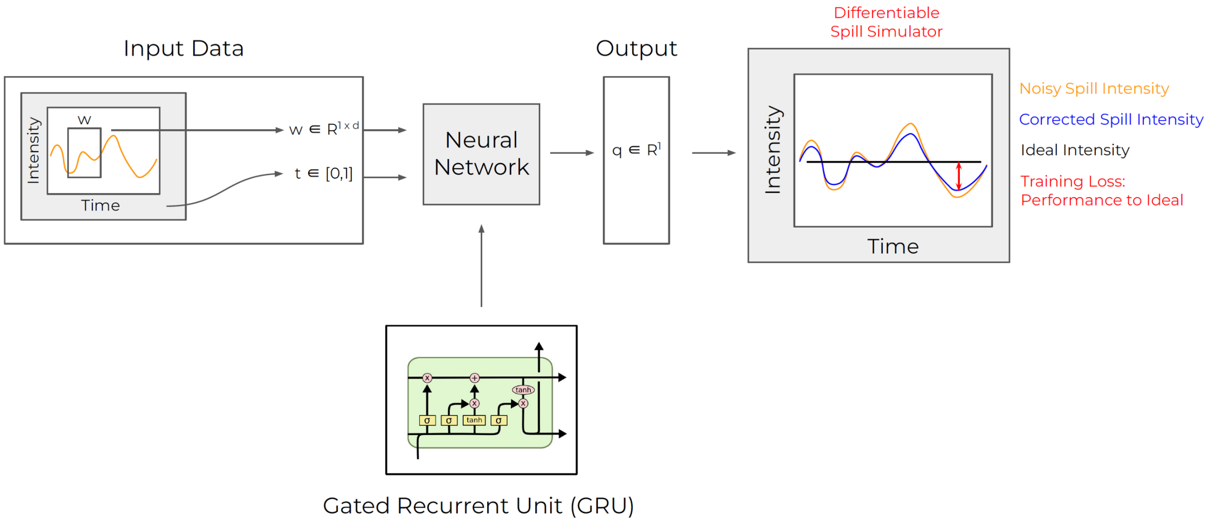

Figure 11.1. High level overview of the learning model in the supervised setting. A window of data is ingested, a quadrupole correction current is predicted, and the differentiable spill simulator calculates the affect of that correction current. We are trying to minimize the difference between the ideal spill and the corrected spill. The image for the GRU is borrowed from Christopher Olah's excellent review of recurrent models.

$$\ell = \text{MSE}\big(\, s, \vec{\mathbf{1}}\,\big) \tag{11.3}$$

Given the formulation in Eq. 11.2, a natural choice is a recurrent network, which ingests an input sequence and produces an output sequence of equal size. Towards this end, we implement a simple GRU **[22]** having two layers, a hidden state of size 128 that ingests, at each point in the spill, a window of the past 40 observations. To allow the model to begin regulation at the first step in the spill, we prepend the spill profile with a vector of length 40. Each element in this prepended vector is assigned the value of the first element in the random noise profile. Then, we walk forward in the usual manner to generate 430 quadrupole corrections. At the end of each spill, the loss is calculated using Equation 11.3 and the model parameters θ are updated.

Figure 11.1 builds upon Figures 2.10 and 2.12 to give an intuitive understanding of how our machine learning model will interact with the physics simulator using supervised learning to update the model weights.

11.4.2. Reinforcement Learning

Reinforcement Learning (RL) is a machine learning methodology based on maximizing rewards and minimizing penalties through trial and error. While our RL solution is still under development, we briefly describe and motivate it here. Of primary interest is the capacity for RL models to be trained on real-world observations, i.e. outside of a simulator, and, as our ultimate goal is to control a physical device, RL represents a compelling opportunity.

RL can be modeled as a Markov Decision Process (MDP) involving environmental states, actions, and rewards. The magnitude of the reward is a function of the state-action pair, which is often dependent on stochastic transition probabilities describing the likelihood of some action 'successfully' transitioning the environment from one state to another.

Following the formulation in Eq 11.2, our goal is to learn a policy mapping noised spills onto quadrupole corrections. At each point in the spill, our RL agent ingests a window of the past 10 observations, in RL called the 'state space', and the produces a single real-valued number corresponding to quadrupole correction, in RL called the 'action space'. For each spill, the model produces 430 such corrections, and our goal is to learn a policy that maximizes the reward (performance) over the entire spill.

To do this, we implement Soft Actor Critic (SAC), which is an off-policy, policy gradient-based algorithm that we chose for two main reasons. First, it is designed to balance exploration and exploitation via entropy regularization, and second, it utilizes a method called the double-Q trick to reduce the bias. Both of these help to stabilize training in settings like ours having large or continuous action spaces.

To prevent the errors from accumulating, we define a termination condition. If the SDF of the corrected spill drops below a predefined threshold, we immediately assign a large negative reward to discourage the agent from continuing down that path. If this condition is met more than 10 times in a spill, we stop the MDP trajectory and start a new one.

Finally, to improve the sample efficiency, we implement replay memory, which is a technique for extracting more information from each sample by storing it and learning from it multiple times.

11.4.3. Results, Challenges, and Future Work

In Figure 11.2, we show the average corrected SDF of using the GRU trained in a supervised manner and the PID. We note that this is as fair a comparison as can be made, as we optimize the PID gain values independently for each low-pass frequency using the differentiable simulator introduced in our previous work [**79**]. The SDF of the input noise for all trials had an average SDF of 0.51. The GRU, trained in a supervised setting, is able to robustly outperform the PID at all low-pass frequencies. Additionally, we see that

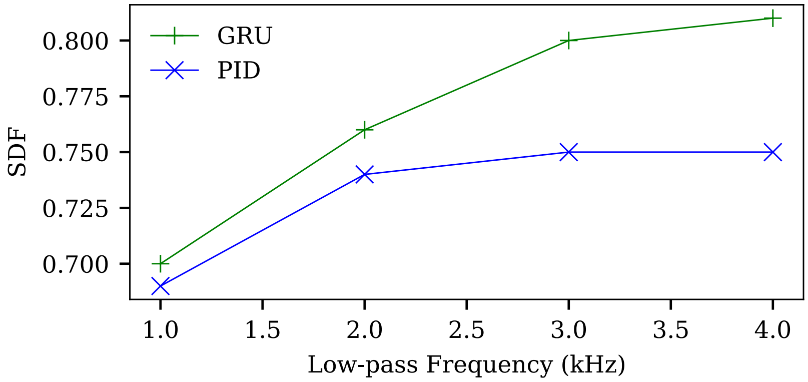

Figure 11.2. Regulation performance of the GRU vs. PID. Each point represents the average SDF over 1k independent spills with identical noise profiles. Input noise had an average SDF of 0.51 for all trials.

the advantage increases with the effective interaction frequency, indicating that the value of capturing nonlinear characteristics of the regulation system scales with the interaction frequency. We put a particular emphasis on this point as it is relevant to physical design choices, particularly for those machine elements that limit the beam response time.

As PID controllers are used not just across Fermilab but many engineering disciplines, we hope these results serve as an impetus for continued exploration into learnable control systems in other environments.

12 | Conclusions

In this dissertation, we've presented a series of works purpose-built to augment and accelerate the scientific process. This included a principled method for learning to drop noisy edges in attention-based GNNs, the first graph pooling algorithm capable of coarsening a graph into an arbitrary number of variable-sized clusters, and the first guided molecular generation methodology for optimizing candidate drug compounds via atomic replacement. Additionally, we showed how semantic segmentation models could be adapted to better understand particle accelerator loss profiles, and how standard recurrent sequence models models could beat the ubiquitous PID controller at its own game.

We argued that machine learning, and specifically graph machine learning, represents the latest in the lineage of tools that have revolutionized science. The methods we introduce take steps in this direction, but there is a great deal of work that needs to be done. Central to realizing the promise of graph ML as a scientific accelerant is a firmer grasp on model interpretability. Adoption and integration will scale with trust, especially for those outside of the machine learning community, the black-box nature of deep neural nets can be difficult to get past.

The message we send to the practitioner should be clear and confident: machine learning is unequivocally useful, as proved many times over in research and industrial labs around the world, and to forgo its use altogether would be a mistake. We need to understand our data, implement with caution, carefully test, monitor in an ongoing fashion, and treat it like we would any other tool in our toolkit: as a tool, not a panacea.

In closing, we choose to study graphs in the context of machine learning because it encourages us to build our models with the same kind of architecture that humans evolved to understand complex systems: to think in terms of objects, their properties, and the relationships between them. This representation is what allows us to *make infinite use of finite means* [**110**] and broaden the efficacy of our knowledge through analogy. At the end of the day, building a similar structure-awareness into our learning models is a utilitarian effort; it reduces sample complexity and improves learning efficiency by leveraging the rich, relational information implicit in the data. But I think it bears mentioning that there is something more fundamental about the role that internal structure plays. Be they atoms, proteins, families or economies, it's the relationships between their constituent parts that make them interesting, unique, and meaningful.

——— * * * ———

I've long held the belief that we honor our universe by understanding it, and it has been one of the great privileges of my life to work on tools that may one day help to enrich that understanding.

Bibliography

[1] Document: Experimental in vitro DMPK and physicochemical data on a set of publicly disclosed compounds. https://www.ebi.ac.uk/chembl/document_report_card/CHEMBL3301361/. Accessed: 2023-5-16.

[2] Fermilab. https://mu2e.fnal.gov/. Accessed: 2024-1-12.

[3] Molecule of the month: Topoisomerases. https://pdb101.rcsb.org/motm/73. Accessed: 2024-1-17.

[4] Mu2e Technical Design Report, 2015.

[5] Ainsworth, R., Adamson, P., Brown, B., Capista, D., Hazelwood, K., Kourbanis, I., Morris, D., Xiao, M., and Yang, M.-J. High Intensity Proton Stacking at Fermilab: 700 kW Running. In *61st ICFA Advanced Beam Dynamics Workshop on High-Intensity and High-Brightness Hadron Beams* (2018), p. TUA1WD04.

[6] Alberts, B., Johnson, A., Lewis, J., Raff, M., Roberts, K., and Walter, P. *The Shape and Structure of Proteins*. Garland Science, 2002.

[7] Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization.

[8] Baek, J., Kang, M., and Ju Hwang, S. Accurate learning of graph representations with graph multiset pooling, June 2021.

[9] Bak, P. *How Nature Works: the science of self-organized criticality*, first edition ed. Copernicus, Apr. 1999.

[10] Bartoszek, L., et al. Mu2e Technical Design Report.

[11] Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G., Vaswani, A., Allen, K., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., and Pascanu, R. Relational inductive biases, deep learning, and graph networks.

[12] Baumbaugh, A., Briegel, C., Drennan, C. C., Fellenz, B., Knickerbocker, K. L., Lewis, J. D., Pordes, S., Thurman-Keup, R. M., and Utes, M. J. Beam Loss Monitor Upgrade Users' Guide, 7 2010.

[13] Berlioz, J., Hazelwood, K., Ibrahim, M., Austin, M., Arnold, J., Schupbach, B., Seiya, K., and Thurman-Keup, R. Synchronous High-Frequency Distributed Readout for Edge Processing at the Fermilab Main Injector and Recycler. presented at NAPAC'22, Albuquerque, New Mexico, USA, Aug. 2022, paper MOPA15, this conference.

[14] Bocci, G., Carosati, E., Vayer, P., Arrault, A., Lozano, S., and Cruciani, G. ADME-Space: a new tool for medicinal chemists to explore ADME properties. *Scientific reports 7*, 1 (July 2017), 6359.

[15] Bohr, N. I. on the constitution of atoms and molecules. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 26*, 151 (July 1913), 1–25.

[16] Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S. V. N., Smola, A. J., and Kriegel, H.-P. Protein function prediction via graph kernels. *Bioinformatics 21 Suppl 1* (June 2005), i47–56.

[17] Brady, R., and Anderson, R. Maxwell's fluid model of magnetism.

[18] Brand, C. O., Mesoudi, A., and Smaldino, P. E. Analogy as a catalyst for cumulative cultural evolution. *Trends in cognitive sciences 25*, 6 (June 2021), 450–461.

[19] Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and locally connected networks on graphs.

[20] Callaway, E. 'the entire protein universe': AI predicts shape of nearly every known protein. `http://dx.doi.org/10.1038/d41586-022-02083-2`, July 2022. Accessed: 2024-1-22.

[21] Chen, Y., Wu, L., and Zaki, M. J. Iterative deep graph learning for graph neural networks: Better and robust node embeddings.

[22] Cho, K., van Merrienboer, B., Bahdanau, D., and Bengio, Y. On the properties of neural machine translation: Encoder-Decoder approaches.

[23] Committee on the Design and Evaluation of Safer Chemical Substitutions: A Framework to Inform Government and Industry Decision, Board on Chemical Sciences and Technology, Board on Environmental Studies and Toxicology, Division on Earth and Life Studies, and National Research Council. *Physicochemical Properties and Environmental Fate*. National Academies Press (US), Oct. 2014.

[24] Debnath, A. K., Lopez de Compadre, R. L., Debnath, G., Shusterman, A. J., and Hansch, C. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry 34*, 2 (Feb. 1991), 786–797.

[25] Diehl, F. Edge contraction pooling for graph neural networks. *arXiv.org* (2019).

[26] Doyle, R., Simons, K., Qian, H., and Baker, D. Local interactions and the optimization of protein folding. *Proteins 29*, 3 (Nov. 1997), 282–291.

[27] Fang, X., Liu, L., Lei, J., He, D., Zhang, S., Zhou, J., Wang, F., Wu, H., and Wang, H. Geometry-enhanced molecular representation learning for property prediction. *Nature Machine Intelligence 4*, 2 (Feb. 2022), 127–134.

[28] Feinberg, E. N., Joshi, E., Pande, V. S., and Cheng, A. C. Improvement in ADMET prediction with multitask deep featurization. *Journal of medicinal chemistry 63*, 16 (Aug. 2020), 8835–8848.

[29] Fey, M., and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds* (2019).

[30] Fey, M., and Lenssen, J. E. Fast graph representation learning with PyTorch geometric. *ArXiv* (2019).

[31] Feynman, R. P., Leighton, R. B., and Sands, M. *The Feynman lectures on physics; New millennium ed.* Basic Books, New York, NY, 2010. Originally published 1963-1965.

[32] Franceschi, L., Niepert, M., Pontil, M., and He, X. Learning discrete structures for graph neural networks. In *Proceedings of the 36th International Conference on Machine Learning* (2019), K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97 of *Proceedings of Machine Learning Research*, PMLR, pp. 1972–1982.

[33] Gao, H., and Ji, S. Graph U-Nets. *IEEE transactions on pattern analysis and machine intelligence PP* (May 2021).

[34] Gasteiger, J. *Chemoinformatics.* Wiley-VCH, Weinheim, 2003.

[35] Gaudelet, T., Day, B., Jamasb, A. R., Soman, J., Regep, C., Liu, G., Hayter, J. B. R., Vickers, R., Roberts, C., Tang, J., Roblin, D., Blundell, T. L., Bronstein, M. M., and Taylor-King, J. P. Utilizing graph machine learning within drug discovery and development. *Briefings in bioinformatics 22*, 6 (Nov. 2021).

[36] Gentner, D., and Markman, A. B. Structure mapping in analogy and similarity. *The American psychologist 52*, 1 (1997), 45–56.

[37] Giles, C. L., Bollacker, K. D., and Lawrence, S. CiteSeer: an automatic citation indexing system. In *Proceedings of the third ACM conference on Digital libraries* (New York, NY, USA, May 1998), DL '98, Association for Computing Machinery, pp. 89–98.

[38] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry.

[39] Goodwin, G. P., and Johnson-Laird, P. N. Reasoning about relations. *Psychological review 112*, 2 (Apr. 2005), 468–493.

[40] Gori, M., Monfardini, G., and Scarselli, F. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* (2005), vol. 2, IEEE, pp. 729–734 vol. 2.

[41] Grange, J., Guarino, V., Winter, P., Wood, K., Zhao, H., Carey, R. M., Gastler, D., Hazen, E., Kinnaird, N., Miller, J. P., Mott,

J., Roberts, B. L., Benante, J., Crnkovic, J., Morse, W. M., Sayed, H., Tishchenko, V., Druzhinin, V. P., Khazin, B. I., Koop, I. A., Logashenko, I., Shatunov, Y. M., Solodov, E., Korostelev, M., Newton, D., Wolski, A., Chapelain, A., Bjorkquist, R., Eggert, N., Frankenthal, A., Gibbons, L., Kim, S., Mikhailichenko, A., Orlov, Y., Rubin, D., Sweigart, D., Allspach, D., Annala, G., Barzi, E., Bourland, K., Brown, G., Casey, B. C. K., Chappa, S., Convery, M. E., Drendel, B., Friedsam, H., Gadfort, T., Hardin, K., Hawke, S., Hayes, S., Jaskierny, W., Johnstone, C., Johnstone, J., Kashikhin, V., Kendziora, C., Kiburg, B., Klebaner, A., Kourbanis, I., Kyle, J., Larson, N., Leveling, A., Lyon, A. L., Markley, D., McArthur, D., Merritt, K. W., Mokhov, N., Morgan, J. P., Nguyen, H., Ostiguy, J.-F., Para, A., Popovic, C. C. P. M., Ramberg, E., Rominsky, M., Schoo, D., Schultz, R., Still, D., Soha, A. K., Strigonov, S., Tassotto, G., Turrioni, D., Villegas, E., Voirin, E., Velev, G., Welty-Rieger, L., Wolff, D., Worel, C., Wu, J.-Y., Zifko, R., Jungmann, K., Onderwater, C. J. G., Debevec, P. T., Ganguly, S., Kasten, M., Leo, S., Pitts, K., Schlesier, C., Gaisser, M., Haciomeroglu, S., Kim, Y.-I., Lee, S., Lee, M.-J., Semertzidis, Y. K., Giovanetti, K., Baranov, V. A., Duginov, V. N., Khomutov, N. V., Krylov, V. A., Kuchinskiy, N. A., Volnykh, V. P., Crawford, C., Fatemi, R., Gohn, W. P., Gorringe, T. P., Korsch, W., Plaster, B., Anastasi, A., Babusci, D., Dabagov, S., Ferrari, C., Fioretti, A., Gabbanini, C., Hampai, D., Palladino, A., Venanzoni, G., Bowcock, T., Carroll, J., King, B., Maxfield, S., McCormick, K., Price, J., Sim, D., Smith, A., Teubner, T., Turner, W., Whitley, M., Wormald, M., Chislett, R., Kilani, S., Lancaster, M., Motuk, E., Stuttard, T., Warren, M., Flay, D., Kawall, D., Meadows, Z., Chupp, T., Raymond, R., Tewlsey-Booth, A., Syphers, M. J., Tarazona, D., Catalonotti, S., Stefano, R. D., Iacovacci, M., Mastroianni, S., Chattopadhyay, S., Eads, M., Fortner, M., Hedin, D., Pohlman, N., de Gouvea, A., Schellman, H., Welty-Rieger, L., Azfar, F., Henry, S., Alkhazov, G. D., Golovtsov, V. L., Neustroev, P. V., Uvarov, L. N., Vasilyev, A. A., Vorobyov, A. A., Zhalov, M. B., Cerrito, L., Gray, F., Sciascio, G. D., Moricciani, D., Fu, C., Ji, X., Li, L., Yang, H., Stöckinger, D., Cantatore, G., Cauz, D., Karuza, M., Pauletta, G., Santi, L., Baessler, S., Bychkov, M., Frlez, E., Pocanic, D., Alonzi, L. P., Fertl, M., Fienberg, A., Froemming, N., Garcia, A., Kaspar, D. W. H. J., Kammel, P., Osofsky, R., Smith, M., Swanson, E., van Wechel, T., and Lynch, K. Muon (g-2) Technical Design Report, 2018.

[42] Grattarola, D., Zambon, D., Bianchi, F. M., and Alippi, C. Understanding pooling in graph neural networks.

[43] Güner, O., and Bowen, J. P. Setting the record straight: the origin of the pharmacophore concept.

[44] Hamilton, W. L. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning 14*, 3, 1–159.

[45] Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs.

[46] Hazelwood, K., Ibrahim, M., Liu, H., Memik, S., Nagaslaev, V. P., Narayanan, A., Nicklaus, D., Prieto, P., Seiya, K., Shi, R., Schupbach, B., Thieme, M., Thurman-Keup, R., and Tran, N. Real-time Edge AI For Distributed Systems (READS): Progress On Beam Loss De-blending for the Fermilab Main Injector and Recycler. In *Conference IPAC'21, Campinas, Brazil* (2021).

[47] Hofstadter, D. R. *Gödel, Escher, Bach: An Eternal Golden Braid*, 20 ed. Basic Books, Feb. 1999.

[48] Hofstadter, D. R. *I Am a Strange Loop*, reprint edition ed. Basic Books, July 2008.

[49] Hofstadter, D. R., and Sander, E. *Surfaces and Essences: Analogy as the Fuel and Fire of Thinking*, 1 ed. Basic Books, Apr. 2013.

[50] Huang, K., Fu, T., Gao, W., Zhao, Y., Roohani, Y., Leskovec, J., Coley, C. W., Xiao, C., Sun, J., and Zitnik, M. Artificial intelligence foundation for therapeutic science. *Nature chemical biology 18*, 10 (Oct. 2022), 1033–1036.

[51] Ibrahim, M. A., Cullerton, E., Diamond, J., Martin, K., Prieto, P. S., Scarpine, V., and Varghese, P. Preliminary Design of Mu2e Spill Regulation System (SRS), 2019.

[52] Institute of Medicine (US) Committee on Military Nutrition Research, Poos, M. I., Costello, R., and Carlson-Newberry, S. J. *The Role of Protein and Amino Acids in Sustaining and Enhancing Performance*. National Academies Press (US), 1999.

[53] Issue, S. MAIN INJECTOR. `https://www.fnal.gov/pub/ferminews/FermiNews99-06-01.pdf`. Accessed: 2024-1-14.

[54] Jackson, G. The Fermilab Recycler Ring Technical Design Report. Revision 1.2.

[55] Jang, E., Gu, S., and Poole, B. Categorical reparameterization with Gumbel-Softmax.

[56] Jiang, X. F., Chen, T. T., and Zheng, B. Structure of local interactions in complex financial dynamics. *Scientific reports 4* (June 2014), 5321.

[57] Jo, J., Baek, J., Lee, S., Kim, D., Kang, M., and Hwang, S. J. Edge representation learning with hypergraphs.

[58] Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A. A., Ballard, A. J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., Back, T., Petersen, S., Reiman, D., Clancy, E., Zielinski, M., Steinegger, M., Pacholska, M., Berghammer, T., Bodenstein, S., Silver, D., Vinyals, O.,

Senior, A. W., Kavukcuoglu, K., Kohli, P., and Hassabis, D. Highly accurate protein structure prediction with AlphaFold. *Nature 596*, 7873 (Aug. 2021), 583–589.

[59] Kemp, C., and Tenenbaum, J. B. The discovery of structural form. *Proceedings of the National Academy of Sciences of the United States of America 105*, 31 (Aug. 2008), 10687–10692.

[60] Kenneth. *The Nature of Explanation*, 1 ed. Cambridge University Press, Oct. 1967.

[61] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization. *ICLR* (2014).

[62] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization.

[63] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (2015), Y. Bengio and Y. LeCun, Eds.

[64] Kipf, T., Fetaya, E., Wang, K.-C., Welling, M., and Zemel, R. Neural relational inference for interacting systems. *Proceedings of the 35th International Conference on Machine Learning* (Feb. 2018).

[65] Kipf, T. N., and Welling, M. Semi-Supervised classification with graph convolutional networks.

[66] Lam, R., Sanchez-Gonzalez, A., Willson, M., Wirnsberger, P., Fortunato, M., Alet, F., Ravuri, S., Ewalds, T., Eaton-Rosen, Z., Hu, W., Merose, A., Hoyer, S., Holland, G., Vinyals, O., Stott, J., Pritzel, A., Mohamed, S., and Battaglia, P. Learning skillful medium-range global weather forecasting. *Science 382*, 6677 (Dec. 2023), 1416–1421.

[67] Landrum, G. Rdkit: Open-source cheminformatics software.

[68] Lee, J., Lee, I., and Kang, J. Self-Attention graph pooling.

[69] Lipinski, C. A., Lombardo, F., Dominy, B. W., and Feeney, P. J. Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings. *Advanced drug delivery reviews 46*, 1-3 (Mar. 2001), 3–26.

[70] Luo, D., Cheng, W., Yu, W., Zong, B., Ni, J., Chen, H., and Zhang, X. Learning to drop: Robust graph neural network via topological denoising.

[71] Mandelbrot, B. B. *Fractals: Form, Chance and Dimension*, reprint ed. edition ed. Echo Point Books & Media, Feb. 2020.

[72] Mathur, J. Local interactions shape plant cells. *Current opinion in cell biology 18*, 1 (Feb. 2006), 40–46.

[73] Maxwell, J. C. On physical lines of force. *Philosophical Magazine 90*, sup1 (Feb. 2010), 11–23.

[74] McCallum, A. K., Nigam, K., Rennie, J., and Seymore, K. Automating the construction of internet portals with machine learning. *Information retrieval 3*, 2 (July 2000), 127–163.

[75] McPherson, M., Smith-Lovin, L., and Cook, J. M. Birds of a feather: Homophily in social networks. *Annual review of sociology 27*, 1 (Aug. 2001), 415–444.

[76] Miao, S., Liu, M., and Li, P. Interpretable and generalizable graph learning via stochastic attention mechanism.

[77] Minaee, S., Boykov, Y., Porikli, F., Plaza, A., Kehtarnavaz, N., and Terzopoulos, D. Image segmentation using deep learning: A survey. *IEEE transactions on pattern analysis and machine intelligence 44*, 7 (July 2022), 3523–3542.

[78] Narayanan, A. THIRD-INTEGER RESONANT EXTRACTION REGULATION SYSTEM FOR MU2E. *Graduate Research Theses & Dissertations.* (2022).

[79] Narayanan, A., Hazelwood, K., Ibrahim, M., Liu, H., Memik, S., Nagaslaev, V. P., Nicklaus, D., Prieto, P., Seiya, K., Shi, R., Schupbach, B., Thieme, M., Thurman-Keup, R., and Tran, N. Optimizing Mu2e Spill Regulation System Algorithms. In *Conference IPAC'21, Campinas, Brazil, paper THPAB243, this conference* (2021).

[80] Navon, D. Forest before trees: The precedence of global features in visual perception. *Cognitive psychology 9*, 3 (July 1977), 353–383.

[81] Noether, E. Invariante variationsprobleme. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse 1918* (1918), 235–257.

[82] Oono, K., and Suzuki, T. Graph neural networks exponentially lose expressive power for node classification. *ICLR* (2019).

[83] Özgür, O. Local interactions. In *Handbook of Social Economics*, J. Benhabib, A. Bisin, and M. O. Jackson, Eds., vol. 1. North-Holland, Jan. 2011, pp. 587–644.

[84] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.

[85] Paul, S. M., Mytelka, D. S., Dunwiddie, C. T., Persinger, C. C., Munos, B. H., Lindborg, S. R., and Schacht, A. L. How to improve R&D productivity: the pharmaceutical industry's grand challenge. *Nature reviews. Drug discovery 9*, 3 (Mar. 2010), 203–214.

[86] Plaut, D. C., McClelland, J. L., Seidenberg, M. S., and Patterson, K. Understanding normal and impaired word reading: computational principles in quasi-regular domains. *Psychological review 103*, 1 (Jan. 1996), 56–115.

[87] Putz, M. V., Duda-Seiman, C., Duda-Seiman, D., Putz, A.-M., Alexandrescu, I., Mernea, M., and Avram, S. Chemical structure-biological activity models for pharmacophores' 3d-interactions. *International journal of molecular sciences 17*, 7 (2016), 1087.

[88] Ranjan, E., Sanyal, S., and Talukdar, P. P. ASAP: Adaptive structure aware pooling for learning hierarchical graph representations.

[89] Ronneberger, O., Fischer, P., and Brox, T. U-Net: Convolutional networks for biomedical image segmentation.

[90] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. In *Neurocomputing: foundations of research*. MIT Press, Cambridge, MA, USA, Jan. 1988, pp. 696–699.

[91] Schueler, F. W. Chemobiodynamics and drug design. *Academic Medicine 36*, 3 (1961), 285–286.

[92] Seidel, T., Wieder, O., Garon, A., and Langer, T. Applications of the pharmacophore concept in natural product inspired drug design. *Molecular Informatics 39*, 11 (2020), 2000059.

[93] Seiya, K., Hazelwood, K. J., Ibrahim, M. A., Nagaslaev, V. P., Nicklaus, D. J., Schupbach, B. A., Thurman-Keup, R. M., Tran, N. V., Liu, H., and Memik, S. Accelerator Real-time Edge AI for Distributed Systems (READS) Proposal.

[94] Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. Collective classification in network data. *AI Magazine 29*, 3 (Sept. 2008), 93–93.

[95] Senior, A. W., Evans, R., Jumper, J., Kirkpatrick, J., Sifre, L., Green, T., Qin, C., Žídek, A., Nelson, A. W. R., Bridgland, A., Penedones, H., Petersen, S., Simonyan, K., Crossan, S., Kohli, P., Jones, D. T., Silver, D., Kavukcuoglu, K., and Hassabis, D. Improved protein structure prediction using potentials from deep learning. *Nature 577*, 7792 (Jan. 2020), 706–710.

[96] Shang, C., Chen, J., and Bi, J. Discrete graph structure learning for forecasting multiple time series.

[97] Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. Pitfalls of graph neural network evaluation. *ArXiv* (2018).

[98] Southey, M. W. Y., and Brunavs, M. Introduction to small molecule drug discovery and preclinical development. *Frontiers in Drug Design and Discovery 3* (2023).

[99] Sterrett, S. G. Darwin's analogy between artificial and natural selection: how does it go? *Studies in History and Philosophy of Science Part C: Studies in History and Philosophy of Biological and Biomedical Sciences 33*, 1 (Mar. 2002), 151–168.

[100] Stocker, R., Jelinek, H., and Durnota, B. *Complex Systems: From Local Interactions to Global Phenomena.* IOS Press, 1996.

[101] Stumpfe, D., Hu, H., and Bajorath, J. Evolving concept of activity cliffs. *ACS omega 4*, 11 (Sept. 2019), 14360–14368.

[102] Thieme, M., Hassan, M., Rupakheti, C., Thiagarajan, K. B., Pandey, A., and Liu, H. Topopool: An adaptive graph pooling layer for extracting molecular and protein substructures. In *NeurIPS 2023 Workshop on New Frontiers of AI for Drug Discovery and Development* (2023).

[103] Thieme, M., Rupakheti, C., Cusack, K., Pandey, A., and Liu, H. Neural atomic replacement via guided molecular adaptation.

[104] Thieme, M., Zhu, Y., and Liu, H. The graph learning attention mechanism: Learnable sparsification without heuristics, 2023.

[105] Unterthiner, T., Software, R., Mayr, A., Klambauer, G., Steijaert, M., Wegner, J. K., Johnson, J., Ceulemans, H., and Hochreiter, S. Deep learning as an opportunity in virtual screening. `https://ml.jku.at/publications/2014/NIPS2014f.pdf`. Accessed: 2024-1-23.

[106] V. Nagaslaev *et al.,*. Third integer resonance slow extraction scheme for a mu2e experiment at fermilab, 2010.

[107] V. Nagaslaev *et al.,*. Third Integer Resonance Slow Extraction Using RFKO at High Space Charge, 2011.

[108] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need.

[109] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks.

[110] von Humboldt, W. *Humboldt: 'On Language': On the Diversity of Human Language Construction and Its Influence on the Mental Development of the Human Species*. Cambridge University Press, Dec. 1999/1836.

[111] Wang, N.-N., Dong, J., Deng, Y.-H., Zhu, M.-F., Wen, M., Yao, Z.-J., Lu, A.-P., Wang, J.-B., and Cao, D.-S. ADME properties evaluation in drug discovery: Prediction of caco-2 cell permeability using a combination of NSGA-II and boosting. *Journal of chemical information and modeling 56*, 4 (Apr. 2016), 763–773.

[112] West, G. *Scale: The Universal Laws of Life, Growth, and Death in Organisms, Cities, and Companies*, reprint edition ed. Penguin Books, May 2018.

[113] Wu, T., Ren, H., Li, P., and Leskovec, J. Graph information bottleneck.

[114] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. A comprehensive survey on graph neural networks.

[115] Ye, Y., and Ji, S. Sparse graph attention networks. *IEEE transactions on knowledge and data engineering* (2021), 1–1.

[116] Ying, R., You, J., Morris, C., Ren, X., Hamilton, W. L., and Leskovec, J. Hierarchical graph representation learning with differentiable pooling. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Red Hook, NY, USA, Dec. 2018), NIPS'18, Curran Associates Inc., pp. 4805–4815.

[117] Yuan, H., Yu, H., Gui, S., and Ji, S. Explainability in graph neural networks: A taxonomic survey. *IEEE transactions on pattern analysis and machine intelligence 45*, 5 (May 2023), 5782–5799.

[118] Zhao, Z.-Y., Tomizuka, M., and Isaka, S. Fuzzy gain scheduling of PID controllers. In *[Proceedings 1992] The First IEEE Conference on Control Applications* (2003), IEEE.

[119] Zhu, J., Rossi, R. A., Rao, A., Mai, T., Lipka, N., Ahmed, N. K., and Koutra, D. Graph neural networks with heterophily.