GEOMETRIC DEEP LEARNING FOR TIME SERIES AND FOUNDATION MODELS

GEOMETRIC DEEP LEARNING FOR FINANCIAL TIME SERIES AND EFFICIENT FINE-TUNING OF FOUNDATION MODELS

BY

REZA ARABPOUR DAHOEI, B.Sc.

a Thesis

submitted to the Department of Computational Science and Engineering

and the School of Graduate Studies

of McMaster University

in partial fulfilment of the requirements

for the degree of

Master of Science

Master of Science (2025) McMaster University

(Department of Computational Science and Engineering) Hamilton, Ontario, Canada

TITLE:	Geometric Deep Learning For Financial Time Series and Efficient Fine-Tuning of Foundation Models
AUTHOR:	Reza Arabpour Dahoei B.Sc. Mathematics and Applications
SUPERVISOR:	Anastasis Kratsios
NUMBER OF PAGES:	xv, 102

Lay Abstract

This thesis presents two contributions at the intersection of artificial intelligence and mathematics.

First, I introduce a novel method for adapting large language models on widely available hardware. This approach recovers half of the performance lost when using an untuned base model instead of a GPU fine-tuned one, while running on a single laptop with minimal cost and energy consumption. It makes specialized models more accessible, preserves privacy by keeping data local, and promotes environmentally responsible computing.

Second, I develop a practical framework for working with history-dependent stochastic processes commonly used in quantitative finance. Such processes are often too large to compute efficiently. The method proposed here compresses them into a low-dimensional representation and then applies a computational model, enabling efficient simulation, estimation, and practical application.

Together, these contributions introduce novel algorithms capable of addressing real-world problems from fresh perspectives.

Abstract

This thesis presents two significant research contributions: one focuses on improving the adaptation of large language models (LLMs) using parameter-efficient fine-tuning (PEFT), and the other addresses the effective modelling of history-dependent stochastic processes—specifically Volterra processes, which are commonly applied in quantitative finance.

In the first part, I introduce a user-friendly adaptation pipeline that boosts the performance of a standard foundation model, bringing it much closer to a fully fine-tuned, task-specific version. Remarkably, it achieves this while using significantly less compute and memory, all while keeping data private. The pipeline leverages existing learnable low-rank adapters (LoRA) for known datasets and predicts adapter values for new datasets using this readily available information. Its main advantage is that it can run on a standard laptop without requiring GPU power, ensuring that data remains local. This method effectively closes about half of the performance gap between an untuned base model and a fully fine-tuned one, making specialized models more accessible to researchers, practitioners, and everyday users who lack expensive infrastructure or work with sensitive data on devices like smartphones.

The second part addresses a computational challenge in translating the non-Markovian Volterra process into a format suitable for computation. This translation is difficult because the data history dimension affecting the current state grows with the length of the path. I propose a two-step approach to make this process manageable: first, the Volterra process is mapped onto a simpler, lower-dimensional manifold; then, a geometric deep learning model—a “hypernetwork"—is applied, specifically designed for the manifold’s structure. We provide both mathematical and computational evidence demonstrating the model’s effectiveness and practicality (with proofs developed by co-authors available in the main paper), along with extensive testing of each parameter to validate our approach.

To my family—especially my father,

who passed away while I was studying abroad. I could not be by his side.

He is and always will be deeply missed.

Acknowledgements

First and foremost, I would like to express my heartfelt gratitude to my supervisor, Dr. Anastasis Kratsios, for his unwavering support, invaluable advice, insightful comments, and guidance throughout my master's program. It is no exaggeration to say I feel lucky to have him as my supervisor. His constant support and understanding have played a crucial role in all aspects of my life and my research in the last two years. I deeply believe many of the good things that happened to me simply could not have been possible without his support, and I will always be thankful for the opportunities and the trust he provided to me.

Next, I would also like to thank my family and my partner, who have been the strength in my heart and the reason in my mind to push through and not give up. Not a single second passed without thinking of them. Without their sacrifice, I could not be here.

Last but not least, I am profoundly grateful to Dr. Erin Clements, Hanadi Attar-Elbard, Emily Warnock, and Dr. Ben Bolker for their constant support, which made me feel like I truly belonged where I am. Also, McMaster University, the Vector Institute, Canadian Institute for Advanced Research, Bank of Montreal, Mitacs, the Province of Ontario, and the Government of Canada for providing me with the opportunity and resources to learn, enjoy, and contribute to the things I love.

Contents

List of Figures

List of Tables

Abbreviations

AI	Artificial intelligence
C.I.	Confidence interval
CPU	Central processing unit
CSE	Computational Science and Engineering
EM	Exact match
GB	Gigabyte
GDN	Geometric deep neural network
GPU	Graphics processing unit
HGN	Hypergeometric network
i.i.d.	Independent and identically distributed
JS	Jensen–Shannon divergence
IMSE	Intrinsic mean squared error
KAN	Kolmogorov-Arnold networks

KL	Kullback–Leibler divergence
LLM	Large language model
LoRA	Low-rank adaptation
MB	Megabyte
ML	Machine learning
MLP	Multi-layer perceptron
MMD	Maximum mean discrepancy
MoE	Mixture of experts
MSE	Mean squared error
PEFT	Parameter efficient fine tuning
RAM	Random access memory
ReLU	Rectified linear unit
RNN	Recurrent neural network
SDE	stochastic differential equation
SSD	Solid-state drive
std. dev.	Standard deviation
WD	Wasserstein distance

Declaration of Academic Achievement

These research contributions have been done during the terms of my master's degree:

1 **Reza Arabpour**, John Armstrong, Luca Galimberti, Anastasis Kratsios, Giulia Livieri. *"Low-dimensional approximations of the conditional law of discrete-time Volterra processes: a non-positive curvature approach"*. Under review at Analysis and Applications, May 2024.

- Contributions: Implemented and ran all the coding, numerical experiments, and ablation studies for the parameters. Introduced ideas to incorporate a new form of transfer learning and parallelization into the pipeline, as a result of which, the framework has become 60 times faster in practice and can be used to solve problems of much larger dimensions in a rational time.

2 **Reza Arabpour**, Haitz Sáez de Ocáriz Borde, Anastasis Kratsios. *"LoRA Fine-Tuning Without GPUs: A CPU-Efficient Meta-Generation Framework for LLMs"*. Accepted to International Conference on Machine Learning (ICML) 2025 Workshop on Efficient Systems for Foundation Models, July 2025.

- Contributions: Have been a part of the idea development and done all the coding and experiments during that phase, and for the final version of the framework as well. Have done one third of the writing and proposed multi-threading and parallelization into the pipeline, making the framework execution time reasonable for end users.

3 “ *Enhancing Predictive Power in Financial Markets: Leveraging Autoencoders for Time Series Embeddings in Capital Markets* ”. Ongoing six-month research project through Mitacs and in partnership with the Bank of Montreal (BMO).

  – Contributions: Created a comprehensive financial data generation system using the state-of-the-art mathematical finance models to generate a large amount of high-quality data useful for training and stress testing deep learning models. Currently working on extracting meaningful features using wavelet analysis techniques and the transformer architecture.

Chapter 1

Introduction

The recent wave of advancement of artificial intelligence (AI) and machine learning (ML) has revolutionized numerous domains by enabling the analysis and prediction of complex, high-dimensional data in different shapes and formats, from numbers to images/audio, and even text. At their core, neural networks are fascinating tools that are mathematical models invented in the last 80 years [1] and are continually evolving. These models are becoming the state of the art in various challenging computational tasks, such as natural language processing (NLP) [2] or computer vision [3]. Despite being under the same umbrella of deep learning, the underlying data that different models are being created for is inherently different, and this difference can change many things. A model that works well on a specific type of data might not be even close to the performance of a different one on another data [4]. These phenomena point to a fundamental question: the study of different data shapes along with the capabilities of different family of models or algorithms and their ability to represent or learn them. This study involves leveraging the geometric structures inherent in data to design more efficient and practical models, which is called geometric deep learning.

[5]

This thesis works on the same perspective but from a computational aspect. It explores two distinct yet structurally related contributions: the development of a framework for adapting large language models (LLMs) efficiently under realistic compute, cost, and privacy constraints; and the creation of practical approximations for conditional dynamics of history-dependent (non-Markovian) Volterra processes that arise in many areas like mathematical finance. The thread connecting these two research ideas is the task of approximating or predicting the internal parameters of a machine learning model that solves a desired problem using a higher-level model. In our case, our downstream problem solver models are primarily deep neural networks, and the higher-level model proposed is known as a hyper-geometric network (HGN), which will be the main focus of the third chapter and the core idea of the framework presented in the second chapter. A major reason for focusing on this point of view is its ability to address critical challenges in accessibility, scalability, and computational feasibility simultaneously, thereby bridging the theoretical insights from mathematics to real-world applications.

1.1 Motivation

A big portion of the AI revolution in the past few years has been a direct result of the invention of very large models with billions of parameters pre-trained on vast datasets and capable of general-purpose tasks. A pre-training has been done on the entire data on the internet, which embodies a huge portion of all the information generated by human beings in history. These large models, called foundation models [6], such as GPT [7], Llama [8], Gemini [9], and Mistral [10] models, exemplify this trend, demonstrating remarkable capabilities in language understanding and generation. However, despite their capability, they are rarely perfectly aligned with a new, domain-specific task. Their deployment for specialized tasks, thus, often requires fine-tuning, a process in which the model parameters are updated to align more with domain-specific data. As models and datasets scale up, full end-to-end fine-tuning of these large foundation models simply becomes impractical for many due to memory, compute, cost, and data-governance constraints. This crucial problem has been addressed by parameter-efficient fine-tuning (PEFT) [11] techniques, offering a strong trade-off between performance and efficiency by modifying only a small subset of parameters while keeping most of the model parameters frozen. Among these, the Low-Rank Adapter (LoRA) [12] approach has become standard due to its combined simplicity and surprisingly powerful effectiveness, often changing as few as 1-4% of parameters while maintaining +90% of performance [12]. However, most PEFT pipelines still assume GPUs to generate outputs and calculate the task-specific gradient to update parameters, which, for modern massive LLMs, makes LoRA fine-tuning still expensive and inaccessible to many. Thus, a pertinent question arises: Can one generate new low-rank adapters to fine-tune large language models on new tasks without the need for GPUs?

Shifting focus to the domain of quantitative finance, one of the main use cases of mathematics and machines in finance is the task of modelling, simulating, or approximating financial time series. These mathematical models are the heart of many downstream tasks, such as risk management and asset pricing, and affect the entire economy every day. There has been a long history of financial markets like stock markets, which date back to the 1500s; however, the mathematical modelling of financial markets arose centuries later in the 1900s when Louis Bachelier used probability theory in his thesis titled The Theory of Speculation to introduce a stochastic process today known as Brownian motion [13]. Stochastic processes are the mathematical tools that are being used to study and model the randomness in many fields and have been advanced to express more complex behaviours, such as history-dependent random processes. Not surprisingly, many financial time series exhibit long memory (history-dependence), unlike Markovian processes, where future states depend only on the current state. A family of models that are capable of modelling such processes is the Volterra processes. Volterra-type models, including rough-volatility families [14], incorporate the entire historical path, making them suitable for phenomena like volatility clustering or long-memory effects in asset prices. However, despite their theoretical promises, they come with practical drawbacks. Such processes are often too large to be feasible to compute efficiently when translated into the computational world, since the effective history dimension affecting the current state grows with the length of the path. In this situation, where traditional approaches, such as Monte Carlo methods or numerical solutions to stochastic differential equations (SDEs), become infeasible for long horizons or high-dimensional settings, a gap in theory and practice becomes more obvious: How can one use these promising non-Markovian models, like the Volterra, in practice to simulate or approximate with them efficiently?

Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 details the LoRA meta-generation framework, including preliminaries on datasets as probability distributions, the proposed pipelines (attentional, normalized, and neural), theoretical guarantees via propositions and theorems, and experimental results on downstream tasks. Chapter 3 focuses on the low-dimensional approximations for Volterra processes, introducing a novel geometric deep learning model, ablation studies on Volterra processes with varying key parameters to see the empirical results of the model's performance, and appendices with algorithms and additional figures. And finally, Chapter 4 concludes this thesis with a summary and future directions.

1.2 Contributions

Chapter 2

This chapter addresses the first question mentioned above: "Can one generate new low-rank adapters to fine-tune large language models on new tasks without the need for GPUs?" by introducing a zero-shot LoRA meta-generation procedure. The answer developed here is to directly predict the adapter parameters using only CPU, without any gradient calculation or training, thus no GPU at all; just a fast, on-device post-processing step that preserves privacy and reduces cost to a fraction while improving the performance of the foundation model by more than 150% on the given dataset. At the core, the framework combines the knowledge of a bank of pre-trained LoRAs using distributional alignment between the new dataset and previously fine-tuned adapter weights on known datasets that can be freely found on the internet. Finding this optimal combination or mixture of LoRAs is theoretically grounded in propositions and theorems that demonstrate, with high probability, a ReLU Multi-Layer Perceptron (MLP) architecture, designed to run efficiently on a CPU, can identify the optimal coefficients for combining existing LoRAs. These optimal LoRA mixture coefficients, representing a weighted sum of pre-trained LoRA parameters, are determined based on the given dataset alignment features. This process effectively minimizes the downstream task loss, which measures the model's error on new, specific tasks. Additionally, this work provides nearly optimal closed-form solutions through lightweight, neural network-free alternatives (e.g., the Attentional or Normalized approaches). Interestingly, experiments reveal that the neural network-free variants of the pipeline perform comparably to the theoretically near-optimal neural network solution (the MLP-based approach).

We conducted a series of comprehensive experiments [15] on more than 500 datasets [16] using a large language model (LLM), *Mistral-7B-v0.2* [10], that is powerful out of the box, but fine-tuning it for specific tasks often demands industry-grade GPU resources. In experiments, we tested three different versions of our pipeline across 502 dataset–adapter pairs and measured using the "Rouge-L" metric. The adaptation was done exclusively on the CPU, while GPUs are used only to evaluate the adapted models. The best lightweight pipeline reached Rouge-L of ≈ 0.52 on average, filling a large portion of the performance gap between the base model with a Rouge-L score of ≈ 0.19 and the fully GPU fine-tuned models (Rouge-L score of ≈ 0.75 on average). In other words, the pipeline "recovers roughly half" of the lost accuracy at nearly zero cost.

Chapter 3

This chapter addresses the second question mentioned, "How can one use these promising non-Markovian models, like the Volterra, in practice to simulate or approximate with them efficiently?" The solution proposed to this challenge, overcoming the curse of dimensionality, is presented as a two-step framework: first, project the (infinite-dimensional) conditional law of the process onto a low-dimensional statistical manifold of non-positive curvature. Second, on that manifold, apply a sequentially geometric deep learning (GDN) model with a hyper-geometric network, or hypernetwork in short, that updates internal parameters of these GDNs over time. In simpler words, we show that by compressing the law into a curved but controlled space and learning on that space with a hypernetwork, the conditional evolution of non-Markovian systems becomes computationally feasible and stable enough for real applications in finance. The hypernetwork can also be read as a gating mechanism in a mixture-of-experts view, letting the model adapt to evolving, non-stationary dynamics. The effectiveness and feasibility of the proposed model are mathematically and computationally proven, with mathematical proofs available in the main paper [17], and supported by an extensive ablation study of each parameter [18]. As can be seen with the extensive ablation studies examining sensitivity to drift (μ), randomness (λ), dimension (d), memory persistence (w), fluctuations (ς), and curvature at Chapter 3, the approach shows strong empirical results, with the hypernetwork generated GDNs for future time steps often tracking the trained GDNs on those time steps closely over time.

All together, the work presented in this thesis introduces practical algorithms for real-world problems through an efficient, novel, and geometry-aware lens. Allowing for the modelling of sophisticated processes within a reasonable time frame with an acceptable error bound, and prioritizing efficiency without sacrificing performance, paves the way for more accessible and sustainable AI practices. These works not only advance methodological frontiers but also underscore the value of interdisciplinary thinking, combining computational science, engineering, and mathematics to address current challenges in machine learning and finance.

Bibliography

[1] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943. URL https://doi.org/10.1007/BF02478259.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[3] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521,7553:436—-444, 2015. URL https://doi.org/10.1038/nature14539.

[4] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997. doi: 10.1109/4235.585893.

[5] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric

deep learning: Grids, groups, graphs, geodesics, and gauges, 2021. URL https://arxiv.org/abs/2104.13478.

[6] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui

Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models, 2022. URL `https://arxiv.org/abs/2108.07258`.

[7] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018. URL `https://api.semanticscholar.org/CorpusID:49313245`.

[8] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023. URL `https://arxiv.org/abs/2302.13971`.

[9] Gemini Team at Google et. al. Gemini: A family of highly capable multimodal models, 2025. URL `https://arxiv.org/abs/2312.11805`.

[10] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023. URL `https://arxiv.org/abs/2310.06825`.

[11] Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, and Fu Lee Wang. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment, 2023. URL `https://arxiv.org/abs/2312.12148`.

[12] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean

Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

[13] Introduction to mathematical finance. https://pi.math.cornell.edu/~mec/Summer2008/spulido/Math_Finance.html. Accessed: 2025-09-01.

[14] Christian Bayer, Peter K. Friz, Masaaki Fukasawa, Jim Gatheral, Antoine Jacquier, and Mathieu Rosenbaum. *Rough Volatility*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2023. doi: 10.1137/1.9781611977783. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611977783.

[15] Reza Arabpour Dahoei. Lora fine-tuning without gpus: A cpu-efficient meta-generation framework for llms. https://github.com/arabporr/LoRA_Fine-Tuning_Without_GPUs_A_CPU-Efficient_Meta-Generation_Framework_for_LLMs, 2025.

[16] Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Anjana Arunkumar, Arjun Ashok, Arut Selvan Dhanasekaran, Atharva Naik, David Stap, Eshaan Pathak, Giannis Karamanolakis, Haizhi Gary Lai, Ishan Purohit, Ishani Mondal, Jacob Anderson, Kirby Kuznia, Krima Doshi, Maitreya Patel, Kuntal Kumar Pal, Mehrad Moradshahi, Mihir Parmar, Mirali Purohit, Neeraj Varshney, Phani Rohitha Kaza, Pulkit Verma, Ravsehaj Singh Puri, Rushang Karia, Shailaja Keyur Sampat, Savan Doshi, Siddhartha Mishra, Sujan Reddy, Sumanta Patro, Tanay Dixit, Xudong Shen, Chitta Baral, Yejin Choi, Noah A. Smith, Hannaneh Hajishirzi, and Daniel Khashabi. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks, 2022. URL https://arxiv.org/abs/2204.07705.

[17] Reza Arabpour, John Armstrong, Luca Galimberti, Anastasis Kratsios, and Giulia Livieri. Low-dimensional approximations of the conditional law of volterra processes: a non-positive curvature approach, 2024. URL `https://arxiv.org/abs/2405.20094`.

[18] Reza Arabpour Dahoei. Hyper networks. `https://github.com/arabporr/HyperNetwork`, 2023.

Chapter 2

LoRA Fine-Tuning Without GPUs: A CPU-Efficient Meta-Generation Framework for LLMs

Abstract

Low-Rank Adapters (LoRAs) have transformed the fine-tuning of Large Language Models (LLMs) by enabling parameter-efficient updates. However, their widespread adoption remains limited by the reliance on GPU-based training. In this work, we propose a theoretically grounded approach to LoRA fine-tuning designed specifically for users with limited computational resources, particularly those restricted to standard laptop CPUs. Our method learns a meta-operator that maps any input dataset, represented as a probability distribution, to a set of LoRA weights by leveraging a large bank of pre-trained adapters for the Mistral-7B-Instruct-v0.2 model. Instead of performing new gradient-based updates, our pipeline constructs adapters via lightweight combinations of existing LoRAs directly on CPU. While the resulting adapters do not match the performance of GPU-trained counterparts, they consistently outperform the base Mistral model on downstream tasks, offering a practical and accessible alternative to traditional GPU-based fine-tuning.

2.1 Introduction

As models and datasets scale up, full fine-tuning becomes increasingly unrealistic for most practitioners. The largest foundation models—often built by tech giants with almost unlimited compute [1, 2, 3, 4, 5]—can have hundreds of billions of parameters, making traditional fine-tuning for individuals prohibitively expensive. Parameter-efficient fine-tuning (PEFT) methods [6, 7, 8, 9, 10] offer a workaround: instead of updating all weights, they tweak a small subset, slashing compute and storage costs while maintaining reasonable performance. Among these, the Low-Rank Adapter (LoRA) [11] approach has become standard due to combined simplicity and surprisingly powerful effectiveness. Nevertheless, for modern massive LLMs, LoRA fine-tuning can still be long and heavy. Thus, the following question arises:

Can one generate new low-rank adapters to fine-tune large language models on new tasks without the need for GPUs?

We address this concern by introducing a *zero-shot LoRA meta-generation procedure aimed at CPU-only users.* Our approach takes novel datasets, each potentially containing a variable number of instances, as input. It then outputs LoRA weights for a pre-trained LLM, where the prediction relies on a combination of instances from an existing bank of LoRAs [12]. Importantly, the way in which these combinations are performed is lightweight enough to be computable on a standard contemporary CPU in a few minutes (see Table 2.2 in Appendix 2.8.3), with no need for GPU clusters.

Main Contribution Our principled LoRA meta-generation pipeline provides lightweight, "cheap" LoRAs that approach the performance of GPU-fine-tuned models (which are often inaccessible to many) and outperform the base "non-finetuned" model.

These contributions are theoretically grounded in Proposition 1 and Theorem 1. Together, these demonstrate that, with high probability, a ReLU Multi-Layer Perceptron (MLP) architecture, designed to run efficiently on a CPU, can identify the optimal coefficients for combining existing LoRAs. These optimal LoRA mixture coefficients, as defined in (2.3.3) (representing a weighted sum of pre-trained LoRA parameters), are determined based on the given dataset alignment features. This process effectively minimizes the downstream task loss, which quantifies the model's error on new, specific tasks. Additionally, our work also provides nearly optimal closed-form solutions through lightweight, neural network-free alternatives (e.g., the Attentional or Normalized approaches). Our experiments reveal that the neural network-free variants of our pipeline perform comparably to the theoretically near-optimal neural network solution (the MLP-based approach).

Section 2.2 provides a discussion of related work concerning LoRA. We introduce the preliminaries for formalizing datasets as probability distributions in Section 2.3. Section 2.4 presents our LoRA generation pipelines. Their respective theoretical guarantees are later detailed in Section 2.5, and experimentally validated in Section 2.6.

2.2 Related Work

Since its introduction, the utility of LoRA [11] has expanded significantly beyond classical LLM post-training and language. It is now employed in diverse fields such as vision language models [13] and Vision Transformers [14]. LoRA has also proven valuable in image generative modeling for rapid Stable Diffusion fine-tuning and personalization [15, 16, 17, 18], and for score distillation [19], although more principled

LoRA-free methods have recently emerged [20]. Its application even extends to fine-tuning base models into reasoning models using reinforcement learning [21], and in the development of new adapters for graph neural networks and Graph Transformers [22].

Alongside this expanding applicability, numerous LoRA variants have emerged, often aiming to further reduce computational overhead. For instance, quantization offers a way to lower memory consumption both during training [23, 24, 25] and after [26]. The number of trainable parameters can also be reduced through adaptive rank allocation [27]. Further inspired by ideas around weight or projection reuse [28, 29], strategies to decrease trainable LoRA parameters include learning diagonal rescaling of frozen random B and A LoRA matrices (VeRA) [30], deriving B and A from the singular value decomposition of the pre-trained W_0 and optimizing a smaller matrix in the resulting space (SVDiff) [31], learning a linear combination of fixed random matrices (NOLA) [32], and fine-tuning with orthogonal matrices (BOFT) [33]. LoRAs have also been explored from a more theoretical viewpoint [34, 35, 36].

Our focus here is on LoRA generation on CPU, which none of the aforementioned works explore. We would like to reiterate that all our pipelines, *including those using artificial neural networks* can be trained solely using CPUs.

2.3 Preliminaries

Datasets as Probability Distributions To describe our pipeline, we first need a unified framework for datasets with a varying number of instances. As such, we fix dimensions $d, D \in \mathbb{N}_+$. Given our training datasets $D_1, \dots, D_N \subset \mathcal{X}$ for some (non-empty) compact input domain $\mathcal{X} \subseteq \mathbb{R}^{d+D}$ corresponding to one of N possible down-stream tasks $\mathcal{T}_1, \dots, \mathcal{T}_N$ which our Transformer model (Mistral-7B-Instruct-v0.2)

$f_\theta : \mathbb{R}^d \to \mathbb{R}^D$, whose parameters $\theta \in \mathbb{R}^p$ lie in a $p \gg 0$ dimensional Euclidean parameter space. Since the entries of each dataset are permutation-invariant, then, following the synthetic data generation literature, e.g. [37], it is natural to represent each dataset D_n as an empirical distribution (probability measure) via

$$P_{D_n} = \frac{1}{N_m} \sum_{(x,y)\in D_m} \delta_{(x,y)} \tag{2.3.1}$$

on the domain $\mathcal{X}$ where $N_m \overset{\text{def.}}{=} \# D_n$; i.e. $P_{D_n} = \sum_{m=1}^{N_m} w_m \delta_{(x_m,y_m)}$ with $w_m = 1/N_m$ for each $m = 1, \dots, N_m$.

The support of the measure P_{D_n}, namely, $\{(x_1, y_1), \dots, (x_m, y_m)\}$ represent instances in D_n and the weights $w_m \in [0, 1]$ sum to 1, i.e. w belongs to the N_m simplex $\Delta_{N_m} \overset{\text{def.}}{=} \{u \in [0, 1]^{N_m} : \sum_{i=1}^{N_m} u_i = 1\}$, and represent the relative frequency of instance of data-point in D_m. We denote the set of probability measures on $\mathcal{X}$ by $\mathcal{P}(\mathcal{X})$.

Pipeline Inputs and Distributional Alignment Scores We then choose a *data-similarity score* where $\rho : \mathcal{P}(\mathcal{X}) \times \mathcal{P}(\mathcal{X}) \to [0, \infty]$. For this, we choose a (dis)similarity metric between probability distributions (measures) on $\mathcal{X}$, e.g. an information-theoretic divergence such as Kullback Leibler (KL) divergence or a metric such as the 1-Wasserstein distance $\mathcal{W}_1$. This dissimilarity score then allows us to extract *alignment scores* between any new dataset D (encoded as a probability measure P_D on the data-domain $\mathcal{X}$) and every dataset $(D_n)_{n=1}^N$ in our database via align : $\mathcal{P}(\mathcal{X}) \to \Delta_N$

$$\text{align}(P_D) \overset{\text{def.}}{=} \text{Softmax}\left((\rho(P_D, P_{D_n}))_{n=1}^N\right). \tag{2.3.2}$$

Once the (softmax-normalized) alignment scores are computed, they are passed to a network. Here in our proof of concept, we use a simple MLP (trained on CPU), which yields a set of *mixture weights* $W_D \in \Delta_N$. These mixture weights are then used to combine the pre-trained LoRA weights $\theta_1, \dots, \theta_N$, from our database. Note that each LoRA weight θ_n was specialized for task $\mathcal{T}_n$ and pre-trained on dataset D_n. The output of our model is thus simply the mixture of LoRAs

$$D \mapsto P_D \mapsto \sum_{n=1}^{N} W_D \, \theta_n \tag{2.3.3}$$

and lies in the *convex hull* of the pre-trained LoRA weights $\theta_1, \dots, \theta_N$ in the parameter space $\mathbb{R}^p$. Therefore, we only need to learn (or compute, as we will see in Section 2.4) the mapping in (2.3.3). Based on this we are able to obtain LoRA weights with no fine-tuning, directly *out-of-the-box*.

2.4 LoRA Generation Pipelines for CPU

We now mathematically formalize our end-to-end cheap LoRA pipelines. Further details on how these were practically implemented can be found in Appendix 2.8.3. Our main theoretical guarantee (Theorem 1) is general enough to apply not only to LoRAs fed into transformers but also to nearly any mixture-of-expert-based parameter prediction pipeline.

2.4.1 Setup

Let $d, D \in \mathbb{N}_+$. Let $\ell : \mathbb{R}^D \times \mathbb{R}^D \to [0, \infty)$ be Lipschitz. Let $f : \mathbb{R}^p \times \mathbb{R}^d \to \mathbb{R}^D$ be a locally Lipschitz model, mapping the parameters $\theta \in \mathbb{R}^p$ and an input $x \in \mathbb{R}^d$ to an output $f_\theta(x) \in \mathbb{R}^D$. Also, We are given a pre-trained model $\theta_0 \in \mathbb{R}^p$.

Purely for simplicity, we consider the standardized data-domain $\mathcal{X} = [0,1]^{d+D}$. Following [38]. We henceforth fix a *task distribution* $\mathbb{P} \in \mathcal{P}(\mathcal{S})$ quantifying the probability of selecting any one dataset in $\mathcal{S}$ at random. We consider a metric space of datasets $\mathcal{D} \subseteq \mathcal{P}([0,1]^{d+D})$ metrized by ρ, where the topology generated by ρ is no coarser than the topology of convergence in distribution. We fix a $K \in \mathbb{N}_+$ datasets paired with "fine-tuned" model parameters $(D_1, \Delta\theta_1), \dots (D_K, \Delta\theta_K)$ in $\mathcal{D} \times \mathbb{R}^p$. Let

$$\operatorname{co}(\Delta\theta) \overset{\text{def.}}{=} \{\vartheta \in \mathbb{R}^p :\, (\exists w \in \Delta_K)\, \vartheta = \sum_{k=1}^{K} w_k \Delta\theta_K\}$$

where $\Delta_K \overset{\text{def.}}{=} \{w \in [0,1]^K :\, \sum_{k=1}^K w_k = 1\}$.

2.4.2 Very-Cheap LoRAs: Attentional Approach

Consider the following approach which maps any new incoming dataset D to the following mixture of LoRAs

$$\mathcal{C}_{\text{Att}}(D) \overset{\text{def.}}{=} \underbrace{[\operatorname{softmin} \circ \operatorname{align}(D)]^\top}_{\text{LoRA Alignment Scores}} \underbrace{(\Delta\theta_1, \dots, \Delta\theta_K)}_{\text{Pre-Trained LoRAs}} \tag{2.4.1}$$

We refer to the pipeline in (2.4.1) as our *attentional approach* since the dataset $D_1, \dots, D_K$ play a similar role to the *keys* in attention mechanisms [39]. The LoRA alignment scores in (2.4.1) are analogous to contextual *alignment scores*, and the pre-trained LoRA parameters play a similar role to the *value matrices* in [39]. The softmin is used instead of a softmax since maximal distance alignment happens when two datasets have a distance of 0 from one another, not some arbitrarily large number. We examine a *normalized* version of distance vector (2.4.1) in our experiments, see

Appendix 2.8.3 for details.

2.4.3 Cheap Nearly-Optimal LoRAs: Neural Approach

Our neural approach injects non-linear flexibility into how the distances are mapped to the LoRA alignment scores in (2.4.1) using a deep learning model $\mathcal{C} : \mathcal{D} \to \operatorname{co}(\Delta\theta)$; in this paper, this will always be an MLP. This allows our cheap LoRA approach to learn how to detect and align complicated non-linear alignments between the new dataset and those defining each pre-trained task. This *neural* approach thus sends any dataset D to the following mixture of LoRAs

$$\mathcal{C}(D) \overset{\text{def.}}{=} \underbrace{[\operatorname{softmin} \circ \hat{f} \circ \operatorname{align}(D)]^\top}_{\text{Neural-LoRA Alignment Scores}} \underbrace{(\Delta\theta_1, \dots, \Delta\theta_K)}_{\text{Pre-Trained LoRAs}} \tag{2.4.2}$$

where $\hat{f} : \mathbb{R}^K \to \mathbb{R}^K$ is an MLP with activation function ς, and we write $\operatorname{align}(D)$ in place of $\operatorname{align}(P_D)$ understanding the correspondence $D \to P_D$ as implicit.

2.5 Theoretical Guarantees

We now provide guarantees on the optimality of both our main approaches. We also demonstrate the existence of an oracle optimizer, yielding the best possible LoRA if the user had access to complete information on the task distribution.

2.5.1 Attentional Approach

Our cheapest out-of-the-box LoRA pipeline (2.4.1) is optimal in a PAC-Bayesian sense of [40].

Proposition 1 (Existence: Optimal Oracles for Fine-Tuning)**.** *For every $K \in \mathbb{N}_+$ and $\{(D_k, \Delta\theta_k)\}_{k=1}^K \subset \mathcal{D} \times \mathbb{R}^p$ with each D_k finite and non-empty. For every $\alpha > 0$ and each dataset $D \in \mathcal{D}$, the LoRA Alignment Scores in* (2.4.1) *satisfy*

$$\underbrace{\operatorname{softmin} \circ \operatorname{align}(D)}_{\textit{LoRA Alignment Scores}} \in \operatorname{argmin}_{w \in \Delta_k} \underbrace{\frac{1}{K} \sum_{k=1}^K w_k \, \rho(D, D_k)}_{\textit{Dataset Alignment}} + \underbrace{\frac{1}{\alpha} \sum_{k=1}^K w_k \log(w_k)}_{\textit{Entropic Penalty}}$$

Proof. See Appendix 2.8.2. □

2.5.2 Neural Approach

The attentional pipeline, in (2.4.1), only checks for the *alignment* of a dataset with the datasets previously used for training the adapters in the bank. In contrast our neural approach, in (2.4.2), optimizes for the *downstream performance* of the predicted mixture of LoRA experts. Surprisingly, at least theoretically, one only needs a small MLP between the distance vector and softmin normalization layers to perform this out-of-the-box downstream (near) optimal LoRA generation. Our first guarantee for the neural approach demonstrates the existence of a map, i.e., an oracle predictor, which returns the best possible downstream optimization.

Proposition 2 (Existence: Optimal Oracles for Fine-Tuning)**.** *For every dataset $D \in \mathcal{D}$ there exists an oracle parameter $\vartheta^\star \in \operatorname{co}(\Delta\theta)$ satisfying*

$$\underbrace{\mathbb{E}_{(X,Y)\sim\mathcal{D}}\Big[\ell(f_{\theta+\vartheta^\star}(X), Y)\Big]}_{\textit{Oracle Error}} = \underbrace{\inf_{\Delta\theta \in \operatorname{co}(\Delta\theta)} \mathbb{E}_{(X,Y)\sim\mathcal{D}}\Big[\ell(f_{\theta+\Delta\theta}(X), Y)\Big]}_{\textit{Optimal Error}}.$$

Proof. See Appendix 2.8.2. □

Our next and main results show that our pipeline can implement the optimal downstream mixture of LoRA predictors to achieve precision. Our result only relies on one structural regularity condition on our data, guaranteeing that: the *inverse problem* of recording a dataset/measure from its distance measurements to the available datasets/measures is possible. Effectively, this means that the metric dimension, in the graph-theoretic sense (see [41] for details), of the space $\mathcal{D}$ is exactly K.

Assumption 1 (Well-Posed Inverse Problem)**.** *Let* $(\mathcal{D}, \rho)$ *be compact and suppose that* ρ *metrizes the weak topology (convergence in distribution) on* $\mathcal{D}$*. We require that: the map* $\operatorname{align} : \mathcal{D} \to [0, \infty)^K$ *injectively maps any* $D \in \mathcal{D}$ *to*

$$\operatorname{align}(D) \overset{\text{def.}}{=} \big(\rho(D, D_k)\big)_{k=1}^{K}.$$

Theorem 1 (ε-Optimal Cheap Fine-tuning)**.** *Let* $\varsigma : \mathbb{R} \to \mathbb{R}$ *be a Lipschitz activation function which is differentiable with non-zero derivative on at least one point. For every* $0 < \varepsilon \leq 1$*, there is a MLP* $\mathcal{C} : \mathbb{R}^K \to \mathbb{R}^K$ *with activation function* ς *such that the* ϵ*-optimal selection property:*

$$\underbrace{\mathbb{E}_{(X,Y)\sim\mathcal{D}}\Big[\ell(f_{\theta+\mathcal{C}(D)}(X), Y)\Big]}_{\textit{Cheap Fine-Tuning}} \leq \underbrace{\inf_{\Delta\theta \in \operatorname{co}(\Delta\theta)} \mathbb{E}_{(X,Y)\sim\mathcal{D}}\Big[\ell(f_{\theta+\Delta\theta}(X), Y)\Big]}_{\textit{Fine-Turning Oracle}} + \varepsilon$$

holds with $\mathbb{P}$*-probability at-least* $1 - \varepsilon$*.*

Proof. See Appendix 2.8.2. $\square$

2.6 Experimental Results

A comprehensive evaluation was conducted to assess the performance of three distinct approaches (Attentional, Normalized, and Neural) in conjunction with four established distance metrics (or divergences): Wasserstein Distance (WD), Kullback–Leibler (KL) divergence, Jensen-Shannon (JS) divergence, and Maximum Mean Discrepancy (MMD). This evaluation aimed to systematically compare the outputs generated by each combination of approach and metric. The primary evaluation criterion for the quality of the generated adapters was Rouge-L, a metric ranging from 0 to 1 that quantifies similarity based on the overlap of the longest common subsequences between generated and reference outputs [42]. We also include Exact Match (EM) results in Appendix 2.8.4.

Our experimental setup used the Mistral-7B-Instruct-v0.2 model [43] and a dataset comprising 502 English dataset-adapter pairs sourced from the Lots-of-LoRAs HuggingFace repository [12]. Further technical details regarding the implementation are provided in Appendix 2.8.3.

Our experimental setup highlighted a key distinction in resource usage: the actual computation and adaptation of the LoRA adapters were performed exclusively on the CPU. GPUs, however, were essential only for the evaluation phase. This is because each adapted LLM, after being modified by our pipeline, needed to be loaded onto a GPU to generate outputs on its respective test set. To thoroughly assess the performance of each approach-distance (or divergence) metric pairing, we executed the entire pipeline twelve times for each of the 502 datasets. This exhaustive process covered every unique combination of approaches and distance metrics. Following the generation of outputs, the Rouge-L score was calculated for the test set of each dataset, and the reported values reflect the average of these scores across all runs.

2.6.1 Performance Comparison and Analysis

Our work is benchmarked against two key performance indicators. First, the performance of the base foundation model without any fine-tuning, representing a scenario where an end-user with limited computational resources applies a foundation model to a new dataset: this yielded an average and standard deviation Rouge-L score of 0.192 ± 0.181. Second, we compare against the performance of a GPU-fine-tuned model, achieved without hardware limitations, which obtained an Rouge-L score of 0.746 ± 0.265. Table 2.1 presents the average and standard deviation of Rouge-L performance for all approaches across the four distance (or divergence) metrics on the downstream task.

The JS divergence-based Normalized approach achieved the highest score, with an average Rouge-L of 0.520. This represents an improvement of 0.328 over the base model's score of 0.192. It is worth mentioning that even our Attentional approach, despite its simplicity, significantly outperforms the base foundation model across all distance metrics. Interestingly, the neural approach does not seem to justify the additional computational cost, as its performance improvement over the Attentional and Normalized approaches is generally minimal or even worse.

Table 2.1: Performance of our cheap LoRA pipelines.

Approach	WD	KL	JS	MMD
Attentional (std. dev.)	0.426(±0.290)	0.501(±0.272)	0.486(±0.270)	0.486(±0.270)
Normalized (std. dev.)	0.495(±0.267)	0.488(±0.269)	**0.520**(±0.277)	0.497(±0.269)
Neural (std. dev.)	0.494(±0.265)	0.482(±0.268)	0.484(±0.272)	0.493(±0.270)

2.7 Conclusion

In conclusion, our work presents a practical, simple, and theoretically supported pipeline for generating LoRAs suitable for fine-tuning LLMs using only a CPU. This pipeline significantly reduces the typically required computational demands, making fine-tuning accessible even to users with limited hardware resources or on edge devices with privacy constraints.

We proved the existence of a lightweight ReLU MLP backbone, runnable on a CPU, that can reliably approximate optimal LoRA adapter weights and biases, thereby effectively minimizing downstream task loss in Theorem 1. Surprisingly, the simplest versions of our pipeline (Attentional and Normalized) achieved performance matching that of the MLP backbone version, further demonstrating the efficiency and power of our approach.

Our experiments, using the Mistral-7B-Instruct-v0.2 model on 502 diverse datasets, demonstrate substantial improvements over the baseline model, with the best configuration achieving a 0.328 increase in performance (Rouge-L score) over the base model, bridging more than half of the performance gap between the base model and the GPU fine-tuned reference. While our CPU-generated adapters do not yet match the performance of GPU-trained adapters, they provide a compelling alternative in resource-limited settings.

Future work could explore the applicability of these approaches to other language models as more LoRA adapter banks become open-source, as well as to tasks beyond NLP. Likewise, it would be of interest to better understand how many LoRA adapters would be required to generate new, high-quality adapters—that is, what size of bank is necessary? We expect this to depend on the task, data modalities, and possibly even the model architecture. Finally, our method could also potentially be used for LoRA initialization (pre-heating) before fine-tuning on a GPU.

Bibliography

[1] Hugo Touvron et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[2] OpenAI. Gpt-4 technical report, 2023. Available at `https://openai.com/research/gpt-4`.

[3] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. Qwen technical report, 2023. URL `https://arxiv.org/abs/2309.16609`.

[4] Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li,

Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL `https://arxiv.org/abs/2412.15115`.

[5] DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang

Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. Deepseek-v3 technical report, 2025. URL `https://arxiv.org/abs/2412.19437`.

[6] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=0RDcd5Axok`.

[7] Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. AdapterHub: A framework for adapting transformers. In Qun Liu and David Schlangen, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 46–54, Online, October 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.7. URL

https://aclanthology.org/2020.emnlp-demos.7.

[8] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, Jing Yi, Weilin Zhao, Xiaozhi Wang, Zhiyuan Liu, Hai-Tao Zheng, Jianfei Chen, Yang Liu, Jie Tang, Juanzi Li, and Maosong Sun. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models, 2022. URL https://arxiv.org/abs/2203.06904.

[9] Bruce X.B. Yu, Jianlong Chang, Lingbo Liu, Qi Tian, and Chang Wen Chen. Towards a unified view on visual parameter-efficient transfer learning. *arXiv preprint arXiv:2210.00788*, 2022.

[10] Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. Parameter-efficient fine-tuning for large models: A comprehensive survey, 2024. URL https://arxiv.org/abs/2403.14608.

[11] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. 2021. URL https://arxiv.org/abs/2106.09685.

[12] Rickard Brüel Gabrielsson, Jiacheng Zhu, Onkar Bhardwaj, Leshem Choshen, Kristjan Greenewald, Mikhail Yurochkin, and Justin Solomon. Compress then serve: Serving thousands of loRA adapters with little overhead, 2024. URL https://openreview.net/forum?id=hHNVn4hFPk.

[13] Xin Li, Dongze Lian, Zhihe Lu, Jiawang Bai, Zhibo Chen, and Xinchao Wang.

Graphadapter: Tuning vision-language models with dual knowledge graph. In *Advances in Neural Information Processing Systems*, 2023.

[14] Wei Dong, Dawei Yan, Zhijun Lin, and Peng Wang. Efficient adaptation of large vision transformer via adapter re-composing. In *Advances in Neural Information Processing Systems*, 2023.

[15] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022. URL `https://arxiv.org/abs/2112.10752`.

[16] Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H Bermano, Gal Chechik, and Daniel Cohen-Or. An image is worth one word: Personalizing text-to-image generation using textual inversion. *arXiv preprint arXiv:2208.01618*, 2022.

[17] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. *arXiv preprint arXiv:2208.12242*, 2022.

[18] Daniel Roich, Ron Mokady, Amit H Bermano, and Daniel Cohen-Or. Pivotal tuning for latent-based editing of real images. *ACM Transactions on Graphics (TOG)*, 42(1):1–13, 2022.

[19] Zhengyi Wang, Cheng Lu, Yikai Wang, Fan Bao, Chongxuan Li, Hang Su, and Jun Zhu. Prolificdreamer: High-fidelity and diverse text-to-3d generation with variational score distillation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=ppJuFSOAnM`.

[20] Artem Lukoianov, Haitz Sáez de Ocáriz Borde, Kristjan Greenewald, Vitor Campagnolo Guizilini, Timur Bagautdinov, Vincent Sitzmann, and Justin Solomon. Score distillation via reparametrized DDIM. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL `https://openreview.net/forum?id=4DcpFagQ9e`.

[21] Shangshang Wang, Julian Asilis, Ömer Faruk Akgül, Enes Burak Bilgin, Ollie Liu, and Willie Neiswanger. Tina: Tiny reasoning models via lora, 2025. URL `https://arxiv.org/abs/2504.15777`.

[22] Pantelis Papageorgiou, Haitz Sáez de Ocáriz Borde, Anastasis Kratsios, and Michael M. Bronstein. Graph low-rank adapters of high regularity for graph neural networks and graph transformers. In *First Workshop on Scalable Optimization for Efficient and Adaptive Foundation Models*, 2025. URL `https://openreview.net/forum?id=gxhZj6uvFC`.

[23] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference, 2021.

[24] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *ArXiv*, abs/2305.14314, 2023. URL `https://api.semanticscholar.org/CorpusID:258841328`.

[25] Han Guo, Philip Greengard, Eric P. Xing, and Yoon Kim. Lq-lora: Low-rank plus quantized matrix decomposition for efficient language model finetuning, 2024.

[26] Prateek Yadav, Leshem Choshen, Colin Raffel, and Mohit Bansal. Compeft:

Compression for communicating parameter efficient updates via sparsification and quantization. *arXiv preprint arXiv:2311.13171*, 2023.

[27] Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning, 2023.

[28] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks, 2018.

[29] Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. What's hidden in a randomly weighted neural network? In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2020. doi: 10.1109/cvpr42600.2020.01191. URL `http://dx.doi.org/10.1109/CVPR42600.2020.01191`.

[30] Dawid J. Kopiczko, Tijmen Blankevoort, and Yuki M. Asano. Vera: Vector-based random matrix adaptation, 2024.

[31] Ligong Han, Yinxiao Li, Han Zhang, Peyman Milanfar, Dimitris Metaxas, and Feng Yang. Svdiff: Compact parameter space for diffusion fine-tuning. *arXiv preprint arXiv:2303.11305*, 2023.

[32] Soroush Abbasi Koohpayegani, KL Navaneet, Parsa Nooralinejad, Soheil Kolouri, and Hamed Pirsiavash. Nola: Networks as linear combination of low rank random basis, 2023.

[33] Weiyang Liu, Zeju Qiu, Yao Feng, Yuliang Xiu, Yuxuan Xue, Longhui Yu, Haiwen Feng, Zhen Liu, Juyeon Heo, Songyou Peng, Yandong Wen, Michael J. Black,

Adrian Weller, and Bernhard Schölkopf. Parameter-efficient orthogonal finetuning via butterfly factorization. In *ICLR*, 2024.

[34] Yuchen Zeng and Kangwook Lee. The expressive power of low-rank adaptation. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=likXVjmh3E`.

[35] Jiacheng Zhu, Kristjan Greenewald, Kimia Nadjahi, Haitz Sáez De Ocáriz Borde, Rickard Brüel Gabrielsson, Leshem Choshen, Marzyeh Ghassemi, Mikhail Yurochkin, and Justin Solomon. Asymmetry in low-rank adapters of foundation models. In *Proceedings of the 41st International Conference on Machine Learning*, pages 62369–62385, 2024.

[36] Anastasis Kratsios, Tin Sum Cheng, Aurelien Lucchi, and Haitz Sáez de Ocáriz Borde. Sharp generalization bounds for foundation models with asymmetric randomized low-rank adapters, 2025. URL `https://arxiv.org/abs/2506.14530`.

[37] Behnoosh Zamanlooy, Mario Diaz, and Shahab Asoodeh. Locally private sampling with public data. *arXiv preprint arXiv:2411.08791*, 2024.

[38] Jonas Rothfuss, Martin Josifoski, Vincent Fortuin, and Andreas Krause. Scalable pac-bayesian meta-learning via the pac-optimal hyper-posterior: from theory to practice. *The Journal of Machine Learning Research*, 24(1):18474–18535, 2023.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[40] Pierre Alquier, James Ridgway, and Nicolas Chopin. On the properties of variational approximations of gibbs posteriors. *Journal of Machine Learning Research*, 17(236):1–41, 2016.

[41] Richard C Tillquist, Rafael M Frongillo, and Manuel E Lladser. Getting the lay of the land in discrete space: A survey of metric dimension and its applications. *SIAM Review*, 65(4):919–962, 2023.

[42] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/W04-1013`.

[43] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b. 2023. URL `https://arxiv.org/abs/2310.06825`.

[44] Renjie Wang, Cody Hyndman, and Anastasis Kratsios. The entropic measure transform. *Canad. J. Statist.*, 48(1):97–129, 2020. ISSN 0319-5724,1708-945X. doi: 10.1002/cjs.11537. URL `https://doi.org/10.1002/cjs.11537`.

[45] Charalambos D. Aliprantis and Kim C. Border. *Infinite dimensional analysis*. Springer, Berlin, third edition, 2006. ISBN 978-3-540-32696-0; 3-540-32696-0. A hitchhiker's guide.

[46] James R. Munkres. *Topology*. Prentice Hall, Inc., Upper Saddle River, NJ, second edition, 2000. ISBN 0-13-181629-2.

[47] Olav Kallenberg. *Foundations of modern probability*, volume 99 of *Probability Theory and Stochastic Modelling*. Springer, Cham, third edition, 2021. ISBN 978-3-030-61871-1; 978-3-030-61870-4. doi: 10.1007/978-3-030-61871-1. URL https://doi.org/10.1007/978-3-030-61871-1.

[48] Alexander S. Kechris. *Classical descriptive set theory*, volume 156 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1995. ISBN 0-387-94374-9. doi: 10.1007/978-1-4612-4190-4. URL https://doi.org/10.1007/978-1-4612-4190-4.

[49] Achim Klenke. *Probability theory—a comprehensive course*. Universitext. Springer, Cham, third edition, 2020. ISBN 978-3-030-56402-5; 978-3-030-56401-8. doi: 10.1007/978-3-030-56402-5. URL https://doi.org/10.1007/978-3-030-56402-5.

[50] J. Dugundji. An extension of Tietze's theorem. *Pacific J. Math.*, 1:353–367, 1951. ISSN 0030-8730,1945-5844. URL http://projecteuclid.org/euclid.pjm/1103052106.

[51] Anastasis Kratsios and Léonie Papon. Universal approximation theorems for differentiable geometric deep learning. *J. Mach. Learn. Res.*, 23:Paper No. [196], 73, 2022. ISSN 1532-4435,1533-7928.

[52] Patrick Kidger and Terry Lyons. Universal approximation with deep narrow networks. In *Conference on learning theory*, pages 2306–2327. PMLR, 2020.

[53] Rickard Brüel Gabrielsson. Lots of loras. `https://huggingface.co/Lots-of-LoRAs`, 2025.

[54] Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Anjana Arunkumar, Arjun Ashok, Arut Selvan Dhanasekaran, Atharva Naik, David Stap, Eshaan Pathak, Giannis Karamanolakis, Haizhi Gary Lai, Ishan Purohit, Ishani Mondal, Jacob Anderson, Kirby Kuznia, Krima Doshi, Maitreya Patel, Kuntal Kumar Pal, Mehrad Moradshahi, Mihir Parmar, Mirali Purohit, Neeraj Varshney, Phani Rohitha Kaza, Pulkit Verma, Ravsehaj Singh Puri, Rushang Karia, Shailaja Keyur Sampat, Savan Doshi, Siddhartha Mishra, Sujan Reddy, Sumanta Patro, Tanay Dixit, Xudong Shen, Chitta Baral, Yejin Choi, Noah A. Smith, Hannaneh Hajishirzi, and Daniel Khashabi. Supernaturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. 2022. URL `https://arxiv.org/abs/2204.07705`.

2.8 Appendix

Funding

A. Kratsios and R. Arabpour acknowledge financial support from an NSERC Discovery Grant No. RGPIN-2023-04482 and No. DGECR-2023-00230. They also acknowledge that resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute[1].

[1]https://vectorinstitute.ai/partnerships/current-partners/

2.8.1 Additional Background

This appendix presents any additional background required for the formulation of our main results, proofs of our guarantees, and additional experimental details.

Foundation Model Fine-tuning and Attention Layers

In modern LLMs, fine-tuning all parameters can be computationally expensive and memory-intensive. LoRA [11] provides an efficient alternative by introducing low-rank updates to pre-trained weight matrices, particularly focusing on attention layers in transformer-based models. Given query Q, key K, and value V matrices, the standard attention mechanism computes the attention scores

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V, \tag{2.8.1}$$

where d_k is the dimension of the keys and queries.

Low-Rank Adapter (LoRA) Fine-tuning

In large transformers, these weight matrices dominate the parameter count, making them an ideal target for LoRA's efficient fine-tuning. By applying low-rank updates to these matrices, LoRA achieves significant savings in memory and computation without retraining the entire model. Consider a pre-trained weight matrix $W_0 \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, typically representing the projection matrices in the attention mechanism. Instead of updating the entire matrix, LoRA modifies the weights by adding a low-rank perturbation:

$$W = W_0 + \Delta W, \tag{2.8.2}$$

where ΔW is constrained to have $\operatorname{rank}(\Delta W) = r \leq \min(d_{\text{out}}, d_{\text{in}})$. To efficiently parameterize ΔW, LoRA decomposes it as:

$$\Delta W = BA, \tag{2.8.3}$$

where $B \in \mathbb{R}^{d_{\text{out}} \times r}$ and $A \in \mathbb{R}^{r \times d_{\text{in}}}$. During fine-tuning, the original weights W_0 remain frozen, and only the parameters in A and B are optimized. In traditional full fine-tuning, the entire weight matrix is updated, requiring $d_{\text{out}} \cdot d_{\text{in}}$ trainable parameters. In contrast, the LoRA decomposition introduces only $r \cdot (d_{\text{in}} + d_{\text{out}})$ trainable parameters, which is more efficient when $r \ll \min(d_{\text{out}}, d_{\text{in}})$.

Distance Measures between Probability Distributions

We remind the reader of the necessary definitions required in formulating the distance between datasets, when interpreted as finitely supported probability measures (distributions). Given two probability distributions P and Q defined on a separable and complete (Polish) metric space $\mathcal{X}$ equipped with its Borel σ-algebra and metrized by a metric $\rho : \mathcal{X}^2 \to [0, \infty)$, these measures of discrepancy (or divergences) are defined as follows:

Wasserstein Distance (WD). For distributions P and Q with cumulative distribution functions F_P and F_Q respectively, the 1-Wasserstein distance is defined as:

$$W_1(P, Q) \overset{\text{def.}}{=} \inf_{\pi} \int_{(x,y)\in\mathcal{X}^2} \rho(x, y)\, \pi(d(x, y)) \tag{2.8.4}$$

where the infimum is taken over all joint probability distributions π on $\mathcal{X} \times \mathcal{X}$ (with the product σ-algebra) whose marginals are P and Q.

Kullback–Leibler Divergence (KL). The KL divergence measures the relative entropy between two distributions:

$$D_{KL}(P \parallel Q) \overset{\text{def.}}{=} \begin{cases} \int \log_{x \in \mathcal{X}} \frac{dP}{dQ}(x) P(dx) & : \text{if } P \ll Q \\ \infty & : \text{if } P \not\ll Q \end{cases} \tag{2.8.5}$$

where $P \ll Q$ denotes the absolute continuity of P with respect to Q, and $\frac{dP}{dQ}$ denotes the Radon-Nikodym derivative, or probability density, of P with respect to Q.

Jensen–Shannon Divergence (JS). The JS divergence is a symmetrized version of KL divergence:

$$D_{JS}(P \parallel Q) = \frac{1}{2} D_{KL}(P \parallel M) + \frac{1}{2} D_{KL}(Q \parallel M) \tag{2.8.6}$$

where $M = \frac{1}{2}(P + Q)$.

Maximum Mean Discrepancy (MMD). If $\mathcal{H}$ is a Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}$ of functions over $\mathcal{X}$ with reproducing kernel function k; then we may also define the MMD between P and Q by

$$\text{MMD}^2(P, Q) = \mathbb{E}_{x,x' \sim P}[k(x, x')] - 2\mathbb{E}_{x \sim P, y \sim Q}[k(x, y)] + \mathbb{E}_{y,y' \sim Q}[k(y, y')]. \tag{2.8.7}$$

If $\mathcal{X}$ is $\mathbb{R}^d$ and $\mathcal{H} = L^2_\gamma(\mathbb{R}^d)$ for the standard Gaussian measure $\gamma \sim N(0, I_d)$, then k is often chosen to be a Gaussian kernel, i.e., $k(x,y) = \exp(-\frac{\|x-y\|^2}{2\sigma^2})$.

2.8.2 Proofs

We now prove the main result of our paper. We begin with the proof of our simplest result, Proposition 1.

Proof of Proposition 1

For any dataset D, note that $\operatorname{argmin}_{w\in\Delta_k} \frac{1}{K}\sum_{k=1}^K w_k\,\rho(D,D_k) + \frac{1}{\alpha}\sum_{k=1}^K w_k \log(w_k)$. Now, by [44, Proposition 1] its unique minimizer, which we denote by $w_D^\star$, is given by

$$w_D^\star = \frac{e^{-\rho(D,D_k)}}{\sum_{i=1}^K e^{-\rho(D,D_i)}} = \operatorname{softmin}\circ \operatorname{align}(D). \qquad \square$$

Simultaneous Proof of Theorem 1 and Proposition 2

We now derive Proposition 2 and Theorem 1 within the same proof, as their derivation is most naturally undertaken together.

Step 1 - Existence of a Measurable Selector

We will first set up the Measurable Maximum Theorem, see e.g. [45, Theorem 18.19]. Consider the constant correspondence

$$\begin{aligned} \varphi : \mathcal{D} &\twoheadrightarrow 2^{\mathbb{R}^p} \\ D &\mapsto \operatorname{co}(\Delta\theta). \end{aligned}$$

Since $\operatorname{co}(\Delta\theta)$ is a closed, non-empty, and bounded set then the Heine-Borel theorem implies that $\operatorname{co}(\Delta\theta)$ is compact. Whence φ is a correspondence with non-empty, compact, and convex values. Let $B \subseteq \mathcal{D}$ be a Borel set, then

$$\varphi(B) \overset{\text{def.}}{=} \bigcup_{D\in B} \varphi(D) = \bigcup_{D\in B} \operatorname{co}(\Delta\theta) = \operatorname{co}(\Delta\theta). \tag{2.8.8}$$

Since $\operatorname{co}(\Delta\theta)$ is closed it is Borel; whence, φ is not only a weakly measurable correspondence [45, 18.1 Definition (1)] but it is also a Borel measurable correspondence [45, 18.1 Definition (3)]. Thus, the correspondence φ satisfies the requirements of [45, Theorem 18.19].

Next, consider the objective function

$$\begin{aligned} L : \mathcal{D} \times \mathbb{R}^p &\to [0,\infty) \\ (D,B) &\mapsto \mathbb{E}_{(X,Y)\sim D}\big[\ell(f_{\theta+\vartheta}(X),Y)\big]. \end{aligned} \tag{2.8.9}$$

We will show that L is a Carathéodory function by showing it is continuous. Since $\mathcal{D} \times \mathbb{R}^p$ is a product (topological) space, then [46, Theorem 19.6] guarantees that f is continuous if and only if each of its component functions is continuous; we show the latter.

Fix $D \in \mathcal{D}$. Since $\mathcal{C}$ and the softmax function are locally Lipschitz, and since ℓ is Lipschitz, then their composition is locally Lipschitz. Whence, for each $(x,y) \in [0,1]^{d+D}$ the map

$$\Lambda_{x,y} : \operatorname{co}(\Delta\theta)) \ni \vartheta \mapsto \ell(\mathcal{C}_{\theta+\vartheta}(x),y))$$

is λ-Lipschitz, for some $\lambda \geq 0$. By Jensen's inequality we have: for each $\vartheta_1, \vartheta_2 \in \operatorname{co}(\Delta\theta)$

$$\begin{aligned}
&\Big|\mathbb{E}_{(X,Y)\sim\mathcal{D}}\big[\Lambda_{X,Y}(\vartheta_1)\big] - \mathbb{E}_{(X,Y)\sim\mathcal{D}}\big[\Lambda_{X,Y}(\vartheta_2)\big]\Big| \\
&= \Big|\mathbb{E}_{(X,Y)\sim\mathcal{D}}\big[\Lambda_{X,Y}(\vartheta_1) - \Lambda_{X,Y}(\vartheta_2)\Big| \\
&\leq \mathbb{E}_{(X,Y)\sim\mathcal{D}}\Big[\big|\Lambda_{X,Y}(\vartheta_1) - \Lambda_{X,Y}(\vartheta_2)\big|\Big] \\
&\leq \lambda\mathbb{E}_{(X,Y)\sim\mathcal{D}}\Big[\big\|\vartheta_1 - \vartheta_2\big\|\Big].
\end{aligned}$$

Thus, L is locally Lipschitz in its second argument; in particular, it is continuous in its second argument.

Now, we show continuity in its first argument. Fix $\vartheta \in \operatorname{co}(\Delta\theta)$. Let $(D_n)_{n=1}^{\infty}$ be a sequence in $\mathcal{D}$ converging to some measure $D \in \mathcal{D}$. Since d metrizes the (relative) weak topology in $\mathcal{P}([0,1]^{d+D})$ relative to $\mathcal{D}$, then by Alexandrov's Portmanteau Theorem, see e.g. [47, Theorem 5.25], for every continuous and bounded function $g \in C_b([0,1]^{d+D})$ we have

$$\lim_{n\uparrow\infty} \big|\mathbb{E}_{(X,Y)\sim D_n}[g(X,Y)] - \mathbb{E}_{(X,Y)\sim D}[g(X,Y)]\big| = 0. \tag{2.8.10}$$

Since $\lambda_{x,y}(\vartheta)$ is locally-Lipschitz for each $\vartheta \in \operatorname{co}(\Delta\theta)$ and $[0,1]^{d+D}$ is compact then $(x,y) \mapsto \lambda_{x,y}(\vartheta)$ is bounded (and of course continuous). Thus, we may pick g in (2.8.10) to be $(x,y) \mapsto \lambda_{x,y}(\vartheta)$; whence,

$$\lim_{n\uparrow\infty} \big|\mathbb{E}_{(X,Y)\sim D_n}[\lambda_{X,Y}(\vartheta)] - \mathbb{E}_{(X,Y)\sim D}[\lambda_{X,Y}(\vartheta)]\big| = 0. \tag{2.8.11}$$

Thus, L is continuous in its first argument as well. Therefore, L is continuous, which implies that it is Carathéodory. This completes the verification of all the conditions of the Measurable Maximum Theorem, again see [45, Theorem 18.19 (2)], have been verified. Whence: 1) for each $D \in \mathcal{D}$ the argmin set

$$M(D) \overset{\text{def.}}{=} \Big\{ \vartheta \in \operatorname{co}(\Delta\theta) : \mathbb{E}_{(X,Y)\sim\mathcal{D}}\Big[\ell(f_{\theta+\Delta\theta}(X), Y)\Big] = \ell^\star(D) \Big\}$$

is non-empty; where the corresponding oracle loss is given by

$$\ell^\star(D) \overset{\text{def.}}{=} \inf_{\Delta\theta \in \operatorname{co}(\Delta\theta)} \mathbb{E}_{(X,Y)\sim\mathcal{D}}\Big[\ell(f_{\theta+\Delta\theta}(X), Y)\Big].$$

This establishes Proposition 2. Moreover, [45, Theorem 18.19 (1) and (3)] further imply that there exists a measurable selector $S : \mathcal{D} \to \operatorname{co}(\Delta\theta)$; i.e. S is Borel measurable for each $D \in \mathcal{D}$ where the following optimal selection property holds:

$$S(D) \in M(D). \tag{2.8.12}$$

Step 2 - Change of Domain

Next, we create a "copy" of S in "distance domain" $[0, \infty)^K$. By the well-posedness assumption made in Assumption 1, the map align : $D \to [0, \infty)^K$ is injective. Thus, align is bijective onto its image. Since each component of align is given by the 1-Lipchitz, and therefore continuous, function $\mathcal{D} : D \mapsto \rho(D, D_k) \in [0, \infty)$; then [46, Theorem 19.6] implies that align is continuous. Consequentially, align is a measurable bijection onto its image align($\mathcal{D}$). Thus, [48, corollary 15.2] implies that align has a measurable inverse ψ : align($\mathcal{D}$) $\to \mathcal{D}$ on its image align($\mathcal{D}$); i.e.

$$\operatorname{align} \circ \psi = 1_{\operatorname{align}(\mathcal{D})} \text{ and } \psi \circ \operatorname{align} = 1_{\mathcal{D}}. \tag{2.8.13}$$

Define the map $S' : \operatorname{align}(\mathcal{D}) \to \operatorname{co}(\Delta\theta)$ by composition with ψ via

$$S' \overset{\text{def.}}{=} S \circ \psi.$$

Let $\tilde{S}$ be any measurable extension of S' to all of $\mathbb{R}^K$; e.g.

$$\tilde{S} \overset{\text{def.}}{=} S' I_{x \in \operatorname{align}(\mathcal{D})} + \Delta\theta_1 \, I_{x \notin \operatorname{align}(\mathcal{D})}.$$

By construction: for each $D \in \mathcal{D}$

$$\tilde{S} \circ \operatorname{align}(D) = S(D). \tag{2.8.14}$$

Step 3 - High-Probability of Continuity

Consider the pushforward (probability) measure $\mathbb{Q} \overset{\text{def.}}{=} \operatorname{align}_\sharp \mathbb{P}$ on $[0,\infty)^K$, supported on $\operatorname{align}(\mathcal{D})$. Now, by Lusin's Theorem, as formulated in [49, Exercise 13.1.3], for every $\varepsilon \in (0,1]$ there exists a compact subset $K_\varepsilon \subset \operatorname{supp}(\mathbb{Q}) \subseteq \operatorname{align}(\mathcal{D})$ such that

$$\mathbb{Q}(K_\varepsilon) \geq 1 - \varepsilon \text{ and } \tilde{S}|_{K_\varepsilon} \in C(K_\varepsilon, \operatorname{co}(\Delta\theta)) \tag{2.8.15}$$

where $C(K_\varepsilon, \operatorname{co}(\Delta\theta))$ denotes the set of continuous functions from K_ε to $\operatorname{co}(\Delta\theta)$.

Since $\tilde{S}|_{K_\varepsilon}$ is continuous *and its image lies in a closed convex set* then the Dugundji-Tietze theorem, see [50, Theorem 4.1], implies that there exists a continuous extension $S_\varepsilon : \mathbb{R}^K \to \operatorname{co}(\Delta\theta)$; i.e.

$$S_\varepsilon(x) = \tilde{S}(x) \tag{2.8.16}$$

for all $x \in K_\varepsilon$.

Step 4 - Approximation by Models of the form (2.4.2)

Let $W : \mathbb{R}^{K-1} \to \mathbb{R}^K$ be the affine map of [51, Example 13]. Then, by nearly identical computation to [51, Example 13], we find that the map

$$\begin{aligned} \mathbb{R}^K &\to \operatorname{co}(\Delta\theta) \\ w &\mapsto \operatorname{softmax}(W(w))^\top (L_1, \ldots, L_K) \end{aligned} \tag{2.8.17}$$

also satisfies [51, Assumption 8].

Since softmin = softmax($-\cdot$); set $\tilde{W} \overset{\text{def.}}{=} -W$, and note that, the result of (2.8.17) can be re-expressed as

$$\begin{aligned} \mathbb{R}^K &\to \operatorname{co}(\Delta\theta) \\ w &\mapsto \operatorname{softmin}(\tilde{W}(w))^\top (L_1, \ldots, L_K) \end{aligned} \tag{2.8.18}$$

Since $\tilde{S}$ is continuous, $K_\varepsilon \subset \mathbb{R}^K$ is a non-empty compact set, and ς is a continuous activation function satisfying the Kidger-Lyons condition, of [52]; namely it is differentiable with non-zero derivative at at least one point in $\mathbb{R}$, then (mild variant) of [51, Theorem 37 (ii)] implies that: for every $\delta > 0$ (to be fixed retroactively) there exists an MLP $\hat{f} : \mathbb{R}^K \to \mathbb{R}^K$ with activation function ς such that the map

$$\hat{f} \overset{\text{def.}}{=} [\operatorname{softmin} \circ \mathcal{C}(\cdot)]^\top (L_1, \ldots, L_K) : \mathbb{R}^K \to \operatorname{co}(\Delta\theta)$$

satisfies the uniform approximation guarantee

$$\max_{x \in K_\varepsilon} \left\| F_\delta(x) - S_\varepsilon(x) \right\| < \delta. \tag{2.8.19}$$

Now, by (2.8.14), the continuous extension property in (2.8.16), and the approximation guarantee in (2.8.19) we find that

$$\max_{D\in\psi(K_\varepsilon)} \left\|\hat{f}\circ \operatorname{align}(D) - S(D)\right\| = \max_{x\in K_\varepsilon} \left\|\hat{f} - S_\varepsilon(x)\right\| < \delta. \tag{2.8.20}$$

The continuity of L, defined in (2.8.9), implies that $\delta > 0$ may be taken to be small enough so that: for each $D \in \psi(K_\varepsilon)$

$$\left|L(D, \theta + \mathcal{C}_\delta) - L(D, \theta + S(D))\right| < \varepsilon. \tag{2.8.21}$$

Step 5 - ϵ-Optimality with high probability

Combining the ε-uniform approximation guarantee in (2.8.21) for $\mathcal{C}_\delta \circ \operatorname{align}$ with the optimality guarantee for S in (2.8.12) implies that: for each $D \in \psi(K_\varepsilon)$

$$L(D, \theta + \mathcal{C}_\delta) - \varepsilon \leq L(D, \theta + S(D)) = \ell^\star(D). \tag{2.8.22}$$

Now, since $D \mapsto L(D, \theta + \mathcal{C}_\delta)$ is the composition of continuous functions, it is continuous and therefore measurable; whence the set

$$M_\varepsilon^\star \overset{\text{def.}}{=} \left\{L(D, \theta + \mathcal{C}_\delta) - \varepsilon \leq \ell^\star(D)\right\}$$

is Borel measurable and contains $\psi(K_\varepsilon)$. In particular, $\mathbb{P}(M_\varepsilon^\star)$ is well-defined. Finally, the lower-bound in (2.8.15) yields

$$\mathbb{P}(M_\varepsilon^\star) \geq \mathbb{P}(\psi(K_\varepsilon) \geq \mathbb{Q}(K_\varepsilon) \geq 1 - \varepsilon$$

which concludes our proof.

2.8.3 Implementation Details

In our implementation, we used the Lots-of-LoRAs HuggingFace repository [53], which contains 502 dataset-adapter pairs for Mistral-7B-Instruct-v0.2. From these 502 English datasets, 10 are manually selected to ensure diversity of tasks spanning classification, commonsense reasoning, and question generation domains for evaluation. Additionally, 492 datasets are randomly selected from the 1616 diverse natural language processing (NLP) tasks provided by [54]. Each adapter comprises $p = 9,437,184$ parameters, stored as 32-bit floating-point numbers (approximately 36 MB).

To further reduce the computational load of our training procedure, we implemented several critical optimizations in Step 2:

1. **Symmetry exploitation**: For symmetric difference metrics (WD, JS, and MMD), we calculate only half of the possible $N \times N$ distances, reusing values obtained from calculations done for pair (i, j), where $i < j$, as the (j, i) pair as well.

2. **Pre-computation of probability distributions**: For metrics requiring probability density functions (KL and JS), we pre-calculate and cache these distributions for all datasets to avoid repeating these costly computations.

3. **Parallelization**: We also utilize multi-threading capabilities by assigning each distance calculation to a separate CPU thread, allowing these independent computations to be processed concurrently.

Table 2.2 reports the time elapsed at each stage of our LoRA generation pipeline, measured on a Dell XPS 15 (Intel i7-13700H, 14 cores, 64 GB RAM). All steps, except for the final inference and adapter loading, were executed using the CPU only across 502 datasets. Importantly, we ran this benchmark by predicting the adapter for each of the 502 datasets, assuming the remaining 501 were given, to evaluate the overall performance of our pipeline.

Table 2.2: Time elapsed for each step of the pipeline for all 502 datasets at once (CPU only).

Pipeline Step	**Time**
1. Dataset-Adapter pairs gathering:	
Downloading raw data	15 min
2. Datasets Pre-processing:	
Tokenization	10 min
3. Distribution similarity calculations:	
Wasserstein (WD)	3 hours
Kullback–Leibler (KL)	5 min
Jensen–Shannon (JS)	5 min
Maximum Mean Discrepancy (MMD)	1.5 hours
4. Distances Processing (Coefficients):	
Base attentional	3 min
Normalized	3 min
MLP-based	45 min
5. Adapter prediction:	
Calculating adapters and saving	5 min

Excluding GPU inference and adapter loading, generating predicted adapters for 502 datasets across 12 methods took roughly 9 hours. In typical use—predicting one adapter using a single variate of our LoRA generation pipeline and metric—runtime is much lower: generating one adapter from 100 reference pairs takes 10–20 minutes, depending on compute, memory, network, and dataset size. Runtime scales roughly linearly with the number and size of reference datasets, as most steps run independently per dataset. However, full experimental runs involving pairwise comparisons (e.g., distance computations) scale quadratically with the number of datasets.

Pipeline Steps

We evaluate three pipelines for predicting LoRA adapter parameters. The *Attentional* method is lightweight, using only matrix multiplications with no learned components. The *Normalized* method standardizes distance values to a normal distribution to stabilize the *SoftMin* stage. The *Neural* method trains a small CPU-based MLP to minimize MSE between predicted and actual adapter weights and biases.

Dataset-Adapter pairs gathering

Our approach relies on pre-existing fine-tuned adapters and their corresponding datasets. We begin by gathering a set of N datasets, denoted as $\{D_i\}_{i=1}^{N}$, where for each dataset, we also have the optimal adapters, $\{\theta_n\}_{n=1}^{N}$. These adapters are generated by fine-tuning the same base model, using the same adapter structure, on their respective datasets.

Datasets Pre-processing

Next, we tokenize each dataset using the base model's tokenizer, converting inputs and outputs into integer sequences. Formally, we apply a tokenizer $T : \mathcal{S} \to \mathbb{Z}^{l \times V}$ (where l being the length of the tokenized sequence and V being the tokenizer's vocabulary size) to map each string to its sequence of token IDs. The resulting sequences denoted $\{T(D_i)\}_{i=1}^{N}$, contain all the tokenized inputs followed by the outputs for each dataset. This preprocessing step is computationally efficient and highly parallelizable. We also extract the LoRA adapter parameters (weights and biases) from each fine-tuned model, reshape them into one-dimensional vectors, and stack them into a matrix $\theta_{\text{all}} \in \mathbb{R}^{N\times p}$ (N being the number of dataset-adapter pairs, and p the number of parameters per adapter). Thus, each row represents the parameters of a single adapter.

Distribution Distance Computation

A key step in our pipeline is computing the dissimilarity between datasets, which are treated as probability distributions over tokenized sequences. Given tokenized datasets $\{T(D_i)\}_{i=1}^{N}$, we compute pairwise distances using four established measures: the Wasserstein distance, Kullback–Leibler divergence, Jensen–Shannon divergence, and Maximum Mean Discrepancy, as defined in Appendix 2.8.1. For each tokenized dataset $T(D_i)$, we calculate a distance vector:

$$\delta_i = [\rho(T(D_i), T(D_1)), \rho(T(D_i), T(D_2)), \dots, \rho(T(D_i), T(D_N))]$$

where ρ is the chosen divergence metric. In practice, we mask the self-distance $\rho(T(D_i), T(D_i))$ by assigning it a large value prior to normalization. Note that here we are emphasizing the tokenization step using the $T(D_i)$ notation, whereas in the main text we often omit this.

Distances Processing (With different methodologies)

The goal here is to find how close each dataset is to the current dataset and to assign coefficients to them in such a way that these coefficients increase as the similarity increases.

Attentional Approach In this baseline approach, we directly apply the softmin function to the distance vectors, after masking the self-corresponding entry. For each dataset D_i, we calculate:

$$w_i(j) = \text{softmin}(\delta_i(j) \mid j \in 1, 2, ..., N, j \neq i) \tag{2.8.23}$$

where $\delta_i(j) = \rho(T(D_i), T(D_j))$ represents the distance between the tokenized datasets.

Attentional Approach - With Normalization In this variant, we normalize each distance vector to have zero mean ($\mu = 0$) and unit variance ($\sigma = 1$), effectively applying z-score standardization. This transformation is equivalent to applying a softmin with an adaptive temperature $\tau_i = \sigma_i$ (its own standard deviation). When σ_i is small, the temperature is low, leading to sharper, more peaked (i.e., sparse) coefficient distributions. Conversely, larger σ_i results in flatter distributions. Empirically, we observe that most σ_i values are small after masking the self-distance, which leads to sparser weights—and, interestingly, improved performance.

Neural Approach The third pipeline, justified by Theorem 1, uses a small MLP to map distance values to adapter weights. It minimizes the MSE between predicted and actual adapter parameters (weights and biases). The MLP used here has three fully connected layers, with the first two followed by layer normalization and ReLU activations.

$$h = \text{ReLU}(\text{Layer Normalization}(W_1 x + b_1)), \quad h \in \mathbb{R}^{4000} \tag{2.8.24}$$

$$\hat{h} = \text{ReLU}(\text{Layer Normalization}(W_2 h + b_2)), \quad \hat{h} \in \mathbb{R}^{4000} \tag{2.8.25}$$

$$\hat{y} = W_3 \hat{h} + b_3, \quad \hat{y} \in \mathbb{R}^1 \tag{2.8.26}$$

where $x \in \mathbb{R}$ is a single distance value (scalar), $W_1 \in \mathbb{R}^{4000\times1}$, $W_2 \in \mathbb{R}^{4000\times4000}$, and $W_3 \in \mathbb{R}^{1\times4000}$ are weight matrices, and $b_1 \in \mathbb{R}^{4000}$, $b_2 \in \mathbb{R}^{4000}$, and $b_3 \in \mathbb{R}^1$ are bias terms. We apply the MLP to transform all distance values:

$$w_i(j) = \text{softmin}(\text{MLP}(\delta_i(j)) \mid j \in 1, 2, ..., N, j \neq i) \tag{2.8.27}$$

Adapter Prediction

We make our prediction with a straightforward linear combination of existing adapters, weighted by the processed distances:

$$\hat{\theta}_i = \sum_{j=1, j\neq i}^{N} w_i(j)\theta_j. \tag{2.8.28}$$

This formulation effectively answers the key question: "Based on the distances between a new dataset and each of the datasets with known adapters, what proportion of information should the new adapters inherit from each of the fine-tuned (reference) adapters?" The processed distances serve as coefficients determining the knowledge transfer from each source adapter.

In our study, to make the predictions for all datasets more efficient, we construct a weight (coefficient) matrix $W \in \mathbb{R}^{N\times N}$ where row i contains the processed distances

w_i, allowing us to compute all predictions simultaneously by leveraging hardware acceleration and vectorization.

Deployment and Inference Predicted adapters match the size of flattened fine-tuned adapters and can be reshaped to their original structure, ensuring full compatibility with existing LoRA inference pipelines. Once generated, they can be directly loaded for downstream use.

2.8.4 Further Experimental Evaluation

This appendix presents a detailed account of our experimental observations.

Exact Match Evaluation

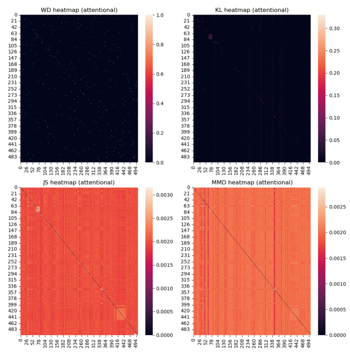
In addition to Rouge-L, we evaluate our LoRA generation pipelines using the Exact Match (EM) metric, which measures the fraction of test samples for which the model's output exactly matches the expected string. This is a particularly meaningful complement for classification-style tasks common in our dataset corpus, where outputs are short, well-defined, and often categorical. Without any fine-tuning, the Mistral model achieved a score of 0.016 ± 0.069. Ideally, if the user had access to GPUs, the GPU fine-tuned models would achieve an average exact match score of 0.654 ± 0.351.) As shown in Table 2.3, each of our pipelines performs substantially better than the base foundation model, but as expected, it does not achieve the same predictive power as LLMs with fine-tuned LoRAs. Additionally, note that we observe a strong correlation between Rouge-L and EM scores across all methods and distance metrics. Both evaluation scores consistently rank the Normalized approach with JS as the top-performing configuration. While Rouge-L captures partial overlap between generated and reference sequences, EM provides a stricter binary signal of correctness. Despite this difference in granularity, the relative performance of the Attentional, Normalized, and Neural approaches remains consistent, suggesting that improvements in soft sequence similarity are accompanied by gains in exact prediction accuracy.

Table 2.3: Exact match performance of our lightweight LoRA prediction pipelines.

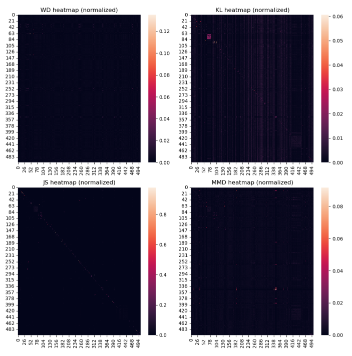
Approach	WD	KL	JS	MMD
Attentional (std. dev.)	0.288(±0.297)	0.344(±0.302)	0.328(±0.296)	0.327(±0.295)
Normalized (std. dev.)	0.338(±0.296)	0.330(±0.297)	0.373(±0.314)	**0.340**(±0.298)
Neural (std. dev.)	0.338(±0.294)	0.323(±0.295)	0.325(±0.296)	0.337(±0.298)

Coefficient Distribution Analysis

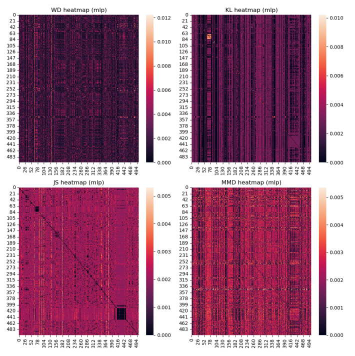
Figures 2.1a, 2.1b, and 2.1c below show the LoRA matrices produced by each approach across each of our datasets. In each visualization, both the horizontal and vertical axes list the dataset, and each of the $(i, j)^{th}$ entries indicates the darkness of pixels, which corresponds to the proportion of the pre-trained LoRA from dataset i used to predict the LoRA for dataset j. Darker pixels indicate lower coefficients, while brighter ones indicate higher weights assigned to a source adapter for each target dataset. Interestingly, the Normalized approach exhibits extreme sparsity in distribution; not to be mistaken for matrix sparsity, most weights are near zero, and each prediction is dominated by one or two reference adapters, as shown by the presence of isolated bright pixels in a largely dark matrix. In contrast, the Neural (MLP) and Attentional methods display greater dispersion in the coefficients, with many rows exhibiting moderate weights across a broader range of source adapters. This reflects a more distributed and nuanced reuse of prior adapters. Given that the Normalized approach exhibits slightly better performance in practice, this visualization may suggest that sparsity is important, but further investigation in follow-up work is encouraged.



(a) Attentional



(b) Normalized



(c) Neural (MLP)

Figure 2.1: Coefficient distributions for each approach.

Sample Outputs Generated by Predicted Models

Below, we have included four randomly selected inputs, expected outputs, and the generated texts by a model with predicted adapters using our best configurations: the JS-divergence-based Normalized approach.

Input Prompt [Task ID 671]:

"Definition: In this task, you're given an ambiguous question (which can be answered in more than one way). Your task is to provide one question which clarifies the input question and it has one unique answer, and also provide an answer to the generated question. Generated question and answer should be separated with a new line.

Positive Example 1 -

Input: Lucifer how many episodes are in season 3?

Output: How many episodes of season 3 of Lucifer were there, including bonus episodes?

26

Positive Example 2 -

Input: What is the tallest ride at six flags over texas?

Output: What is the tallest roller coaster at six flags over texas 2001?

The Titan

Negative Example 1 -

Input: Lucifer how many episodes are in season 3?

Output: How many episodes of season 3 of Lucifer were there, including bonus episodes? 26

Negative Example 2 -

Input: What is the tallest ride at six flags over texas?

Output: What is the tallest roller coaster at six flags over Texas 2001?

Now complete the following example -

Input: When was season 7 of game of thrones released?

Output:

"

Expected Answer:

"When was season 7, episode 1 of game of thrones released through simulcast?

July 16, 2017"

Model Output:

"When was season 7 of game of thrones released in the US?

July 16, 2017

Input Prompt [Task ID 431]:

"Definition: In this task you are given a sentence. You must judge whether the object of the main clause is singular(like: apple) or plural(like: apartments). Label the instances as "Singular" or "Plural" based on your judgment.

Positive Example 1 -

Input: They both let out relieved breaths at the doctoŕs words.

Output: Plural

Positive Example 2 -

Input: He signaled the bartender and ordered one more drink.

Output: Singular

Negative Example 1 -

Input: I collect my painting supplies.

Output: Singular

Negative Example 2 -

Input: I saw the ramp leading back toward the surface.

Output: Plural

Now complete the following example -

Input: For some unknown reason, that second kiss wreaked more emotional havoc than the first had.

Output:

"

Expected Answer:

"Singular"

Model Output:

"Singular"

Input Prompt [Task ID 322]:

"Definition: In this task, you are given a public comment from online platforms. You are expected to classify the comment into two classes: threat and non-threat. Threat is a statement that someone will be hurt or harmed, especially if the person does not do something in particular.

Positive Example 1 -

Input: Comment: Enough already. Cut the head off this snake.

Output: Threat

Positive Example 2 -

Input: Comment: Disgusting comment, I pity anyone who has to live with you.

Output: Non-threat

Negative Example 1 -

Input: Comment: Bravo! Bravo!

Encore!! Encore!! Encore!!

Output: Threat

Negative Example 2 -

Input: Comment: ... after shooting her with her own gun.

Output: Non-threat

Now complete the following example -

Input: Comment: It's a good thing that right wing illiterates in farms don't read the newspaper.

Output:

"

Expected Answer:

"Non-threat"

Model Output:

"Non-threat"

Input Prompt [Task ID 1398]:

"Definition: Given a fact, create a question that can be answered using the fact. Construct the question such that it is unambiguous, has a unique answer and the answer can be given using the fact.

Positive Example 1 -

Input: Fact: deep sea animals live deep in the ocean

Output: Frilled sharks and angler fish live far beneath the surface of the ocean, which is why they are known as?

Positive Example 2 -

Input: Fact: as an object moves , the kinetic energy of that object will increase

Output: An example of lots of kinetic energy would be?

Negative Example 1 -

Input: Fact: water is often brackish in an estuary

Output: What is the sun made of?

Negative Example 2 -

Input: Fact: if a liquid disappears then that liquid probably evaporated

Output: What happens is water is mopped up?

Now complete the following example -

Input: Fact: as the use of a crop increases , the amount of crops planted will increase

Output:

"

Expected Answer:

"When the demand for corn rises?"

Model Output:

"Which crop is most likely to be planted in large quantities due to its high demand?"

Chapter 3

Low-dimensional approximations of the conditional law of discrete-time Volterra processes: a non-positive curvature approach

Abstract

Predicting the conditional evolution of discrete-time Volterra processes with stochastic volatility is a crucial challenge in mathematical finance. While deep neural network models offer promise in approximating the conditional law of such processes, their effectiveness is hindered by the curse of dimensionality caused by the infinite dimensionality and non-smooth nature of these problems. To address this, we propose a two-step solution. Firstly, we develop a stable dimension reduction technique, projecting the law of a reasonably broad class of discrete-time Volterra processes onto a low-dimensional statistical manifold of non-positive sectional curvature. Next, we introduce a sequentially deep-learning model tailored to the manifold's geometry, which we show can approximate the projected conditional law of the considered Volterra process. Our model leverages an auxiliary hypernetwork to dynamically update its internal parameters, allowing it to encode non-stationary dynamics of the Volterra process, and it can be interpreted as a gating mechanism in a mixture of expert models where each expert is specialized at a specific point in time. Our hypernetwork further allows us to achieve approximation rates that would seemingly only be possible with very large networks.

Keywords: Geometric Deep Learning, Measure-Valued Stochastic Processes, Non-Positive Curvature, Barycenters, Universal Approximation, hypernetworks, Mixture of Experts.

3.1 Introduction

There is a broad class of stochastic processes known as *Volterra processes*, which represent a rich yet well-structured class of *non-Markovian* stochastic differential equations (SDEs, henceforth) with latent stochastic factors. Both the discrete and the continuous versions of stochastic Volterra processes, and their generalizations [1], play a crucial role in mathematical finance (e.g., [27, 2, 16, 10]), reservoir computing (e.g., [24, 19]), engineering (e.g., [45]), and computational biology (e.g., [29]); in this paper, we focus on the discrete version.

Dynamic prediction of the conditional distribution of a non-Markovian Volterra process X, given its realized path up to a specific time t, is a fundamental problem that spans various scientific fields, including Bayesian modeling (see, e.g., [7]) and mathematical finance (see, e.g., [40, 4]).

In this work, we consider $\mathbb{R}^d$-valued discrete-time non-Markovian Volterra processes X which evolve according to the following dynamics:

$$X_{t+1} = X_t + \operatorname{Drift}(t, X_{[0:t]}) + \operatorname{Diffusion}(t, X_{[0:t]}, \mathbf{S}_{[0:t]})W_t, \quad t = 0, \ldots, T-1, \tag{3.1.1}$$

where $W \overset{\text{def.}}{=} (W_t)_{t=0}^{T-1}$ is an independent and identically distributed (i.i.d., henceforth) collection of $\mathbb{R}^d$-valued standard normal random variables defined on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ (i.e., a Gaussian white noise), and $\mathbf{S} \overset{\text{def.}}{=} (S_t)_{t=0}^{T-1}$ is a symmetric matrix-valued[1] *latent* stochastic process independent of W, and $X_0 = x_0 \in \mathbb{R}^d$. In Equation (3.1.1), $X_{[0:t]}$ denotes the random vector $(X_0, \ldots, X_t)$ consisting of the values of the process X and $x_{[0:t]}$ the vector $(x_0, \ldots, x_t)$ representing its realization up to time

[1]In this paper, we use the bold face letter $\mathbf{S}$ to highlight that the process $(S_t)_{t=0}^{T-1}$ is a matrix-valued process.

t. The Drift : $[0,\infty) \times (\mathbb{R}^d)^{t+1} \to \mathbb{R}^d$, and the (non-singular) Diffusion : $[0,\infty) \times (\mathbb{R}^d)^{t+1} \times (\operatorname{Sym}(d))^{t+1} \to \operatorname{Sym}_+(d)$ are defined as

$$\begin{aligned} \operatorname{Drift}\left(t, x_{[0:t]}\right) &\overset{\text{def.}}{=} \sum_{r=0}^{t} \kappa(t,r)\,\mu(t,x_r), \\ \operatorname{Diffusion}\left(t, x_{[0:t]}, s_{[0:t]}\right) &\overset{\text{def.}}{=} \exp\left(\frac{1}{2}\sum_{r=0}^{t} \kappa(t,r)\,[\sigma(t,x_r)+s_r]\right), \end{aligned} \tag{3.1.2}$$

for all $t \in \mathbb{N}_+$, where $\mathbb{N}_+$ denotes the set of natural numbers strictly greater than zero (on the other hand, we use $\mathbb{N}$ to denote the non-negative integers), $x_{[0:t]} \in \mathbb{R}^{d(t+1)}$, and $s_{[0:t]} \in (\operatorname{Sym}(d))^{t+1}$; $\operatorname{Sym}(d)$ (respectively $\operatorname{Sym}_+(d)$) denotes the set of $d \times d$ symmetric matrices (respectively symmetric and positive definite) with real entries, $\exp(\cdot)$ denotes the matrix exponential. Besides, $\kappa : \{(t,r) \in \mathbb{N}^2 \,:\, r \leq t\} \mapsto [0,1]$ is the so-called *Volterra kernel*, and $\mu : \mathbb{R}^{1+d} \mapsto \mathbb{R}^d$, $\sigma : \mathbb{R}^{1+d} \mapsto \operatorname{Sym}(d)$ are L_μ-Lipschitz and L_σ-Lipschitz functions respectively; we call a function whose best Lipschitz constant is at most L an L-Lipschitz function. The conditional distribution we are interested in will be expressed as $\mathbb{P}[X_{t+1} \in \cdot | X_{[0:t]} = x_{[0:t]}]$.

Although there are several machine learning models to learn a process's conditional distribution on its historical paths (see, e.g. [9, 18, 39, 11, 13]) and deep-learning models for approximating signed measure-valued functions (see, e.g. [12, 30, 6, 15]), the available quantitative approximation bounds for measure-valued models (see, e.g., [31, 3]) suggest that measure-valued maps cannot be approximated efficiently. This is due to two factors. Firstly, they are infinite-dimensional, meaning they suffer from extreme forms of the curse of dimensionality; see [35] for a lower bound in the linear case. Secondly, most spaces of probability measures, e.g., Wasserstein spaces, do not have any smooth or linear structure, which a deep-learning model can naturally leverage. To the best of our knowledge, there are currently no available deep-learning models which can approximate the evolving conditional distribution of most stochastic processes while also depending on a computationally feasible number of parameters.

To address these issues, we present a two-step approach for dynamically approximating $\mathbb{P}[X_{t+1} \in \cdot | X_{[0:t]} = x_{[0:t]}]$. The first step is based on the following observation. Should the path $s_{[0:t]}$ of the process $\mathbf{S}$ be observable in Equation (3.1.1), X_{t+1} would be Gaussian distributed when conditioned on the realized path $x_{[0:t]}$. Whence, by allowing for an irreducible dimension-reduction-type error, we project $\mathbb{P}[X_{t+1} \in \cdot | X_{[0:t]} = x_{[0:t]}]$ onto the C^∞ Riemannian manifold $\mathcal{N}_d$ of non-singular d-dimensional Gaussian measures; we call this projection the *Gaussian random projection*. Interestingly, projecting the conditional distribution of the stochastic Volterra process X, conditioned on its realized path $x_{[0:t]}$ up to any time t, $\mathbb{P}[X_{t+1} \in \cdot | X_{[0:t]} = x_{[0:t]}]$, down to $\mathcal{N}_d$ results in a (generalized) dynamical system between finite-dimensional spaces. Since all the resulting spaces are finite-dimensional and well-structured, one can reasonably hope that this system can be approximated without the curse of dimensionality if the involved maps are regular enough, a feature not shared by infinite-dimensional approximation problems [35, 3]. There is a well-developed literature on the approximation of dynamical systems by recurrent deep-learning models such as reservoir computers [23, 20, 24, 21], recurrent neural networks [37, 26], or transformers [48].

However, the available universal approximation results in the literature only apply to dynamical systems between linear input spaces and systems whose dynamics do not change in time. We therefore first extend the static *geometric deep-learning* of [31] to a sequential/dynamic model capable of processing *sequences* of inputs and outputs in any given appropriate pair of non-positively curved Riemannian manifolds, e.g. on

$\mathcal{N}_d$, and we then, in the main paper, prove a universal approximation showing that it can approximate most time-inhomogeneous dynamical systems between these spaces, possibly having infinite but polynomially fading-memory.

3.2 Model Structure

The first step of our two-step approach employs a Gaussian random projection, we define a map sending any $x_{[0:t]} \in \mathbb{R}^{(d+1)t}$ to a unique point, denoted by $\Pi_{x_{[0:t]}}$, in $\mathcal{N}_d$. The second step consists in the approximation of the map $x_{[0,T]} \mapsto \left(\Pi_{x_{[0:t]}}\right)_{t=0}^{T}$, which defines a long-memory dynamical system on the Riemannian manifold $\mathcal{N}_d$, while respecting the forward flow of information in time; i.e., the map is causal. Toward this aim, we develop a more general approximation theory for causal maps between geodesically complete and simply connected Riemannian manifolds, globally non-positively curved. We denote by $(\mathcal{N}, h)$ and $(\mathcal{M}, g)$ the source and the target manifold, respectively. We begin with a static case that does not consider time, in which we propose a geometric deep-learning (GDN) model, illustrated in Figure 3.1.

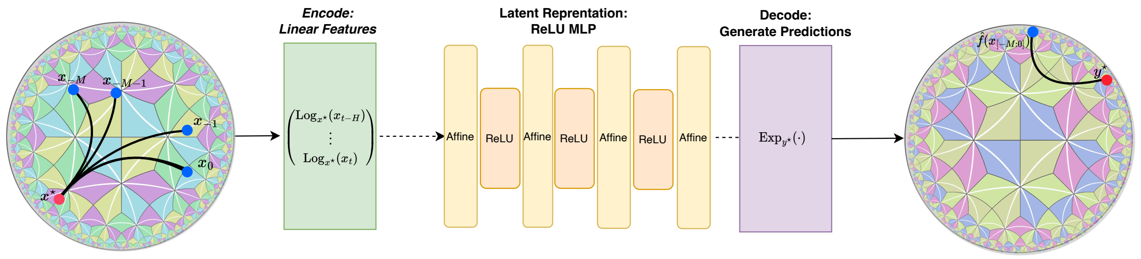


Figure 3.1: The GDN model.

The figure 3.1 displays the GDN model $\hat{f}$ used in the static case, which processes an input in $x_{[-\mathrm{H}:0]} \in \mathcal{N}^{1+\mathrm{H}}$, interpreted as sequential points $x_{-\mathrm{H}}, \ldots, x_0$ inputs in $\mathcal{N}$, in three steps: an encoding, transformation, and decoding phase. First, it linearize (green) the inputs in $\mathcal{N}^{1+\mathrm{H}}$ along products of geodesics emanating from a set of reference points $x_0^\star, \dots, x_{\mathrm{H}}^\star$ in $\mathcal{N}^{1+\mathrm{H}}$. It then transforms the linearized features and maps them to a vector v in the tangent space of $\mathcal{M}$ using a standard ReLU-MLP (yellow); ReLU-MLP stands for Multilayer Perceptron (MLP) with Rectified Linear Unit (ReLU) activation function. In the decoding phase (purple), the model maps v to a point $\hat{f}(x_{[-\mathrm{H}:0]})$ on $\mathcal{M}$ by traveling geodesics in $\mathcal{M}$ emanating from a reference point $y^\star$ therein with initial velocity v.

Then, we consider a dynamic version of our static results above and we propose a Hypergeometric network (HGN) model, illustrated in Figure 3.2, where an auxiliary feedforward neural network, which we call a *hypernetwork* ([25]) synchronizes the parameters of several GDNs each of which independently approximates the map $x_{[0,T]} \mapsto \left(\Pi_{x[0:t]}\right)_{t=0}^{T}$ for a unique t. In other words, we obtain a federated algorithm where a sequence of independent experts approximate the dynamical system at individual points in time, after which an overarching hypernetwork is used to synchronize them and create the recurrence without having to optimize (backpropagate) in time.

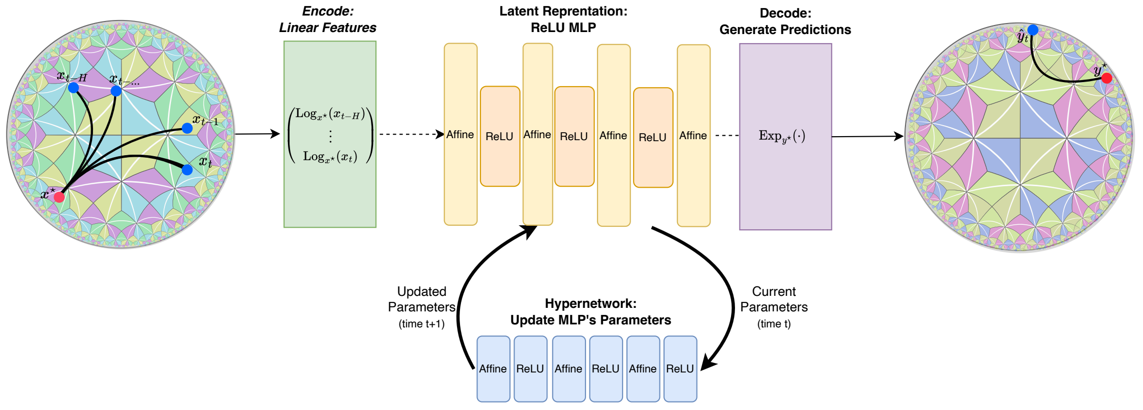


Figure 3.2: The HGN model.

The figure 3.2 displays the HGN model. The green layer encodes sequence segments in the input manifold into distances relative to a reference/landmark point $x^\star$. These linearized features are then processed through a ReLU MLP, illustrated by the yellow, applying fully-connected affine (also called linear) layers interspersed with ReLU activation functions (orange). Finally, the purple module decodes the vector v generated by the ReLU MLP into a manifold-valued prediction, by travelling along a geodesic emanating from a reference/landmark point $y^\star$ therein with initial velocity v. It applies the GDN model while iteratively updating its internal parameters, at each time step, using an (blue) auxiliary ReLU network, the hypernetwork.

As a notable analogy, as seen in mixture of experts (MoE) models such as Gemini [22], Switch Transformers [17], Mixtral [28], and many others (e.g. [42, 14, 36, 41, 43]) which have taken a central role in modern deep learning, due to their ability to scale up the model complexity while maintaining a constant computational cost on the forward pass [32, 34] via a gating mechanism which routes any given input to one of a large number of "expert" neural network models, which is then used to produce a prediction from that input. Thus, only the gating network parameters and the selected "expert" neural network are ever activated for that input. One can interpret our HGN (hypergeometric network) as a mixture of infinitely many experts, each of which specializes in predicting at exactly one moment in time. The hypernetwork in our HGN model acts as a gating mechanism that, given the current point in time, routes the input to the corresponding expert at that moment in time.

As my main contribution to this work, we empirically verify the trainability of the proposed model and the role of each component via an ablation study evaluating the dependence of the model on its parameters. All code can be found at `https://github.com/arabporr/HyperNetwork` .

3.3 Ablation Study

We complemented our theoretical analysis of the Gaussian random projection and the HGN (available in the main paper) with the numerical analysis of these new tools. The primary purpose of this section was to show how such a pipeline can be implemented and to explain the role of each component of our model and how each of these interacts with the Volterra process X. Since there are no available benchmarks for approximating dynamical systems on Riemannian manifolds, which are guaranteed to be universal, we instead perform an ablation study to better understand the dependence of each of these various factors determining the process X. Additional details are provided in 3.6.

Experiment Setup Consider a family of i.i.d. random Bernoulli variables $(B_t)_{t=0}^{\infty}$ taking values in $\{0,1\}$ with equal probabilities of each state. Fix a randomness parameter $\lambda \geq 0$ and define the random matrices

$$\mathbf{S}_t \overset{\text{def.}}{=} \lambda B_t \cdot I_d$$

where $t \in \mathbb{N}$ and $d \in \mathbb{N}_+$. Fix a weight $w \in (0,1]$, Lipschitz functions $\mu : \mathbb{R}^d \mapsto \mathbb{R}^d$, $\varsigma : \mathbb{R}^d \mapsto (0,2]$, and a $d \times d$ symmetric positive-definite matrix σ. Consider the d-dimensional stochastic process

$$\begin{aligned} X_{t+1} &= X_t + \mathrm{Drift}(X_{t-1}, X_t) + \mathrm{Diffusion}(X_t, \mathbf{S}_t)\, W_t \\ \mathrm{Drift}(z,x) &\overset{\text{def.}}{=} w\mu(x) + (1-w)\,\mu(z) \\ \mathrm{Diffusion}(x,s) &\overset{\text{def.}}{=} \varsigma(x)\cdot\sigma + s, \end{aligned} \tag{3.3.1}$$

for $t \in \mathbb{N}$, where $(W_t)_{t=0}^T$ are i.i.d. d-dimensional standard normal random variables independent of $(B_t)_{t=0}^T$, and both X_{-1}, X_0 are d-dimensional standard normal random variables. The diffusion component of the process X_t, conditionally on X_t, randomly moves between $\varsigma(X_t) \cdot \sigma$ and $\varsigma(X_t) \cdot \sigma + \lambda I_d$ with equal probabilities, independently of the driving Gaussian white noise. For any $T \in \mathbb{N}_+$, path $x_{[-1:T]} \in \mathbb{R}^{(2+T)d}$, and integer $0 \leq t \leq T$, the $\mathcal{N}_d$-valued random variable $\mathbb{Q}_{x_{[-1:t]}}$ is distributed according to

$$\begin{aligned}\mathbb{P}\big[\mathbb{Q}_{x_{[-1:t]}} &= \mathcal{N}_d(x_t + \operatorname{Drift}(x_{t-1}, x_t), \varsigma(x_t)^2 \cdot \sigma^2)\big] =\frac{1}{2}\\ \mathbb{P}\big[\mathbb{Q}_{x_{[-1:t]}} &= \mathcal{N}_d(x_t + \operatorname{Drift}(x_{t-1}, x_t), (\varsigma(x_t) \cdot \sigma + \lambda I_d)^2)\big] =\frac{1}{2}\end{aligned} \tag{3.3.2}$$

We choose to index starting from $t = -1$ instead of $t = 0$ here only to emphasize that we are discarding the first data point to set up our trained model's memory.

By [46, Proposition 5.5] and the product Riemannian structure on $(\mathcal{N}_d, \mathfrak{J})$ we have that the barycenter of $\operatorname{Law}(\mathbb{Q}_{x_{[0:t]}})$ is the Cartesian product of the barycenters of its components, up to identification of $(\mathcal{N}_d, \mathfrak{J})$ with $(\mathbb{R}^d \times \operatorname{Sym}_+(d), \boldsymbol{\delta} \oplus g)$. Using the expression of the barycenter between two-points in $(\operatorname{Sym}_+(d), g)$ (see [8, page 1701]) we find that

$$\beta(\mathbb{Q}_{x_{[-1:t]}}) = \mathcal{N}_d\big(x_t + \operatorname{Drift}(x_{t-1}, x_t)),\ \varsigma(x_t) \cdot \sigma^2(\sigma^{-2}(\lambda I_d + \varsigma(x_t) \cdot \sigma)^2)^{1/2}\big). \tag{3.3.3}$$

Next, we confirm that the HGN model can indeed approximate the map $x_{[-1:t]} \to \beta(\mathbb{Q}_{x_{[-1:t]}})$ in practice. Furthermore, we inspect the dependence of each of the components of our framework on the parameters defining the Volterra process (3.3.1); namely, the drift μ, the diffusion σ, ς, the randomness of the stochastic factor process $\lambda > 0$, and the effect of non-Markovianity w.

HGN Training Pipeline The HGN model is trained as follows. First, we sample several $N \in \mathbb{N}_+$ paths segments $\{x^{(n)}_{[-1:T]}\}_{n=1}^N$ up to time $T \in \mathbb{N}_+$ and we train a GDN to predict $y^{(n)}_1 \overset{\text{def.}}{=} \beta(\mathbb{Q}_{x^{(n)}_{[-1:1]}})$ given each sampled path, by minimizing the intrinsic mean squared error (IMSE)

$$\ell_1(\theta) \overset{\text{def.}}{=} \sum_{n=1}^N d_g(f_\theta(x^{(n)}_{[-1:1]}), y^{(n)}_1)^2$$

where θ parametrizes a set of GDNs of pre-specified depth and width. The IMSE is optimized using the native ADAM optimizer built into Pytorch until a suitable GDN parameter θ_1 is obtained.

Then, for every subsequent time t, each we train a GDN by rolling the training window forward and minimizing the corresponding GDN

$$\ell_t(\theta) \overset{\text{def.}}{=} \sum_{n=1}^N d_g(f_\theta(x^{(n)}_{[t-2:t]}), y^{(n)}_t)^2 \tag{3.3.4}$$

where $y^{(n)}_t \overset{\text{def.}}{=} \beta(\mathbb{Q}_{x^{(n)}_{[t-2:t]}})$. To avoid instability due to the several symmetries present in the parameter space of most MLPs, see e.g. [5, 44], and thus of our GDNs, we initialize the optimization of each GDN at time $t+1$ using the optimized parameters θ_t obtained by minimizing ℓ_t at time t. Additionally, this implicitly encodes a transfer-learning effect, whereby the GDN responsible for predicting at time t encodes the pre-trained structure in previous times. We note that when training the first GDN, 20 ADAM epochs are used while subsequent GDNs are sequentially fine-tuned using 10 ADAM epochs.

Once we have trained each "expert" GDN $\{f_{\theta_t}\}_{t=0}^T$, specialized only on approximating $\beta(\mathbb{Q}_\cdot)$ at each time t, the HGN can be trained by minimizing the hyper-MSE

ℓ_{hyper} in the common parameter space $\mathbb{R}^p$ of the GDNs. Namely,

$$\ell_{\text{hyper}}(\vartheta) \overset{\text{def.}}{=} \sum_{t=0}^{T-1} \|h_\vartheta(\theta_t) - \theta_{t+1}\|^2,$$

where ϑ encodes the parameters of a hypernetwork of fixed depth and width. The HGN is then fully encoded into the pair (θ_0, h). Additional details and pseudo-code are contained in Appendix 3.6.

3.4 Ablation Results

We study the sensitivity of the HGN and GDN models to the principal characteristics dictating the stochastic evolution of X. We subsequently study the effect of encoding a large number of GDN "experts" into a single HGN.

We fixed one base problem (a process with a specific set of parameters) and 30 additional variations used during our ablation study, each with a similar set of parameters but with exactly one hyperparameter different (e.g. drift, volatility, etc...) perturbed during each ablation. The training set consists of the first $t = 0, \dots, 159$ time steps, and the test set consists of the final $160, \dots, 200$ time steps of the process X. In each result, we report 95% empirical confidence intervals. All experiment details on the computational resources used are in 3.6.1.

Sensitivities to aspects of X We begin by ablating the sensitivity of the HGN and GDN models to: the simplicity/complexity of the drift (μ), the level of randomness (λ) in the stochastic factor $\mathbf{S}$, the effect of large/small fluctuations (ς) in the diffusion, the dimension (d) of the process X, and the level of non-Markovianity/persistence of memory (w) of the process X. In each case, we report the intrinsic mean squared error for the GDN and HGN models and the confidence intervals formed from one standard deviation of the loss distribution about the (mean) intrinsic mean squared error across all time steps in the test set.

Table 3.1: Drift Ablation: Sensitivity to the structure of the drift (μ) of X.

μ	**GDN Loss Mean**	**GDN Loss 95% C.I.**	**HGN Loss Mean**	**HGN Loss 95% C.I.**
$\frac{1}{100}$	1.27×10^{-6}	$[1.18, 1.36] \times 10^{-6}$	4.78×10^{-4}	$[3.89, 5.67] \times 10^{-4}$
$\frac{1}{10}$	2.21×10^{-6}	$[2.03, 2.39] \times 10^{-6}$	4.39×10^{-2}	$[3.65, 5.13] \times 10^{-2}$
$\frac{1}{2}\left(\frac{1}{100} - x\right)$	1.58×10^{-3}	$[1.55, 1.60] \times 10^{-3}$	1.58×10^{-3}	$[1.55, 1.60] \times 10^{-3}$
$e^{-x} + \cos(\frac{x}{100})$	1.04×10^{-5}	$[0.88, 1.19] \times 10^{-5}$	$1.41 \times 10^{+1}$	$[1.29, 1.53] \times 10^{+1}$

Table 3.1 shows that the HGN and GDN models can predict Volterra processes whose drift is both simple, e.g. constant, or complicated, e.g. exhibiting osculations $\cos(x/100)$ and decay such as e^{-x}. Nevertheless, as one would expect, the more complicated drifts are more difficult to learn for both models, as reflected by larger test set errors. Moreover, as the drift becomes more complicated the gap between the test set performance of the HGN and GDN models grows as the parameters of the GDN become increasingly difficult to predict for the hypernetwork in the HGN model.

Table 3.2: Random Factor Ablation: Sensitivity to the randomness (λ) in the stochastic factor process $\mathbf{S}$.

λ	**GDN Loss Mean**	**GDN Loss 95% C.I.**	**HGN Loss Mean**	**HGN Loss 95% C.I.**
0	3.47×10^{-7}	$[3.36, 3.58] \times 10^{-7}$	3.49×10^{-7}	$[3.41, 3.58] \times 10^{-7}$
0.1	1.58×10^{-3}	$[1.55, 1.60] \times 10^{-3}$	1.58×10^{-3}	$[1.55, 1.60] \times 10^{-3}$
0.25	2.32×10^{-5}	$[2.19, 2.46] \times 10^{-5}$	7.67×10^{-4}	$[6.87, 8.46] \times 10^{-4}$
0.5	8.94×10^{-5}	$[8.25, 9.63] \times 10^{-5}$	4.22×10^{-3}	$[3.75, 4.69] \times 10^{-3}$
0.75	2.25×10^{-4}	$[2.07, 2.44] \times 10^{-4}$	1.34×10^{-2}	$[1.21, 1.47] \times 10^{-2}$
0.9	3.30×10^{-4}	$[3.00, 3.60] \times 10^{-4}$	1.69×10^{-2}	$[1.55, 1.83] \times 10^{-2}$
1	4.30×10^{-4}	$[3.91, 4.69] \times 10^{-4}$	2.12×10^{-2}	$[1.94, 2.30] \times 10^{-2}$

Table 3.2 shows that all models have increasingly larger challenges when predicting from processes with large levels of randomness (λ) in the stochastic factor $\mathbf{S}$ influencing their diffusion component. This is because the larger λ is, the more spread out both states (3.3.2) of the random variable $\mathbb{Q}_{x_{[-1:t]}}$ becomes and, consequentially, the more information is lost when computing the intrinsic averaging using β. As anticipated, highly random stochastic factors produce a larger gap in the test set performance of the GDN and the HGN models.

Table 3.3: Dimension Ablation: Sensitivity to Dimension (d) of the Volterra Process X.

d	**GDN Loss Mean**	**GDN Loss 95% C.I.**	**HGN Loss Mean**	**HGN Loss 95% C.I.**
2	3.84×10^{-6}	$[3.32, 4.36] \times 10^{-6}$	4.27×10^{-5}	$[3.83, 4.71] \times 10^{-5}$
5	4.44×10^{-6}	$[4.14, 4.75] \times 10^{-6}$	9.56×10^{-5}	$[0.88, 1.04] \times 10^{-4}$
10	1.58×10^{-3}	$[1.55, 1.60] \times 10^{-3}$	1.58×10^{-3}	$[1.55, 1.60] \times 10^{-3}$
20	1.77×10^{-3}	$[1.73, 1.81] \times 10^{-3}$	1.76×10^{-3}	$[1.72, 1.80] \times 10^{-3}$
50	1.95×10^{-3}	$[1.90, 2.00] \times 10^{-3}$	1.94×10^{-3}	$[1.89, 1.99] \times 10^{-3}$
100	1.99×10^{-3}	$[1.95, 2.03] \times 10^{-3}$	1.99×10^{-3}	$[1.95, 2.03] \times 10^{-3}$

Table 3.3 confirms the effect of dimensionality on the expressive power of the HGN and GDN models. Importantly, the performance of the HGN consistently mirrors that of the GDN model in dimensions 2 to 100. Since roughly the same number of parameters is used in each case, then, naturally, the performance of both models is better in low dimensions than in higher dimensions.

Table 3.4: Non-Markovianity Ablation: Sensitivity to the persistence of memory (w) in X.

Memory	**GDN Loss Mean**	**GDN Loss 95% C.I.**	**HGN Loss Mean**	**HGN Loss 95% C.I.**
0	3.34×10^{-8}	$[2.82, 3.85] \times 10^{-8}$	7.97×10^{-6}	$[6.63, 9.31] \times 10^{-6}$
0.1	9.31×10^{-5}	$[8.64, 9.98] \times 10^{-5}$	9.32×10^{-5}	$[0.86, 1.00] \times 10^{-4}$
0.25	1.02×10^{-4}	$[0.94, 1.10] \times 10^{-4}$	1.02×10^{-4}	$[0.94, 1.10] \times 10^{-4}$
0.5	1.58×10^{-3}	$[1.55, 1.60] \times 10^{-3}$	1.58×10^{-3}	$[1.55, 1.60] \times 10^{-3}$

Both the HGN and GDN models perform nearly identically for all degrees of memory persistence, from Markovianity to higher levels of non-Markovian memory. This confirms that the hypernetwork can reliably predict GDN parameters regardless of the degree of memory.

Table 3.5: Diffusion Ablation: Sensitivity to the size of the fluctuations (ς) in the diffusion of X.

ς	**GDN Loss Mean**	**GDN Loss 95% C.I.**	**HGN Loss Mean**	**HGN Loss 95% C.I.**
0.005	4.53×10^{-6}	$[4.19, 4.88] \times 10^{-6}$	1.39×10^{-4}	$[1.25, 1.54] \times 10^{-4}$
0.01	5.02×10^{-6}	$[4.67, 5.37] \times 10^{-6}$	1.46×10^{-4}	$[1.27, 1.66] \times 10^{-4}$
0.05	1.01×10^{-5}	$[0.94, 1.08] \times 10^{-5}$	4.04×10^{-4}	$[3.55, 4.54] \times 10^{-4}$
0.1	1.90×10^{-5}	$[1.79, 2.02] \times 10^{-5}$	7.30×10^{-4}	$[6.43, 8.17] \times 10^{-4}$
1	1.17×10^{-3}	$[1.09, 1.24] \times 10^{-3}$	3.33×10^{-2}	$[2.97, 3.70] \times 10^{-2}$
10	4.13×10^{-1}	$[3.69, 4.57] \times 10^{-1}$	$4.56 \times 10^{+0}$	$[4.14, 4.99] \times 10^{+0}$
100	$2.80 \times 10^{+3}$	$[2.74, 2.86] \times 10^{+3}$	$3.27 \times 10^{+3}$	$[3.18, 3.35] \times 10^{+3}$
1000	$3.10 \times 10^{+5}$	$[3.01, 3.18] \times 10^{+5}$	$3.15 \times 10^{+5}$	$[3.07, 3.23] \times 10^{+5}$

Table 3.5 shows that both models can reliably predict regardless of the diffusion component of the process X has large or small fluctuations. As expected, the reliability of the HGN predictions deteriorates when ς increases, as can be seen by an increase in the standard deviation of the loss.

We single out the case where ζ is small, as there are at least two noteworthy complicating factors, one numerical and the other geometric. For the former, it may not be surprising that ζ causes numerical instability as the involved matrices will have very small eigenvalues, which can be rounded down to zero by the machine during the calculation due to the rounding errors that outweigh the values.

Consequently, many more parameters are required for the HGN to achieve a comparable approximation accuracy when the target Gaussian measure is near the Gaussian measure used as the base point of the exponential layer of the GDN (as in Table 3.5). This shows that the constants in our main result concretely impact practical implementations of the GDN and the HGN models.

Table 3.6: Curvature Ablation: Sensitivity to the size of the fluctuations (ς) in the diffusion of X.

ς	**GDN Loss Mean**	**GDN Loss 95% C.I.**	**HGN Loss Mean**	**HGN Loss 95% C.I.**
0.000001	2.34×10^{-3}	$[2.30, 2.38] \times 10^{-3}$	2.32×10^{-3}	$[2.28, 2.36] \times 10^{-3}$
0.0001	1.52×10^{-3}	$[1.48, 1.57] \times 10^{-3}$	1.54×10^{-3}	$[1.50, 1.58] \times 10^{-3}$
0.001	1.58×10^{-3}	$[1.55, 1.60] \times 10^{-3}$	1.58×10^{-3}	$[1.55, 1.60] \times 10^{-3}$

The next set of ablation studies continue to examine the efficacy of the hypernetwork in encoding and predicting GDN parameters in the test set.

Ablation of the Hypernetwork Encoding

Our work also guarantees that the hypernetwork can effectively encode a large number of GDNs. In particular, doing so suggests that the HGN model can be recursively rolled, allowing us to predict well into the future. Two questions naturally arise: 1) In practice, is a hypernetwork encoding of a sequence of GDN models legitimately trainable? 2) Does the hypernetwork continue to generate well-performing GDN models out-of-sample in future times? This section yields an affirmative *yes* to both of these questions; thus showing the feasibility and reliability of the hypernetwork in the HGN model.

In this next experiment, we test this by comparing three different degrees of hypernetwork encoding. These experiments are run with a subset of the same configurations from the last section; which we annotate in each figure caption. Each of the figures 3.3 and 3.4 plot the test set performance of each model, which begins at time $t = 160$ and ends at time $t = 200$.

(1) (GDN) no hypernetwork is used and only a different GDN "expert" is used to generate predictions at any time, trained at time $t-1$. (2) (HGN 1 step) at every time

t, the hypernetwork loads the predictions of the GDN at time $t-1$ and uses them to predict a GDN, which is then used to predict at time t. In this case, the hypernetwork component of the HGN is only ever used to make one-step-ahead predictions. (3) (HGN) loads the GDN parameters at time 160 and then sequentially predicts the parameters at every subsequent time t using its predicted parameters at time $t-1$, up until the terminal time $t = 200$. All three models perform nearly identically, as illustrated by the log-scale losses. This shows that the hypernetwork encoding of the GDN models is practically effective.

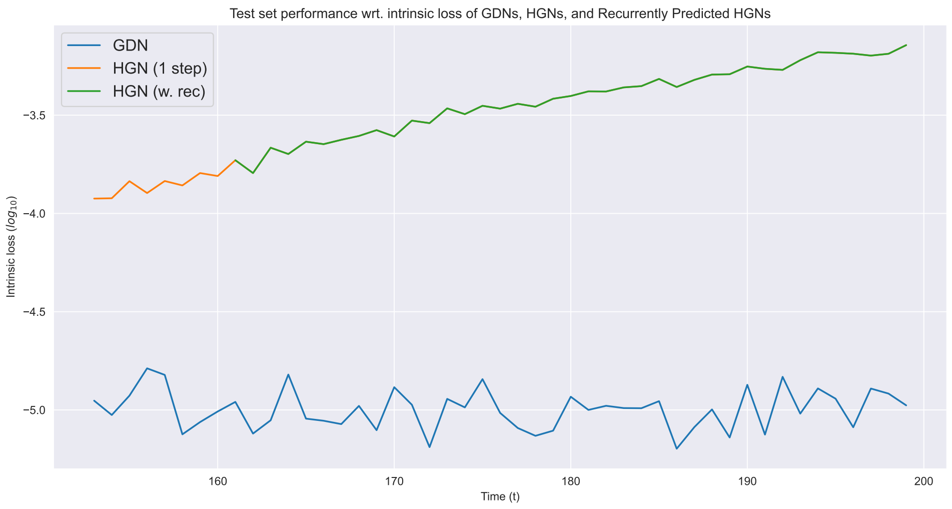


Figure 3.3: Situation I - Nearly Logarithmic Degradation of HGN Accuracy

Our experiments uncover two types of behaviours that the HGN can exhibit. The first one is shown in Figure 3.3 above, the HGN performance slowly (logarithmically) departs from that of the GDN as time rolls forward. This is typically what is observed in most of our experiments. In this case, there is a small but growing gap between the test set performance of the HGN and the GDN model, which increases as time flows forward. Furthermore, this gap is roughly the same for both the 1-step and recurrent

HGN models.

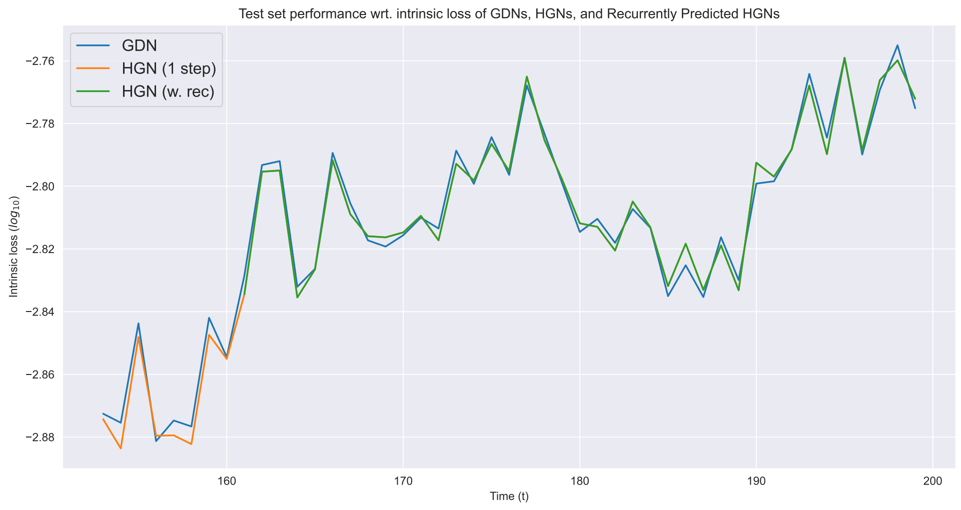


Figure 3.4: Situation II - Nearly Perfect GDN Prediction by HGN

The second scenario, can be seen in Figure 3.4, is when the HGN continues to nearly perfectly predict the performance of the GDN as time rolls forward. This occurs in a subset of experiments where the GDN parameters do not change significantly between time steps.

The GDN training occasionally suffers from exploding gradients during training; one can re-run the stochastic gradient descent algorithm when this happens. We note that there is nothing particular about instances when this happens (see Appendix 3.6.4).

3.5 Conclusion

We presented a framework for obtaining low-dimensional approximations of the conditional distribution (non-Markovian) stochastic Volterra processes in discrete time. First, we develop a tool, the Gaussian projection, for projecting the condition of such processes down onto the C^∞ Riemannian manifold $\mathcal{N}_d$ of non-singular Gaussian measure with a perturbation $\mathfrak{J}$ of its standard information geometry with favourable geometric and computational properties. Like classical tools for dimension reduction of probability measures (e.g. information projections), the Gaussian projection is a projection-type optimization problem; however, unlike those tools, the Gaussian projection is a Lipschitz operation and can even be a C^∞ map under additional conditions. Using these insights, we then constructed a sequential geometric deep learning model which is compatible with the non-positive curvature of $(\mathcal{N}_d, \mathfrak{J})$. We also numerically illustrated the HGN model, showing its practical viability. We conducted an ablation study, confirming our main theoretical results (which are available in the full paper).

Future Research

In future work, we aim to extend our analysis to general stochastic processes, thereby going beyond the stochastic Volterra setting. We would also like to explore the impact of projecting onto different information-like geometries when approximating the conditional law of various processes, and how to choose such geometries if one has information on the structure of these processes.

Besides that, we consider replacing our ReLU MLP backbones with other deep learning architectures such as Kolmogorov-Arnold networks (KAN) and modifying our proofs with that backbone's approximation theorem. For instance, using a KAN we can use the approximation theorem of [33], using MLPs with different activation functions, e.g. ReLU^k or smooth activation functions, we can respectively rely on the results of [38] or of [49], or using shallow neural networks we can appeal to the optimal rates of [47]. Each of these is an interesting avenue of possible future research.

3.6 Appendix

Experiment Details

We include the details of the experiments in Section 3.3. We first explicitly describe the algorithms we used to generate the sample paths from the Volterra process in (3.3.1), to compute its Gaussian random projections, and to train the HGN from this data. We then provide details on the hardware used to train these models.

3.6.1 Experiment and Compute Details

The architecture used to tackle those problems (identical for all the models) had six layers with a maximum size of 512 in the GDN part and eight layers with a maximum size of 1024 in the hypernetwork (ignoring the input-output layers). Since the base parameter's T value was 200, we had to train 200 GDNs and 1 hypernetwork. Considering the architecture mentioned, we had around $500,000$ parameters in each GDN and $750,000,000$ in the hypernetwork to train. Despite their size, these calculations are highly parallelizable. We trained our models within a reasonable time by exploiting basic optimizations and employing the graphics processing unit (GPU) to run the computations within each model in parallel. There are still simple ideas that can improve the performance of these networks, like training GDNs in parallel using multiple GPUs simultaneously. However, there was no need to do that since our training time was short enough, and we would need more GPUs to achieve that.

We ran experiments on the Vector Institute for Artificial Intelligence's computing cluster. Since each problem (process created with a specific set of parameters) is entirely independent, we used 30 machines, one for each problem in parallel. All the machines had the same configuration with 6 CPU cores, 1 Nvidia T4 GPU, 20 GB of RAM, and 40 GB of SSD memory. The run-time limit for the instances was 12 hours, although all machines finished their jobs in less than 8.5 hours. The problem with the base parameter set took about 4 hours and 44 minutes, and the overall average was around 6 hours. Note that these differences might be seen because not all machines were on the same host computing node. Thus, the CPU and RAM models and clock frequencies different amount for all the machines.

3.6.2 Algorithm Descriptions

The following algorithm is used to generate sample paths of the Volterra process defined in (3.3.1).

Algorithm 1: Construct $\boldsymbol{X}$

```
Require: Number of training samples $N \in \mathbb{N}_+$, time-horizon $T \in \mathbb{N}_+$, dynamics $\mu$,
  $\sigma$, and $\varsigma$, "noise" parameter $0 \leq \lambda$, memory $1 \leq w \leq 0$.
 For $n : 0, \dots, N-1$ in parallel
   for $t : 0, \dots, T$ do
     if $t = 0$ then
       $x^n_{-1} \leftarrow 0$
       $x^n_0 \leftarrow 0$                                  // Get Initial States
       end if
       else
       Sample: $Z \sim N_d(0, I_d)$                          // Generate Gaussian Noise
       Sample: $B \sim \mathrm{Binom}(\{0,1\}; 1/2)$         // Generate Hidden Process
       $x \leftarrow w\mu(x^n_{t-1}) + (1-w)\mu(x^n_t)$      // Get Drift
       $y \leftarrow (\varsigma(x^n_t) \cdot \sigma + B\,\lambda\, I_d)Z$   // Get Diffusion
       $x^n_{t+1} \leftarrow x^n_t + x + y$                  // Update Diffusion
       end if
     $x^n \leftarrow (x^n_t)_{t=0}^T$                        // Save Sample Path
     end for
   end
 $\boldsymbol{X} \leftarrow \{x^n\}_{n=0}^{N-1}$                // Compile Dataset
 return $\boldsymbol{X}$
```

The next algorithm (Algorithm 2) implements the Gaussian projection of the Volterra process in (3.3.1). This is given by the closed-form expression derived in (3.3.3).

Algorithm 2: Given a set of paths $\mathbf{X}$ in $\mathbb{R}^{(2+T)d}$, this algorithm returns an array of input-output pairs, whose elements are pairs of paths $x_{[-1:T]}$ in $\mathbf{X}$ paired with the parameters determining the path of Gaussian distributions $y^x \overset{\text{def.}}{=} (\mathbb{Q}_{x_{[-1:t]}})_{t=0}^T$ traced out by sequentially applying the barycenter map to the process $(\mathbb{Q}_{x_{[-1:t]}})_{t=0}^T$.

```
Require: Time-Horizon, finite set of paths $\boldsymbol{X} \subseteq \mathbb{R}^{(2+T)d}$, drift $\mu$, diffusion
    parameters $\sigma$ and $\varsigma$, and a “randomness” $\lambda \geq 0$.
For $x \overset{\text{def.}}{=} x_{[-1:T]} \in \boldsymbol{X}$ in parallel
    for $t : 0, \dots, T$ do
        $\mu_t^x \leftarrow x_t + \text{Drift}(x_{t-1}, x_t)$    // Get Mean of Projection
        $\sigma_t^x \leftarrow \varsigma(x_t) \cdot \sigma^2(\sigma^{-2}(\lambda\, I_d + \varsigma(x_t) \cdot \sigma)^2)^{1/2}$    // Get Covariance of Projection
    end for
    $y^x \leftarrow \big(\mu_t^x, \text{vec}(\sigma_t^x)\big)_{t=0}^T$    // Get Outputs
end for
return $\boldsymbol{Z} \leftarrow \{(x, y^x)\}_{x \in \boldsymbol{X}}$    // Return Array of Input-Output Pairs $\{(x, y^x)\}_{x \in \boldsymbol{X}}$
```

Once the input data has been generated using Algorithm 1 and the corresponding Gaussian random projections have been computed using Algorithm 2, then we have input-output pairs which can be used to train the GDN and HGN models. Observe that the training scheme that we used for GDNs avoids backpropagating in time (as with RNNs). Thus, even if the HGN has a recursion, it can be trained without recursion similarly and thus enjoys some of the training stability properties of transformers, which RNNs do not share; namely, no backpropagation in time.

```
Algorithm 3: Construct HGN
Require: A dataset $\boldsymbol{Z} \leftarrow \{(x, y^x)\}_{x\in\boldsymbol{X}}$, GDN depth and widths, (ReLU)
    hypernetwork dimensions $[\mathbf{d}]$.
    For $t : 0, \dots, T$ in parallel
        $\hat{f}_{\theta_t} \leftarrow \underset{\hat{f}_\theta \in \mathcal{GDN}_{[S],[L]}}{\operatorname{argmin}} \sum_{x\in\boldsymbol{X}} \left\|\hat{f}_\theta(x_{[t-1:t]}) - y_t^x\right\|^2$    // Optimize GDN Nodes
        $z_t \leftarrow (\theta_t, t)$    // Separate Parameters
    end
    /* Create Recurrence/ Encode Causality */
    $\hat{h} \leftarrow \underset{h\in\mathcal{NN}_{[\mathbf{d}]}^{\mathrm{ReLU}}}{\operatorname{argmin}} \sum_{t=0}^{T} \|h(z_t) - z_{t+1}\|_2^2$
    /* Server receives parameters of optimized neural filters for each time window
    */
    $L : \mathbb{R}^{P([\mathbf{d}])} \times \mathbb{R}^Q \mapsto \mathbb{R}^{P([\mathbf{d}])}$ projection onto first component
    return Trained HGN: $(\hat{f}, z_0, L)$.
```

3.6.3 Additional Loss Curves

We plot the loss curves, in the test set, of a representative subset of the experiments reported in Tables 3.1, 3.4, 3.2, 3.5, and 3.6. Figure 3.5 shows that the behaviour illustrated in Figures 3.3 and 3.4 persists across our experiments.

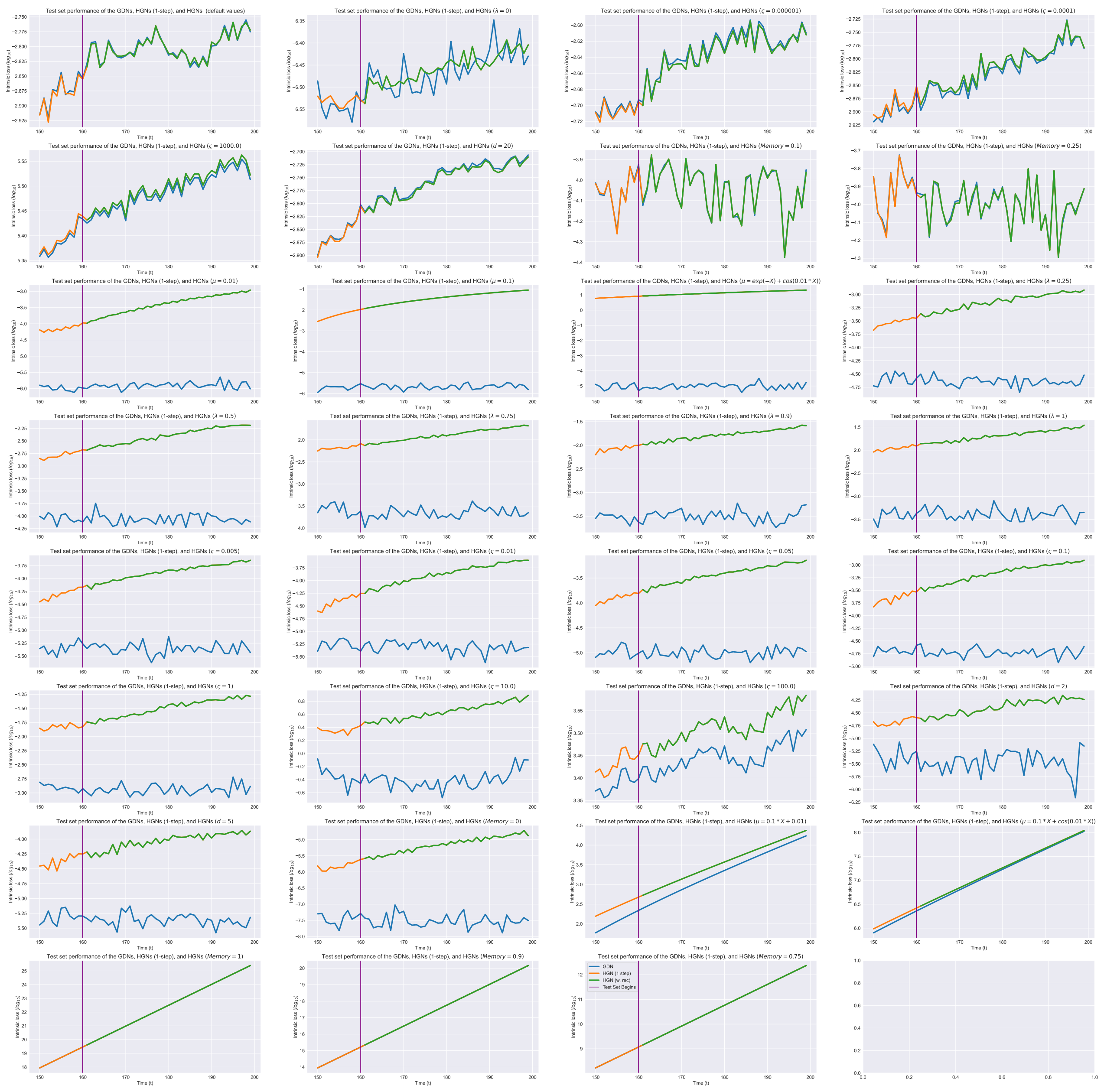


Figure 3.5: Typical Learning Curves - Including Cases With Exploding Gradients

3.6.4 Exploding Gradients due to Small Eigenvalues

There is one additional case, illustrated in Figure 3.5, where the GDN or HGN suffers from exploding gradients during training. We included some examples of learning curves illustrating the exploding gradients phenomenon that occurs.

A priori there is nothing particular about the target function being learned when this occurs. For instance, the experiments in Table 3.1 where $x \mapsto \frac{1}{2}(\frac{1}{100} - x)$ is essentially the same as when $x \mapsto \frac{1}{10}\,x + \frac{1}{100}$; thus both should be equally easy to learn. However, in the experiment recorded in Table 3.7, the model experiences exploding gradients during training and thus the reported loss is relatively large. Similarly, the drift $x \mapsto e^{-x} + \cos(\frac{x}{100})$ and $x \mapsto \frac{x}{10} + \cos(\frac{x}{100})$ are essentially the same; but again the latter is not being learned due to exploding gradients during training (see Table 3.7 again). Similarly, the value of w is similar to those considered in Table 3.4; however, the loss in situations where gradients exploded during training is several magnitudes larger.

Table 3.7: Examples of Exploding Gradients

μ	**GDN Loss Mean**	**GDN Loss 95% C.I.**	**HGN Loss Mean**	**HGN Loss 95% C.I.**
$\frac{1}{10}\,x + \frac{1}{100}$	$4.19 \times 10^{+3}$	$[2.75, 5.64] \times 10^{+3}$	$6.22 \times 10^{+3}$	$[4.24, 8.21] \times 10^{+3}$
$\frac{x}{10} + \cos(\frac{x}{100})$	$2.82 \times 10^{+7}$	$[1.92, 3.71] \times 10^{+7}$	$2.98 \times 10^{+7}$	$[2.05, 3.91] \times 10^{+7}$
w	**GDN Loss Mean**	**GDN Loss 95% C.I.**	**HGN Loss Mean**	**HGN Loss 95% C.I.**
0.75	$3.32 \times 10^{+11}$	$[1.55, 5.09] \times 10^{+11}$	$3.32 \times 10^{+11}$	$[1.55, 5.09] \times 10^{+11}$
0.9	$1.42 \times 10^{+19}$	$[0.45, 2.40] \times 10^{+19}$	$1.42 \times 10^{+19}$	$[0.45, 2.40] \times 10^{+19}$

A closer look at the error logs shows that the exploding gradient occurs due to rounding errors in the Riemannian distance function when the logarithm is applied to small eigenvalues of the relevant positive-definite matrix. Though gradient clipping typically solves this issue, it occasionally resurfaces, and we thus report it here.

Bibliography

[1] E. Abi Jaber, C. Cuchiero, L. Pelizzari, S. Pulido, and S. Svaluto-Ferro. Polynomial volterra processes. *Electronic Journal of Probability*, 29:1–37, 2024.

[2] E. Abi Jaber, M. Larsson, and S. Pulido. Affine Volterra processes. *The Annals of Applied Probability*, 29(5):3155–3200, 2019.

[3] B. Acciaio, A. Kratsios, and G. Pammer. Designing universal causal deep learning models: The geometric (hyper) transformer. *Mathematical Finance*, 34(2):671–735, 2024.

[4] F. Aichinger and S. Desmettre. Utility maximization in multivariate Volterra models. *SIAM Journal on Financial Mathematics*, 14(1):52–98, 2023.

[5] S. Ainsworth, J. Hayase, and S. Srinivasa. Git Re-Basin: Merging models modulo permutation symmetries. In *The Eleventh International Conference on Learning Representations*, 2023.

[6] F. E. Benth, N. Detering, and L. Galimberti. Neural networks in Fréchet spaces. *Annals of Mathematics and Artificial Intelligence*, 91(1):75–103, 2023.

[7] J.-M. Bernardo and A. F. M. Smith. *Bayesian theory.* Wiley Series in Probability

and Mathematical Statistics: Probability and Mathematical Statistics. John Wiley & Sons, Ltd., Chichester, 1994.

[8] D. A. Bini and B. Iannazzo. Computing the karcher mean of symmetric positive definite matrices. *Linear Algebra and its Applications*, 438(4):1700–1710, 2013.

[9] C. M. Bishop. Mixture density networks. *Aston University*, 1994.

[10] A. Bondi, G. Livieri, and S. Pulido. Affine Volterra processes with jumps. *Stochastic Processes and their Applications*, 168:104264, 2024.

[11] A. Borovykh, S. Bohte, and C. W. Oosterlee. Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*, 2017.

[12] T. Chen and H. Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks and Learning Systems*, 6(4):911–917, 1995.

[13] I. Chevyrev and H. Oberhauser. Signature moments to characterize laws of stochastic processes. *Journal of Machine Learning Research*, 23(176):1–42, 2022.

[14] M. N. R. Chowdhury, S. Zhang, M. Wang, S. Liu, and P.-Y. Chen. Patch-level routing in mixture-of-experts is provably sample-efficient for convolutional neural networks. In *International Conference on Machine Learning*, pages 6074–6114. PMLR, 2023.

[15] C. Cuchiero, P. Schmocker, and J. Teichmann. Global universal approximation of functional input maps on weighted spaces. *arXiv preprint arXiv:2306.03303*, 2023.

[16] C. Cuchiero and J. Teichmann. Generalized Feller processes and Markovian lifts of stochastic Volterra processes: the affine case. *Journal of Evolution Equations*, 20(4):1301–1348, 2020.

[17] W. Fedus, B. Zoph, and N. Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

[18] J. Gauthier. Conditional generative adversarial nets for convolutional face generation. *Class project for Stanford CS231N: convolutional neural networks for visual recognition, Winter semester*, 2014(5):2, 2014.

[19] L. Gonon, L. Grigoryeva, and J.-P. Ortega. Reservoir kernels and Volterra series. *arXiv preprint arXiv:2212.14641*, 2022.

[20] L. Gonon and J.-P. Ortega. Reservoir computing universality with stochastic inputs. *IEEE Transactions on Neural Networks and Learning Systems*, 31(1):100–112, 2019.

[21] L. Gonon and J.-P. Ortega. Fading memory echo state networks are universal. *Neural Networks*, 138:10–13, 2021.

[22] Google. Gemini. Google, 2024.

[23] L. Grigoryeva and J.-P. Ortega. Universal discrete-time reservoir computers with stochastic inputs and linear readouts using non-homogeneous state-affine systems. *Journal of Machine Learning Research*, 19(24):1–40, 2018.

[24] L. Grigoryeva and J.-P. Ortega. Differentiable reservoir computing. *Journal of Machine Learning Research*, 20(179):1–62, 2019.

[25] D. Ha, A. M. Dai, and Q. V. Le. Hypernetworks. In *International Conference on Learning Representations*, 2017.

[26] C. Hutter, R. Gül, and H. Bölcskei. Metric entropy limits on recurrent neural network learning of linear dynamical systems. *Applied and Computational Harmonic Analysis*, 59:198–223, 2022.

[27] A. Jacquier, C. Martini, and A. Muguruza. On VIX futures in the rough bergomi model. *Quantitative Finance*, 18(1):45–61, 2018.

[28] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. d. l. Casas, E. B. Hanna, and F. Bressand. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[29] M. J. Korenberg and I. W. Hunter. The identification of nonlinear biological systems: Volterra kernel approaches. *Annals of Biomedical Engineering*, 24:250–268, 1996.

[30] Y. Korolev. Two-layer neural networks with values in a Banach space. *SIAM Journal on Mathematical Analysis*, 54(6):6358–6389, 2022.

[31] A. Kratsios. Universal regular conditional distributions via probabilistic transformers. *Constructive Approximation*, 57(3):1145–1212, 2023.

[32] A. Kratsios, H. S. d. O. Borde, T. Furuya, and M. T. Law. Approximation rates and VC-dimension bounds for (P)ReLU MLP mixture of experts. *arXiv preprint arXiv:2402.03460*, 2024.

[33] A. Kratsios and T. Furuya. Kolmogorov-arnold networks: Approximation

and learning guarantees for functions and their derivatives. *arXiv preprint arXiv:2504.15110*, 2025.

[34] A. Kratsios, T. Furuya, J. A. L. Benitez, M. Lassas, and M. de Hoop. Mixture of experts soften the curse of dimensionality in operator learning. *arXiv preprint arXiv:2404.09101*, 2024.

[35] S. Lanthaler and A. M. Stuart. The parametric complexity of operator learning. *arXiv preprint arXiv:2306.15924*, 2023.

[36] P. Li, Z. Zhang, P. Yadav, Y.-L. Sung, Y. Cheng, M. Bansal, and T. Chen. Merge, then compress: Demystify efficient SMoe with hints from its routing policy. In *The Twelfth International Conference on Learning Representations*, 2024.

[37] Z. Li, J. Han, E. Weinan, and Q. Li. Approximation and optimization theory for linear continuous-time recurrent neural networks. *Journal of Machine Learning Research*, 23:42–1, 2022.

[38] T. Mao and D.-X. Zhou. Rates of approximation by relu shallow neural networks. *Journal of Complexity*, 79:101784, 2023.

[39] M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[40] Y. Mishura, S. Ottaviano, and T. Vargiolu. Gaussian Volterra processes as models of electricity markets. *SIAM Journal on Financial Mathematics*, 15(4):989–1019, 2024.

[41] J. Puigcerver, C. R. Ruiz, B. Mustafa, and N. Houlsby. From sparse to soft

mixtures of experts. In *The Twelfth International Conference on Learning Representations*, 2024.

[42] E. M. Saad, N. Verzelen, and A. Carpentier. Active ranking of experts based on their performances in many tasks. In *International Conference on Machine Learning*, pages 29490–29513. PMLR, 2023.

[43] R. Saqur, A. Kratsios, F. Krach, Y. Limmer, J.-J. Tian, J. Willes, B. Horvath, and F. Rudzicz. Filtered not mixed: Stochastic filtering-based online gating for mixture of large language models. *arXiv preprint arXiv:2406.02969*, 2024.

[44] E. Sharma, D. Kwok, T. Denton, D. M. Roy, D. Rolnick, and G. K. Dziugaite. Simultaneous linear connectivity of neural networks modulo permutation. *arXiv preprint arXiv:2404.06498*, 2024.

[45] S. B. Shiki, S. Da Silva, and M. D. Todd. On the application of discrete-time Volterra series for the damage detection problem in initially nonlinear systems. *Structural Health Monitoring*, 16(1):62–78, 2017.

[46] K.-T. Sturm. Probability measures on metric spaces of nonpositive curvature. In *Heat kernels and analysis on manifolds, graphs, and metric spaces (Paris, 2002)*, volume 338 of *Contemp. Math.*, pages 357–390. Amer. Math. Soc., Providence, RI, 2003.

[47] Y. Yang and D.-X. Zhou. Optimal rates of approximation by shallow relu k neural networks and applications to nonparametric regression. *Constructive Approximation*, pages 1–32, 2024.

[48] C. Yun, S. Bhojanapalli, A. S. Rawat, S. Reddi, and S. Kumar. Are transformers universal approximators of sequence-to-sequence functions? In *International Conference on Learning Representations*, 2020.

[49] S. Zhang, J. Lu, and H. Zhao. Deep network approximation: Beyond relu to diverse activation functions. *Journal of Machine Learning Research*, 25(35):1–39, 2024.

Chapter 4

Conclusion

In this thesis, I have taken a look at the intersection of geometric deep learning, foundation models, and mathematical finance, and focused on addressing current practical computational challenges in large language model adaptation and non-Markovian stochastic process approximations. As seen in the two primary contributions, Chapters 2 and 3, I have demonstrated how geometry-aware frameworks can tackle fundamental problems such as the curse of dimensionality and enhance accessibility, scalability, and performance in real-world applications.

As presented in Chapter 2, the first contribution, I introduced a CPU-efficient meta-generation framework for fine-tuning large language models using low-rank adapters (LoRA) without using any GPU computation in a reasonable amount of time. The proposed pipeline leverages distributional alignment between a given new dataset and a bank of other datasets with known optimal adapters, enabling zero-shot prediction of adapter parameters through three different lightweight methods: attentional, normalized, and neural approaches. In our extensive experiments on over 500 datasets using Mistral-7B, these theoretically proven frameworks have successfully outperformed the raw base model in more than 97% of the datasets while recovering nearly half of the performance gap (on average) between untuned base models and fully fine-tuned ones. Being light enough to run entirely on ordinary hardware, this framework promises privacy-preserving adaptation, reducing memory and energy consumption, and helping to make the specialized LLMs more accessible for users without high-end infrastructure. At the end of the day, this work not only advances parameter-efficient fine-tuning under hardware constraints but also highlights the potential of reusing a droplet of the huge amount of public information on foundation models, hopefully taking a small step towards more sustainable AI practices.

The second contribution, Chapter 3, tackled the computational intractability of history-dependent Volterra processes, which are commonly used for modeling some of the most important aspects of financial markets, such as volatility prediction and asset prices, due to their non-Markovian behaviors. As a workaround for the main problem, the curse of dimensionality, I proposed a two-step framework in detail: first, to project the infinite-dimensional conditional law onto a low-dimensional manifold of non-positive curvature, and second, to train local experts followed by a hyper-geometric network (HGN) overlay that adapts these geometric deep neural networks (GDNs) over time. Our ablation study across parameters like drift (μ), randomness (λ), dimension (d), memory persistence (w), fluctuations (ς), and curvature further confirmed the model's robustness. The results showed that the HGN-forecasted GDNs often follow the trained GDNs' performance closely, or near-perfectly, if considering the inherent numerical instabilities of calculations and mathematical operations on computers. This achievement of overcoming the curse of dimensionality by using geometric deep learning not only enables the efficient simulation and estimation of

Volterra processes but also underscores the value of looking through a geometry-aware lens when facing interdisciplinary problems and the doors that can be opened by mathematical tools.

These contributions illustrate a unified theme: approximating or predicting model parameters via higher-level, geometry-informed architectures to solve downstream problems efficiently. Borrowing and combining ideas from different fields, namely, machine learning, mathematics, and quantitative finance, in this thesis, I advance efficient methodologies that are not only theoretically sound but also practically feasible, as supported by mathematical proofs, algorithmic implementations, and extensive empirical experimentation.

Although these models are demonstrating strong results, like any other model, they have limitations. During this research, we have done our best to look from a critical point of view to our work and be the first one to see the caveats and weaknesses of our work, many of which have been addressed and solved, and many remain as future research opportunities.

The LoRA meta-generation pipeline, for instance, relies on the availability of a bank of pre-trained adapters and datasets and may underperform on highly idiosyncratic datasets that lie far from the distribution of the datasets with known optimal adapters. Other places that have room for improvement are the tokenization procedure and the neural network architecture used in the neural approach. Some promising ideas would be using more complex methods to encode the information of each dataset or using other neural network architectures, such as attention mechanisms or graph neural networks, to better predict the combination ratios, which can potentially improve the quality of the outputs due to the simplicity of the version we used. Similarly, the Volterra approximation assumes a specific manifold structure, which may require further refinement for processes with extreme non-stationarities or higher-dimensional complexities. Besides this, the same module simplicity, as seen in the LoRA framework, can be seen in both the GDNs and the HGN used for the Volterra approximation pipeline. The models used in that study have only a few layers and lack architectural complexity. As a result, expecting enhancement in predictive power by exploring other advanced architectures is not far from reality.