Towards Efficient and Performant Generative AI Inference

Submitted in partial fulfillment of the requirements for
the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Harry Dong

B.A., Computer Science, University of California, Berkeley
B.A., Statistics, University of California, Berkeley

Carnegie Mellon University
Pittsburgh, PA
April 2026

Acknowledgements

The work in this thesis was fortunate to be supported in part by the Liang Ji-Dian Graduate Fellowship; the Michel and Kathy Doreau Graduate Fellowship in Electrical and Computer Engineering; the Wei Shen and Xuehong Zhang Presidential Fellowship at Carnegie Mellon University; the Air Force Research Laboratory under FA8750-20-2-0504; the Carnegie Mellon University Manufacturing Futures Initiative, made possible by the Richard King Mellon Foundation; the Department of Transportation under 693JJ321C000013; NSF under CAREER ECCS-1818571, DMS-2134080, and ECCS-2126634; ONR under N00014-19-1-2404; and the CIT Dean's Fellowship.

...

Roughly 99% of this thesis covers the work I have done throughout my Ph.D. Though it may come as a surprise, I did not in fact spend 99% of my waking hours over the past nearly five years doing research at my desk (the true percentage is a secret known only by myself). However, what is not a secret is that a Ph.D. is not a solo endeavor. Often by serendipity, I have met so many wonderful people who make me feel luckier than anything I could imagine. So, I would like to dedicate this first 1% of my thesis to the people who have made the remaining 99% possible.

First, I would like to thank my advisor and committee chair, *Prof. Yuejie Chi*. Your guidance, expertise, and understanding have been indispensable as I navigated various phases of my Ph.D. Each interaction with you inspired me with new ideas and greater motivation, culminating in the mindset and work I am proud of today. If I were to do it all over again, I would always want you to be my advisor.

Thank you, *Prof. Beidi Chen*, a down to earth mentor and collaborator since my early Ph.D. years. You helped bring me into a field I knew little about, completely transforming the trajectory of my Ph.D. for the better. You genuinely wanted me to succeed, and because of that, I have come to bother you more times than I can remember for advice on research, careers, and life.

To the remaining members of my thesis committee, *Dr. Bilge Acun* and *Prof. Andrea Zanette*, thank you. I appreciate the time you have spent to improve my research and thesis. Our conversations are always productive, and I will continue learning from you both.

I am also grateful to my incredible colleagues. I would like to thank my labmates (*Prof. Harlin Lee, Prof. Vince Monardo, Dr. Tian Tong, Dr. Boyue Li, Prof. Laixi Shi, Dr. Shicong Cen, Dr. Pedro Valdeira, Prof. Zhize Li, Dr. Sudeep Salgia, Diogo Cardoso, Jiin Woo, Lingjing Kong, Zixin Wen, He Wang, Xingyu Xu, Tong Yang, and Timofey Efimov*) for expanding my horizons to problems and solutions I would have never thought about. Thank you to the InfiniAI Lab members (*Haizhong Zheng, Ranajoy Sadhukhan, Zhuoming Chen, Yang Zhou, Hongyi Liu, Krish Agarwal, Hanshi Sun, and Vash Tiwari*) for letting me bug you with my never-ending questions and informally adopting me into your lab.

Additional thanks to all the professors for whom I have had the pleasure to teach with (*Prof. José Moura, Prof. Soummya Kar, and Prof. Aswin Sankaranarayanan*).

I was lucky to have my Ph.D. years enriched by various internships that have taken me to new cities. Thank you to my intern manager at Meta, *Dr. Karthik Sankararaman*, for mentoring me from the California while worked from NYC. The things I learned from you extend well beyond my research. Thank you, *Dr. Tyler Johnson and Dr. Emad Soroush*, for a wonderful time on your team at Apple in Seattle. You showed me how AI development works on the industrial scale. Thank you to *Dr. Megna Shah, Dr. Sean Donegan, and Dr. Jeff Simmons* for hosting me at Wright-Patterson Air Force Base and being adaptive through our years of collaboration in a new domain. You all have given me a deeper appreciation of the engineering that powers some of the world's most complex machines.

Thank to all my pre-Ph.D. mentors who took a chance on me, setting me on the path of research: *Dr. Théophile Cabannes, Prof. Alexandre Bayen, Prof. Roy Ben-Shalom, Dr. Jan Balewski, Prof. Jonathan Fischer, Prof. Yun Song, and Ms. McKay.* You all gave me firsthand experience with the intricacies of your respective fields, lessons I still cherish in my interdisciplinary work today.

Since these margins are too narrow to contain a complete list of everyone and everything else deserving of thanks, I would like to make a few shout-outs. Thank you to my friends from day one at CMU, *Dr. Akhil Padmanabha, Arjun Ramesh, Tianshu Huang, Eric Giuffrida, Jessy Ha, Eric Xu, and Dr. Nicolas Gratius*, who all immediately made Pittsburgh feel like home. Thank you to *Dereje Shenkut, Dr. John Shi, Shreyas Chaudhari, Pranav Srinivasa, Dr. Mark Lindsey, Dr. Sudeep Salgia, Dr. Arjun Ramesh, Timofey Efimov, and the CMU Ballroom team* for keeping me active whether it be running, racquetball, tennis, weightlifting, calisthenics, biking, dancing, or trying new sports. Thank you, *Dr. Tyler Vuong and Dr. Roshan Sharma*, for being early influences on defining my research direction that eventually led to this thesis. To *Helen Tang, Dr. Anand Siththaranjan, Richard Chen, Aditya Iyer, Ellen Nguyen, Colby Leiske, Tobe To, and Wyatt Dias*, thank you for consistently keeping in touch despite the distance and reminding me how quickly

time passes. Additional thanks to all the friends and colleagues who made my internships at AFRL, Apple, and Meta unforgettable; the CMU ECE administrative staff who have made my life easier; my favorite third spaces; all the Porter B regulars; and all the 621 revelers.

And most of all, I would like to thank my family whose unconditional support got this all started.

Harry Dong

Abstract

Rapid progress in generative artificial intelligence (AI) has taught models, namely large language models (LLMs), surprising capabilities, which drive the spread of their potential to various domains and applications. Yet, inefficiencies in various components of the architecture and inference pipeline limit broader deployment. The demanding nature of model inference can harm latency, throughput, and memory. Fortunately, a close look at the model architecture, data, intermediate features, and hardware behavior reveal emergent patterns and redundancy that provide clues to optimize inference for efficiency and performance.

This thesis takes an end-to-end approach to attain efficient and performant generative AI inference in a variety of settings and their associated bottlenecks following common themes of sparsity, low-rank structures, and parallelism. Starting with bottlenecks of common LLM settings, we propose methods to reduce computational footprints in key-value caches and model weights, allowing for low latency and high throughput generation. In addition, we explore inference optimizations in reasoning tasks, where performance can be brittle and scale is often necessary, by introducing small recovery modules and a generalization of parallel LLM inference. Finally, we take the lessons in these works on textual data and carry them over to generative modeling in the materials science domain to accelerate and improve collection of an expensive imaging modality. Together, our methods and insights push towards the compute-performance Pareto frontier for AI inference across a diverse set of applications.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Since the introduction of transformer architecture (Vaswani et al., 2017), the field of artificial intelligence (AI) has seen astronomical progress and an explosion of downstream applications in areas such as computer vision, speech processing, and healthcare. Perhaps above all, the impact of AI can be felt most heavily in natural language processing (NLP), where transformers greatly benefit from scaling to billions—sometimes even trillions—of parameters and training on enormous datasets (Hoffmann et al., 2022; Kaplan et al., 2020), sparking the rise of powerful large language models (LLMs) with most having transformers at their cores (Achiam et al., 2023; Anthropic, 2024; Dubey et al., 2024; Guo et al., 2025; Team et al., 2024c). Although scale enhances performance, optimizations in resource utilization are sidelined. This comes with high inference costs which have negative implications on applicability, accessibility, and energy consumption (Samsi et al., 2023; Strubell et al., 2020). While reducing compute without harming performance is one direction we can take, understanding what is important for performance can also hint at ways to scale compute more effectively. Consequently, there is an inherent tradeoff between inference efficiency and performance, so *we aim to maximize performance with every bit of compute we are given.* AI models are gaining new capabilities by the day, but we also need to ensure they are are easy and cheap to use.

We recognize that AI "efficiency" and "performance" are loaded terms. Perhaps most familiarly, an efficient model can be measured by metrics like latency, throughput, and

memory consumption in which strategies that reduce resource overutilization are especially nifty. Adding on, *resource underutilization*, such as suboptimal use of hardware and information, can also be an issue, and hence further contributes to a model's inefficiencies. On the other hand, performance, as referred to in this thesis, is a general evaluation of a model's capabilities. Commonly, this includes minimizing and maximizing certain metrics for a broad class of tasks. In addition, qualitative measures, especially useful when evaluations cannot be boiled down to a few numbers, will also provide valuable insight into performance.

With an initial cursory glance at modern AI models likes LLMs, the bottlenecks that limit inference efficiency and performance are amorphous. Deployment settings for AI models are exceptionally heterogeneous, even though the model architectures are similar. For instance, the expectations and challenges of hosting a model on a server are not the same as local inference on personal devices. Moreover, applications like question answering, agents, and scientific analysis will all need special domain-specific considerations. Though daunting at first, a decomposition of the facets of inference reveals common denominators that allow us to take surgical approaches to improve efficiency and performance.

1.1 The Many Facets of AI Inference

Each work in this thesis explicitly and purposefully leverages information from various facets that influence LLM inference and provide clues for effective algorithmic design. Also depicted in Figure 1.1, these facets include:

- **Architecture:** The model architecture, including parameters and their evolution throughout training, can determine how information is learned and stored. Choices like attention mechanisms and model width have directly influence performance and efficiency.
- **Data:** Helpful patterns, trends, and redundancy in data exist in specific tasks and in general. Often, one domain has unique constraints that call into question assumptions in another domain, requiring a redesign from first principles.

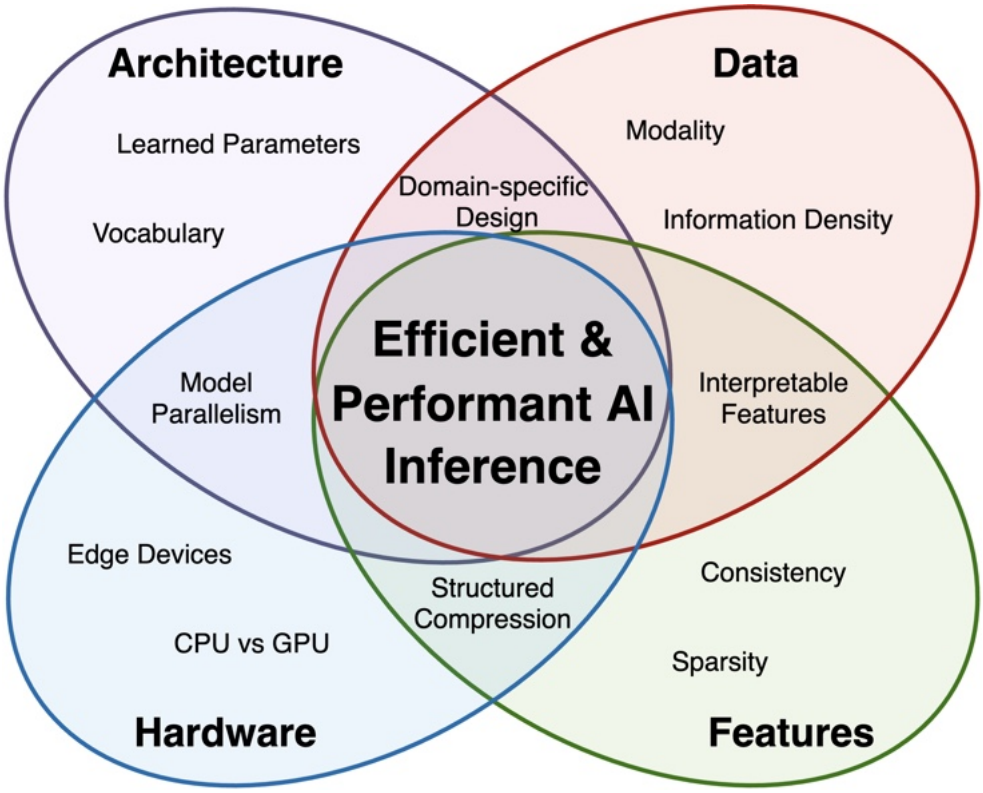

Figure 1.1: Facets of efficient and performant AI inference that guide the design of our methods.

- **Features:** With a peek under the hood, hidden states contain information that the model uses. They can have distinct sparse and low-rank structure that exist in various forms which can come in handy.
- **Hardware:** Surprises await as algorithms turn into physical operations on hardware. By taking a full stack view, understanding and leveraging hardware behavior can guide more effective algorithmic design, such that improvements on paper translate to tangible benefits in the real world.

Inference is a combination of these facets, so it is imperative to think about how they interact and synthesize with each other. As we will see throughout this thesis, our methods will optimize in the intersection of multiple or all facets as we take a holistic approach to efficient and performant inference.

1.2 Contributions and Organization

In this thesis, with a strong emphasis on LLMs, **we propose methods to ensure AI models use allocated compute to the maximum potential across domains for efficient and performant inference**. Between cutting wasteful operations and exploiting underutilized resources, we push inference towards the compute-performance Pareto frontier. Depending on the use case, different challenges and requirements will help guide the design of our methods. In other words, we ground our methods in reality by recognizing and adapting to the nuances and idiosyncrasies of various applications of AI. Beyond the shared goal of optimizing inference, the core observations and approaches of the works presented in this thesis also follow common themes of **sparsity**, **low-rank**, and **parallelism**. These illuminate powerful and intuitive cornerstones that our algorithms are built upon.

Much of this thesis improves LLM autoregressive inference which powers many popular AI-integrated products like chatbots, agents, and search engines, yet our insight goes beyond NLP. Text is a topical testing ground and what many frontier AI models are trained on, but it is not the only place where inference optimizations are needed, as we will see later in this thesis when we explore science applications.

This thesis is organized as follows:

- Chapter 1: For the remainder of this chapter, we define notation and terminology that we will abide by throughout this thesis.
- Chapter 2: We cover a brief overview of transformers while using notation that we will continue to use in the following chapters and provide relevant works that improve LLM inference.
- Chapter 3: Next, we dive into `LESS`, our novel hybrid KV cache compression method low-latency and high-throughput generation for long sequences (Dong et al., 2024c).
- Chapter 4: We introduce `GRIFFIN`, our simple adaptive structured pruning method to reduce latency by leveraging a surprisingly pervasive and consistent phenomenon called flocking (Dong et al., 2024a).
- Chapter 5: Looking at the limits associated with efficient methods applied to LLM

reasoning, our efficient low-rank method `Caprese` significantly or completely recovers lost reasoning performance from existing compression methods (Dong et al., 2026a).

- Chapter 6: Then, we propose `Bridge`, a parallel inference scaling method that shares information between sequences mid-generation to prevent information wastage for high quality reasoning response sets (Dong et al., 2026b).
- Chapter 7: With insights from our methods and observations in NLP, we introduce scalable deep learning algorithms for materials microscopy (Dong et al., 2023b).
- Chapter 8: Finally, we wrap up with a macroscopic view of our work and describe future directions in the field.

1.3 Notation

Throughout this thesis, we adopt consistent notational choices. We denote constants with unstylized letters (e.g., k, β, and S). Vectors, which are assumed to be row vectors unless stated otherwise, are denoted by bold lowercase letters (e.g., $\boldsymbol{x}$). We use bold uppercase letters for matrices (e.g., $\boldsymbol{W}$). We define tensors as an N-th order array where $N \geq 3$, which are denoted with bold uppercase calligraphic letters (e.g., $\boldsymbol{\mathcal{X}}$). Note that vectors and matrices are cases when the order N is 1 and 2, respectively. Maintaining the same arrangement of axes, indices of these structures are indicated with subscripts outside square brackets (e.g., $[\boldsymbol{Y}]_i$ and $[\boldsymbol{Y}]_{\cdot,j}$) and sometimes also with colons to capture ranges (e.g., $[\boldsymbol{Y}]_{i:i+k}$). We describe sets with unbolded uppercase calligraphic letters (e.g., $\mathcal{D}$). Other artifacts, such as functions and distributions, are wildcards but will be made clear in the given context.

1.4 Terminology

The field of AI is rapidly evolving with new ideas. To iron out potential misunderstandings in this thesis, we use the following common terminology and assumptions below for brevity or consistency with current literature.

Transformers and LLMs. We refer to decoder-only transformers as simply "transformers" *with the exception of Chapter 7* (which we will also remind in that chapter). This simplification is because the decoder transformer is the main architecture of current LLMs by far. Hence, this thesis also assumes LLMs are decoder transformers.

Attention. Attention blocks operate differently depending on the model. In the context of (decoder) transformers, "attention" refers to causal or masked self-attention where tokens cannot peek into future tokens. In the context of encoder transformers, "attention" refers to full bidirectional self-attention. Note that the use of attention actually predates transformers (Bahdanau et al., 2015; Cheng et al., 2016).

Dimension vs. Axis vs. Order. An N-th order or N-D tensor is an array with N axes. For example, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{D_1 \times \cdots \times D_N}$ is an N-th order tensor where the dimension of the n-th axis is D_n for $1 \leq n \leq N$. A high-order tensor is one where $N \geq 3$ while a high dimension refers to a large number of entries D_n along an axis index n. "Dimension" and "feature size" are commonly used interchangeably in deep learning.

What is Reasoning? The exact definition of reasoning in the context of AI is a debated topic, sometimes venturing into psychology and philosophy. This thesis offers no fresh perspective or stance on this debate. Instead, we follow precedent in recent relevant literature, using "reasoning" for models or tasks that use or require logical deduction, akin to System-2 thinking for humans (Kahneman, 2011). In many works, this includes tasks like math problems, agents, and puzzles, excluding tasks like information retrieval, language modeling, and straightforward arithmetic.

1.5 Relevant Publications

The following works are comprehensively explored in this thesis ordered by publication chronology:

- Dong et al. (2023b): **A Lightweight Transformer for Faster and Robust EBSD Data Collection**, Harry Dong, Sean Donegan, Megna Shah, Yuejie Chi, *Scientific Reports*, 2024.
- Dong et al. (2024c): **Get More with LESS: Synthesizing Recurrence with KV Cache Compression for Efficient LLM Inference**, Harry Dong, Xinyu Yang, Zhenyu Zhang, Zhangyang Wang, Yuejie Chi, Beidi Chen, *International Conference on Machine Learning (ICML)*, 2024.
- Dong et al. (2024a): **Prompt-prompted Adaptive Structured Pruning for Efficient LLM Generation**, Harry Dong, Beidi Chen, Yuejie Chi, *Conference on Language Modeling (COLM)*, 2024.
- Dong et al. (2026a): **Scalable LLM Reasoning Acceleration with Low-rank Distillation**, Harry Dong, Bilge Acun, Beidi Chen, Yuejie Chi, *Conference on Parsimony and Learning (CPAL)*, 2026.
- Dong et al. (2026b): **Generalized Parallel Scaling with Interdependent Generations**, Harry Dong, David Brandfonbrener, Eryk Helenowski, Yun He, Mrinal Kumar, Han Fang, Yuejie Chi, Karthik Abinav Sankararaman, *International Conference on Learning Representations (ICLR)*, 2026.
- Dong et al. (2026c): **Multimodal Diffusion to Mutually Enhance Polarized Light and Low Resolution EBSD Data**, Harry Dong, Timofey Efimov, Megna Shah, Jeff Simmons, Sean Donegan, Marc De Graef, Yuejie Chi, *under review*, 2026.

Moreover, the above works draw heavily from observations, insights, and results of our earlier works:

- Dong et al. (2023a): **Towards Structured Sparsity in Transformers for Efficient Inference**, Harry Dong, Beidi Chen, Yuejie Chi, *ICML Workshop on Efficient Systems for Foundation Models*, 2023.
- Dong et al. (2023c): **Deep Unfolded Tensor Robust PCA with Self-supervised Learning**, Harry Dong, Megna Shah, Sean Donegan, Yuejie Chi, *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2023.

- Dong et al. (2023d): **Fast and Provable Tensor Robust Principal Component Analysis via Scaled Gradient Descent**, Harry Dong, Tian Tong, Cong Ma, Yuejie Chi, *Information and Inference: A Journal of the IMA*, 2023.
- Efimov et al. (2025): **Leveraging Multimodal Diffusion Models to Accelerate Imaging with Side Information**, Timofey Efimov, Harry Dong, Megna Shah, Jeff Simmons, Sean Donegan, Yuejie Chi, *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2025.

Chapter 2

Preliminaries

Starting off with an overview of the transformers (Section 2.1) and related work on AI efficiency and inference scaling (Section 2.2), this chapter sets the stage for the remainder of this thesis.

2.1 Background

In this section, we delineate the standard transformer architecture, along with its training and inference setup. Most of this thesis concerns the decoder transformer, the backbone of most LLMs, so we begin with this architecture in Section 2.1.1. Adjustments for encoder transformers are discussed in Section 2.1.4 which are more relevant to Chapter 7.

2.1.1 Decoder Transformer Architecture

Illustrated in Figure 2.1, the decoder transformer (Radford et al., 2018) is the choice for numerous state-of-the-art LLMs, such as the Llama (Dubey et al., 2024), Gemma (Team et al., 2024c), and Qwen (Yang et al., 2025a) series.[1] The original architecture was proposed by Vaswani et al. (2017) and has remained largely unchanged. Here, we briefly describe its architecture with considerations for recent widely-used adjustments that improve their performance or computational requirements. For the remainder of this paper, we refer

[1]Hence, when we use the term "LLM" in this thesis, the implication is that it is a decoder transformer.

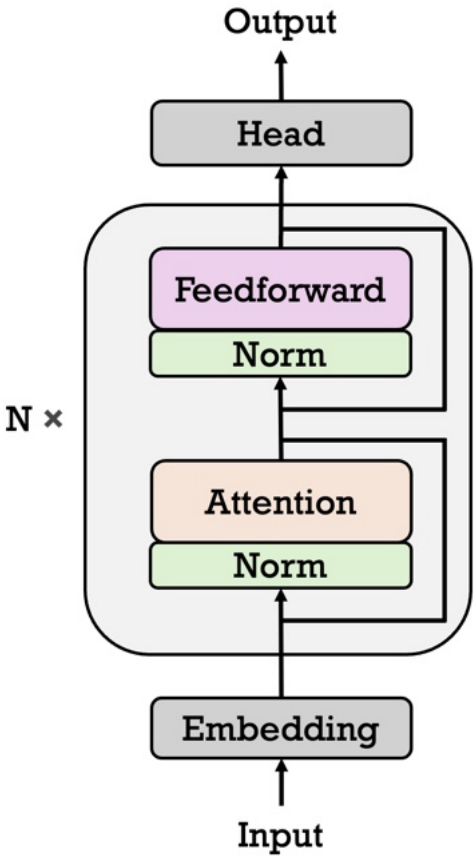

Figure 2.1: Transformer architecture. This abstraction applies to encoders and decoders.

to decoder-only transformers as simply transformers, unless stated otherwise (Chapter 7). Descriptions of encoder transformers can be found in Section 2.1.4.

In general, transformers, composed of a series of transformer blocks sandwiched between an embedding and linear head, process sequential inputs to produce sequential outputs. Each transformer block is made up of attention and feedforward (FF) blocks. For all linear operations described in the follow sections, they can be accompanied with biases using proper broadcasting, but we omit them for simplicity.

Attention Blocks

Making up half of a transformer block, the attention block (Bahdanau et al., 2015; Cheng et al., 2016) mixes information along the sequence axis. Let S be the input sequence length, and D, D_{qk}, and D_{vo} be various feature sizes. For an input $\boldsymbol{X} \in \mathbb{R}^{S \times D}$, we obtain queries

$\boldsymbol{Q}_h \in \mathbb{R}^{S \times D_{\text{qk}}}$, keys $\boldsymbol{K}_h \in \mathbb{R}^{S \times D_{\text{qk}}}$, and values $\boldsymbol{V}_h \in \mathbb{R}^{S \times D_{\text{vo}}}$ for each head $h = 1, 2, \ldots, H$ by

$$\boldsymbol{Q}_h = p_{\text{query}}(\boldsymbol{X}\boldsymbol{W}_{\boldsymbol{Q},h}), \tag{2.1}$$

$$\boldsymbol{K}_h = p_{\text{key}}(\boldsymbol{X}\boldsymbol{W}_{\boldsymbol{K},h}), \tag{2.2}$$

$$\boldsymbol{V}_h = \boldsymbol{X}\boldsymbol{W}_{\boldsymbol{V},h}, \tag{2.3}$$

for $\boldsymbol{W}_{\boldsymbol{Q},h} \in \mathbb{R}^{D \times D_{\text{qk}}}$, $\boldsymbol{W}_{\boldsymbol{K},h} \in \mathbb{R}^{D \times D_{\text{qk}}}$, and $\boldsymbol{W}_{\boldsymbol{Q},h} \in \mathbb{R}^{D \times D_{\text{vo}}}$. Positional encoding functions, p_{query} and p_{key}, add positional information to the otherwise permutation invariant calculations of attention. Some weights for keys and values may be shared across heads, known as multi-query (Shazeer, 2019) or grouped-query (Ainslie et al., 2023) attention, depending on the degree of sharing. Using a decoder mask $\boldsymbol{M} \in \mathbb{R}^{S \times S}$ ($[\boldsymbol{M}]_{i,j} = -\infty$ for $j > i$ and 0 elsewhere), per-head attention is calculated as

$$\boldsymbol{A}_h = \text{softmax}\left(\frac{\boldsymbol{Q}_h \boldsymbol{K}_h^\top}{\sqrt{D_{\text{qk}}}} + \boldsymbol{M}\right)\boldsymbol{V}_h. \tag{2.4}$$

Finally, a final linear transformation $\boldsymbol{W}_{\boldsymbol{O}} \in \mathbb{R}^{HD_{\text{vo}} \times D}$ on the concatenated outputs per head gives us the output of the attention block:

$$\text{Attn}(\boldsymbol{X}) = \begin{bmatrix}\boldsymbol{A}_1 & \boldsymbol{A}_2 & \cdots & \boldsymbol{A}_H\end{bmatrix}\boldsymbol{W}_{\boldsymbol{O}}. \tag{2.5}$$

Feedforward Blocks

The feedforward (FF) block makes up the second half of a transformer layer and mixes information along the feature axis. For an input $\boldsymbol{X} \in \mathbb{R}^{S \times D}$, FF blocks can be simply formulated as

$$\text{FF}(\boldsymbol{X}) = \text{FF}_2(\underbrace{\text{FF}_1(\boldsymbol{X})}_{\boldsymbol{Z}}) \tag{2.6}$$

where FF_1 is nonlinear and $\text{FF}_2(\boldsymbol{Z}) = \boldsymbol{Z}\boldsymbol{W}_2$ for FF activations $\boldsymbol{Z} \in \mathbb{R}^{S \times D_{\text{FF}}}$. This describes a wide range of FF architectures and arbitrary activation functions σ. For instance, in older LLMs like GPT 2 (Radford et al., 2019) and OPT (Zhang et al., 2022),

$$\text{FF}_1(\boldsymbol{X}) = \sigma(\boldsymbol{X}\boldsymbol{W}_1). \tag{2.7}$$

More modern LLMs, such as in the Llama and Gemma families, tend to use gated linear units (GLUs) (Shazeer, 2020) in FF blocks:

$$\text{FF}_1(\boldsymbol{X}) = \sigma(\boldsymbol{X}\boldsymbol{W}_g) \odot (\boldsymbol{X}\boldsymbol{W}_1) \tag{2.8}$$

where $\odot$ signifies element-wise multiplication. For all examples, $\boldsymbol{W}_1, \boldsymbol{W}_g \in \mathbb{R}^{D \times D_{\text{FF}}}$ and $\boldsymbol{W}_2 \in \mathbb{R}^{D_{\text{FF}} \times D}$ where typically, $D_{\text{FF}} \gg D$.

Putting It Together

With the attention and FF blocks defined, we are ready to describe the transformer architecture that many LLMs use. For an input sequence $\boldsymbol{t} \in [V]^S$ of length S discrete tokens from a size V vocabulary, it is embedded into a high dimensional continuous space to form features $\boldsymbol{X} \in \mathbb{R}^{S \times D}$. From here, $\boldsymbol{X}$ enters a series of L transformer layers, each containing an attention block followed by a FF block. Let $\text{Norm}_{l,.}$, Attn_l, and FF_l be a normalization layer, attention block (2.5), and FF block (2.6), respectively, at transformer layer $l = 1, 2, \ldots, L$. Then, the transformer layer l computes $\boldsymbol{X}_{\text{FF},l}$:

$$\boldsymbol{X}_{\text{Attn},l} = \boldsymbol{X}_{\text{FF},l-1} + \text{Attn}_l(\text{Norm}_{l,\text{Attn}}(\boldsymbol{X}_{\text{FF},l-1})), \tag{2.9}$$

$$\boldsymbol{X}_{\text{FF},l} = \boldsymbol{X}_{\text{Attn},l} + \text{FF}_l(\text{Norm}_{l,\text{FF}}(\boldsymbol{X}_{\text{Attn},l})), \tag{2.10}$$

where $\boldsymbol{X}_{\text{FF},0} \coloneqq \boldsymbol{X}$. Finally, the output from the last transformer layer, $\boldsymbol{X}_{\text{FF},L}$, passes through a linear head: $\boldsymbol{Y} = \boldsymbol{X}_{\text{FF},L}\boldsymbol{W}_{\text{LH}}$, where $\boldsymbol{W}_{\text{LH}} \in \mathbb{R}^{D \times V}$ and $\boldsymbol{Y} \in \mathbb{R}^{S \times V}$. Each $[\boldsymbol{Y}]_t$ corresponding to the t-th token contains logits that can be used to find the token probabilities at index $t + 1$.

2.1.2 Inference

To generate S_{G} tokens from a length S_{P} input sequence, we start with passing the entire prompt through the transformer. The logits in $[\boldsymbol{Y}]_{S_{\text{P}}}$ determine the first generated token, which is appended to the prompt and passed together back into the transformer to generate the next token. This process is repeated until some stopping condition is met. This method of autoregressively generating tokens requires cubic scaling of computation with respect to the total sequence length.

Key-value Caching

Key-value (KV) caching alleviates a significant amount of computational demand for generation by skipping repeated computation in attention blocks, *reducing computation to be quadratic with respect to the total sequence length at the cost of linear scaling in memory.* Here, inference is split into two distinct phases: prompt (or prefill) and generation. The prompt phase propagates the entire sequence through the model as before but also saves every key and value in every layer, attention head, and token index. This is because attention keys and values of previous tokens are identical for every future token to use, and FF blocks are anyways independent of token index. In turn, every step in the generation phase involves passing only the most recently generated token into the transformer, instead of the whole concatenated sequence with the prompt, while constantly appending new KV pairs for every new token. While maintaining a KV cache losslessly accelerates generation, this places a massive burden on memory which can often be inflexible in practice, which we discuss further in Chapter 3.

Inference Scaling

The number of generation tokens can be made intentionally variable (Figure 2.2). Whether it be spending more tokens to outline a model's steps (sequential scaling) or sampling multiple rollouts (parallel scaling) before producing a final answer, inference scaling can produce higher quality outputs at the cost of additional computation, memory, and time. Nevertheless, these class of scaling methods show large benefits for tasks that require

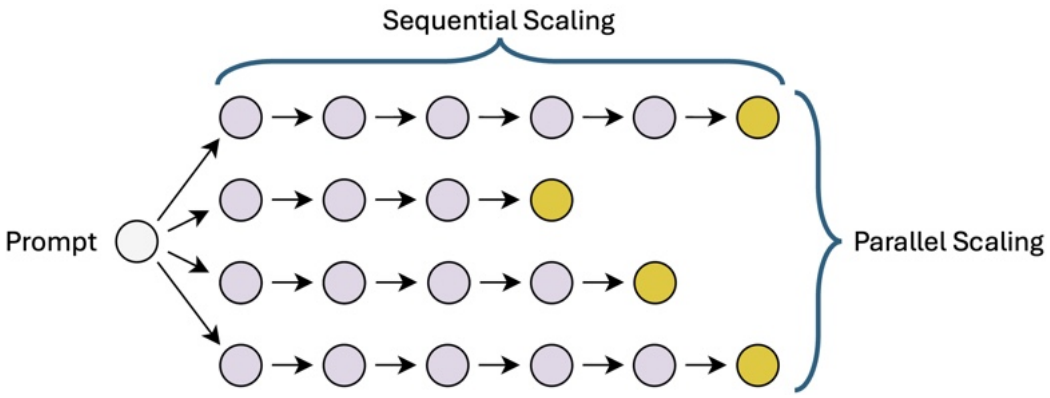

Figure 2.2: For a single prompt, LLM performance can improve by letting it generate longer CoTs (sequential scaling) and/or more responses (parallel scaling).

reasoning, such as in math, science, and logic. However, this only makes efficiency a more dire issue. This is the focus of Chapters 5, 6, and 7.

2.1.3 Training

While the focus of this thesis is on improving inference, we sometimes will need to briefly fine tune models to elicit desired inference behavior. Hence, we succinctly provide an overview on LLM training.

Training an LLM is split into multiple stages. A model is first pretrained on a large corpus of data (e.g., 15T tokens for Llama 3 8B (Dubey et al., 2024)). A pretrained model, also known as a base model, will already has strong general capabilities. From here, multiple post-training steps can further improve performance for downstream applications and consumer-facing products. One critical post-training step is instruction tuning (Wei et al., 2021) where base LLMs are taught to follow instructions, rather than just acting as text completion models. Another post-training step is alignment (Bai et al., 2022; Ouyang et al., 2022), where human preference data guide desirable characteristics of LLM outputs, such as safety, bias, and style. To improve technical reasoning skills, LLMs may undergo another stage of post-training on verifiable tasks like math and coding (Guo et al., 2025; Lambert et al., 2024).

Different training stages can use different learning paradigms. Pretraining on vast

amounts of data is self-supervised by defining the learning objective as next token prediction. Mathematically, this means for a tokenized sample $\boldsymbol{t} \in [V]^S$, the input is $[\boldsymbol{t}]_{1:S-1}$, and the token-wise target is the shifted sequence $[\boldsymbol{t}]_{2:S}$. The model learns to minimize the cross-entropy loss. Once pretrained, we can use other paradigms. Supervised fine tuning (SFT) takes prompt and target response pairs to perform the same cross-entropy loss for the response token indices. In some settings, it may be preferable to use reinforcement learning (RL), where the model can learn from information about a generated output's reward.

2.1.4 Adjustments for Encoder Transformers

Encoders like BERT (Devlin et al., 2019) and Vision Transformers (Dosovitskiy et al., 2021) differ slightly from decoder transformers. Architecturally, the difference is in the attention calculation. In contrast to (2.4), attention blocks in encoders allow all tokens to attend to each other, forgoing the need of a decoder mask:

$$\boldsymbol{A}_h = \text{softmax}\left(\frac{\boldsymbol{Q}_h \boldsymbol{K}_h^\top}{\sqrt{D_{\text{qk}}}}\right)\boldsymbol{V}_h. \tag{2.11}$$

Similar to decoders, encoders also suffer from computational bottlenecks associated with attention due to constructing matrices of shape $S \times S$. Unlike decoders, pure encoders do not need to store a KV cache.

The objective of encoder models is also different. Instead of next token prediction, they provide text embeddings which can be useful for denoising, retrieval-augmented generation, or data grouping, among many others. In turn, self-supervised pretraining usually involves recovering a sequence given incomplete or shuffled observations. More formally, for a sample $\boldsymbol{t}$ and corruption function c, a common training objective would be to recover $\boldsymbol{t}$ from $c(\boldsymbol{t})$. For vision, the training objectives can include classification, contrastive learning, and masked autoencoding.

Additionally, encoder inference is simpler compared to their decoder counterpart. Since encoders are not used for generation, only a single forward pass is needed (no autoregres-

sion). Encoders can also be used in conjunction with decoders to create encoder-decoder transformers (Raffel et al., 2020; Vaswani et al., 2017). Here, embeddings from the encoder are used in newly added cross-attention blocks in the decoder as keys and values.

2.2 Related Works

The remainder of this chapter will cover the intuition, benefits, and limitations of existing work on improving inference for transformers.

2.2.1 Attention Approximation Algorithms

Attention blocks in transformers are a bottleneck when processing long sequences due to the large size of the KV cache. Many methods try to alleviate this with sparse or low-rank approximations of attention features, each with their strengths and weaknesses.

KV Cache Policies

Many current methods to reduce the KV cache footprint involve keeping a tiny subset of the keys and values either with some pruning policy (Ge et al., 2023; Han et al., 2023; Liu et al., 2023b; Oren et al., 2024; Xiao et al., 2023; Zhang et al., 2023b) or a local attention mechanism (Child et al., 2019; Parmar et al., 2018). The former method can be applied directly to trained models whereas the latter typically cannot, so with limited compute, deploying a KV cache pruning policy is more practical. Such methods take advantage of the observation that many KV pairs are irrelevant for attention in some tasks and thus omitting them leads to negligible performance loss. For instance, H_2O (Zhang et al., 2023b), continuously accumulates attention probabilities at each generation step to identify a set of heavy-hitting KV pairs to be cached together with the most recent pairs. Not explicitly designed for KV cache compression, algorithms for infinite inference (Han et al., 2023; Xiao et al., 2023) maintain a full cache, but as the input sequence exceeds the maximum context length of a model, KV pairs in the middle of the sequence are dropped. Staying within the maximum context length, this results in a cache that maintains the

most recent and first few tokens. Regardless of the sparse method, maintaining a tight KV cache budget can seriously impair model performance, as we will see in Section 3.3.

There also exist several non-eviction based methods. DMC involves using a large amount of data to fine tune models to choose between accumulating or appending each KV pair (Nawrot et al., 2024). CacheGen's KV cache compression at the bit-level takes a query-agnostic approach (Liu et al., 2023a). In vision tasks, token merging is an effective way to cut down the number of tokens to process (Bolya et al., 2022; Renggli et al., 2022).

Low-rank Attention

Low-rank structures in attention have been explored extensively (Tay et al., 2022), namely from the lens of recurrent neural networks (RNNs).[2] Unlike transformers, RNNs integrate information from all previous tokens into *hidden states*, analogous low-rank structures to KV caches that organically occupy constant memory. In fact, this feature of RNNs over transformers has motivated research in alternative architectures (Dao et al., 2022b; Gu and Dao, 2023; Peng et al., 2023; Poli et al., 2023; Sun et al., 2023b), but for now, their adoption in LLMs is very limited compared to transformers. Though not as performative as these alternative architectures, linear transformers that break apart the attention operation into kernels also maintain a constant sized KV cache (Choromanski et al., 2020; Katharopoulos et al., 2020; Peng et al., 2021; Tsai et al., 2019) by reformulating the cache into an RNN hidden state. These types of caching mechanisms are *low-rank* since information is condensed along the sequence axis, rather than explicitly maintaining individual tokens. This is possible when we replace the softmax with a separable similarity metric $\phi(\boldsymbol{q}_t)\psi(\boldsymbol{K}_t)^\top$ for some row-wise functions ϕ and ψ, letting $\boldsymbol{q}_t \in \mathbb{R}^{1\times D}$ and $\boldsymbol{K}_t \in \mathbb{R}^{t\times D}$ be the query and keys at step t, respectively. To elaborate, if ϕ and ψ are such that

$$\boldsymbol{a}_t = \text{softmax}\left(\frac{\boldsymbol{q}_t \boldsymbol{K}_t^\top}{\sqrt{D}}\right)\boldsymbol{V}_t \approx \frac{\phi(\boldsymbol{q}_t)\psi(\boldsymbol{K}_t)^\top \boldsymbol{V}_t}{\phi(\boldsymbol{q}_t)\psi(\boldsymbol{K}_t)^\top \mathbf{1}_{S\times 1}}, \tag{2.12}$$

[2]The use of kernels on queries and keys to factorize the softmax operation in attention (Choromanski et al., 2020; Katharopoulos et al., 2020; Tsai et al., 2019) is often categorized as a low-rank attention method. We follow this custom as well.

we just need to cache hidden states $\boldsymbol{H}_t = \psi(\boldsymbol{K}_t)^\top \boldsymbol{V}_t \in \mathbb{R}^{R\times D}$ and $\boldsymbol{z}_t = \sum_{s=1}^{t} \psi([\boldsymbol{K}_t]_s) \in \mathbb{R}^{1\times R}$ for inference which can be expressed recursively as

$$\boldsymbol{H}_{t+1} = \boldsymbol{H}_t + \psi(\boldsymbol{k}_t)^\top \boldsymbol{v}_t, \tag{2.13}$$

$$\boldsymbol{z}_{t+1} = \boldsymbol{z}_t + \psi(\boldsymbol{k}_t) \tag{2.14}$$

for each new KV pair $(\boldsymbol{k}_t, \boldsymbol{v}_t)$. At initialization, $\boldsymbol{H}_0 = \boldsymbol{0}_{R\times D}$ and $\boldsymbol{z}_0 = \boldsymbol{0}_{1\times R}$. This is a clear improvement from having to store ever increasing sizes of $\boldsymbol{K}_t$ and $\boldsymbol{V}_t$, as the memory consumption is independent from t. Note that our presentation differs slightly since we do not constrain $\phi = \psi$ (Chen et al., 2023c). With this formulation, transformers act like RNNs which occupy constant memory during generation by not appending but updating hidden states during each generation step. Since LLMs are not typically trained in this fashion, a major challenge is to induce this property without significant computation or adjustment to the original weights (Kasai et al., 2021). While its dilution of information restricts its performance when specific tokens need to be recalled with strong signals (Khandelwal et al., 2018), this is exactly what a sparse KV cache algorithm can do, so we can fully take advantage of its infinite compression capability to obtain some high level representation of the less important tokens, meaning kernelized attention is a good candidate method for `LESS` to learn the residual (Chapter 3).

Sparse and Low-rank Decomposition

The study of robust principal component analysis (Candès et al., 2011; Chandrasekaran et al., 2011; Dong et al., 2023d; Tong et al., 2021) has shown this type of decomposition greatly enhances approximation accuracy and expressibility beyond just sparse or low-rank matrices alone. Its success has spread to deep learning areas such as efficient attention (Chen et al., 2021; Sun et al., 2025), model compression (Li et al., 2023), and fine-tuning (Nikdan et al., 2024).

2.2.2 Feedforward Algorithms

The sheer size of LLM FF blocks makes them an additional source of inefficiency. Thankfully, FF blocks tend to learn structures that can be exploited.

Feedforward Activation Sparsity

Various forms of sparsity in trained transformer FF blocks have been observed in the past (Dettmers et al., 2022; Dong et al., 2023a, 2024b; Geva et al., 2020; Li et al., 2022; Liu et al., 2023c). In ReLU-based LLMs like OPT (Zhang et al., 2022), the activations can be exceptionally sparse and become more apparent for larger models (Liu et al., 2023c). As more models use non-sparse activation functions like GLU variants (Shazeer, 2020), it is difficult for neurons to have no contribution to the output since these functions do not have an interval that maps to zero. Without exact sparsity, the efficacy of efficient methods that rely on exact sparsity becomes limited. As such, this has ushered a wave of models that are either adapted from available models (Jiang et al., 2024; Liu et al., 2021b; Mirzadeh et al., 2023; Zhang et al., 2021; Zheng et al., 2024) or trained from scratch (Fedus et al., 2022) which can produce activations that are exactly zero with little to no performance loss. Even so, these methods require considerable amounts of computational resources to develop.

Pruning

Pruning (LeCun et al., 1989) is one sparsity-guided way to tackle compute and memory bottlenecks of models, not just present in FF blocks. Previously, the common method would be some variation of iteratively rounding weights down to zero based on some score and retraining to recover any lost performance (Blalock et al., 2020; Frankle and Carbin, 2018; Liang et al., 2021; Liu et al., 2024). While this can result in most parameters being pruned, this method comes with a few issues. First, with the increasing scale of LLMs, retraining becomes impractical for most. Fortunately, cheap methods to effectively prune LLMs have been developed (Dery et al., 2024; Frantar and Alistarh, 2023; Jaiswal et al., 2023; Sun et al., 2023a). The second issue is that unless pruning is done in a structured

manner (Li et al., 2023; Ma et al., 2023; Santacroce et al., 2023; Xia et al., 2023, 2022), it is difficult to see real computational savings, yet structured pruning often leads to much more severe performance degradation. Third, pruning usually enforces sparsity to be static at inference which can be strong assumption since FF blocks are widely believed to contain the model's memory which accessed differently per input (Geva et al., 2020).

Mixture of Experts

Making sparsity more dynamic has motivated the design of mixture-of-experts (MoEs) (Jacobs et al., 1991) to avoid computing low-impact features in trained models at varying granularities (Csordás et al., 2023; Dai et al., 2024; Liu et al., 2023c; Piórczyński et al., 2023; Sadhukhan et al., 2026; Yerram et al., 2024; Zhang et al., 2021; Zheng et al., 2024). The main idea of MoEs is to use a gating function to dynamically identify a subset of parameters that will be used to compute the layer's output, reducing the number of active parameters at inference. In the ideal case, all active neurons are selected and inactive neurons are ignored for each input. However, current methods either require excessive training or rely on ReLU or ReLU-like activation functions.

2.2.3 Other Acceleration Algorithms

Beyond the methods described in previous sections, there are still many other classes of methods that improve efficiency of trained neural networks which are not within the scope of this thesis, but we include a quick overview of some to paint a more holistic picture of deep learning model efficiency and show insights relevant to our work. Many of these methods do not conflict with each other, so composition of multiple methods (including the ones already described) is possible and common in practice.

Other compression algorithms include knowledge distillation and quantization. For knowledge distillation (Hinton et al., 2015), a small model (student) learns from a powerful and larger model (teacher). As a result, the student model learns to mimic the teacher model's output distribution instead of learning solely from the data labels. In some cases, this can even be used to overcome the lack of data (Guo et al., 2025). Quantization com-

presses the weights or the activations by casting them into lower precision. While casting to half precision (BF16 of FP16) is common without noticeable performance degradation on simpler tasks, low-bit compression requires special consideration for sparse outlier values (Dettmers et al., 2022; Lin et al., 2024).

There are also algorithms that are theoretically lossless[3] such as speculative decoding, parallelization, and systems optimizations. Speculative decoding (Leviathan et al., 2023) uses a smaller model to generate multiple draft tokens and a larger model to verify the generations. Hence, final tokens are generated with lower latency, as long as the smaller model does not make frequent mistakes. Models and data can also be parallelized at inference, distributing operations across devices to accelerate inference or fit models that exceed the space of one device (Dong et al., 2024b; Narayanan et al., 2021). However, this will introduce a recurring communication cost. There are also improvements at the systems level to improve efficiency such as with fused kernels. A popular example is FlashAttention (Dao et al., 2022a), where a change in the order of operations in attention blocks can minimize memory read and write costs.

2.2.4 Inference Scaling & Reasoning

The growing capabilities of LLMs provide great promise for complex reasoning tasks such as those that involve significant amounts of math, science, and logic. Currently, instilling these capabilities into LLMs requires more than scaling the model (Hoffmann et al., 2022; Kaplan et al., 2020). In particular, inference scaling has shown immense success to elicit reasoning behavior from a pretrained model. Described below, inference scaling methods usually fall into two main classes: sequential and parallel. Whether it be sequentially or in parallel, many more tokens are generated in comparison to typical inference.

[3]Reality can be complicated due to unpredictable behavior from numerical instabilities, engine differences, and networking that may introduce discrepancies in edge cases (He and Lab, 2025; Yao et al., 2025).

Sequential Scaling

Due to the success of chain-of-thought (CoT) (Wei et al., 2022) which spells out each step to the answer rather than jumping to the answer, many models are trained specifically to generate long CoTs to mimic thinking (Guo et al., 2025; Muennighoff et al., 2025; Rastogi et al., 2025; Team et al., 2025; Yang et al., 2025a). Dubbed thinking or reasoning models, these LLMs extend the generation length up to tens of thousands of tokens to allow for backtracking, reflection, exploration, and evaluation within its reasoning trace.

Parallel Scaling

For one input, an LLM can generate multiple threads in parallel to generate one or more complete responses. To produce one response (N-to-1), the task can be broken into multiple subtasks that are tackled in parallel before merging or integrating them into the final response (Hsu et al., 2025; Jin et al., 2025; Rodionov et al., 2025; Yang et al., 2025b). Alternatively, to produce multiple complete responses (N-to-N), one can simply sample multiple responses from the LLM with non-greedy decoding. This way, the probability of producing a high-quality answer increases (Brown et al., 2024; Snell et al., 2024), and a post-generation synthesis method like self-consistency (Wang et al., 2022) or merging (Chen et al., 2023b; Qi et al., 2025; Zhao et al., 2025).

If we look outside of LLMs, this is not a new concept. Parallel inference scaling can be seen as an application of Bayesian sampling methods where direct Monte Carlo sampling independent generations is the simplest form of parallel scaling. Other forms of parallelism also exist in classical machine learning. Ensemble methods like random forests and bagging train multiple sub-models such that during inference, each sub-model produces an output which is merged with others (e.g., averaging or voting) to produce the final output which is often more reliable (Breiman, 1996, 2001). With training-intensive neural network architectures like LLMs and diffusion models, training multiple models is computationally and economically impractical. However, since modern generative models are stochastic during inference, multiple outputs can be sampled from the same model and same input, leading to the current jargon of "parallel inference scaling" which this thesis follows. Analogously,

refinements such as importance sampling (Faria and Smith, 2025; Yao et al., 2025) and rejection sampling (Chen et al., 2026; Leviathan et al., 2023; Verine et al., 2024) also exist for these generative models. Later in this thesis (Section 7.4), we show this technique has merit for diffusion models too.

2.3 Chapter Summary

Transformers are the workhorse of LLMs and in effect, advancements in AI. However, they can be clunky with various inefficiencies in various settings. Many works try to ease these bottlenecks to broaden deployment, but there is still much work to be done. Additionally with tasks that see enormous benefits from inference scaling, more compute appears necessary, but it is important to ask if we are using the added compute in the best way possible. Hence, two complementary directions emerge: **1)** identifying sources of resource overutilization to improve efficiency and **2)** identifying sources of resource underutilization to improve performance. These are deeply entangled together as insights from one can inform the other. Continuing further along this thesis, we will show our progress in both directions.

Chapter 3

Sparse and Low-rank KV Caching

We begin the main part of this thesis with KV cache management, an inherent issue with decoder transformer models. With poor scalability, KV caching requires pragmatic solutions to real computational limits. This chapter is based on Dong et al. (2024c).

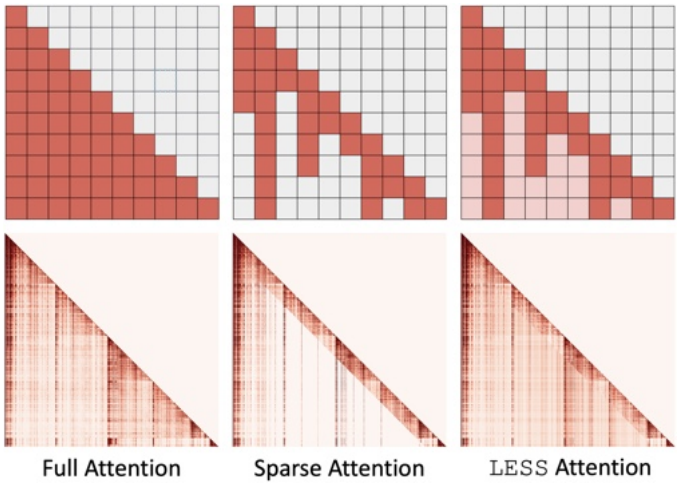

Figure 3.1: Toy (top row) and Llama 2 7B (bottom row) example decoder attention maps with H_2O as the underlying sparse policy. In the top row, red/pink and gray squares are positive and zero attention probabilities, respectively. In the bottom row, darker colors indicate larger attention probabilities. Sparse attention policies zero out many positive attention probabilities. Our method, `LESS`, ensures all previous tokens will have some contribution to the attention layer output to better retain information.

Article: ...ban on fracking in Wales...support for oil and gas applications is contrary to the approach of the Welsh Government of promoting renewable and low carbon forms of energy...

(Incorrect) Sparse Policy Summary: The Welsh government has announced measures to speed up the development of fracking.

Figure 3.2: Incorrect summary by Falcon 7B with sparse policy H_2O.

3.1 KV Cache Scaling in Practice

KV caching, an inference-time mechanism to accelerate LLM generation, comes with an immense memory burden by storing a growing number KV pairs for each previous token, layer, attention head, and intra-batch sample. For example, the Llama 2 7B model's KV cache for an input of batch size 64 and sequence length 1024 is 2.5× the model size. With memory being expensive to scale, LLM use in settings like chatbots, long document understanding, and biological sequence processing are limited.

A line of work discussed in Section 2.2.1, which we refer to as *sparse KV policies*, explores the selection of the best subset of KV pairs to cache (Han et al., 2023; Liu et al., 2023b; Xiao et al., 2023; Zhang et al., 2023b). Although very promising, these methods are inevitably and irrecoverably discarding KV pairs deemed, in one way or another, less important than others, leading to gaps in attention maps as shown in Figure 3.1. Consequently, they are boldly assuming tokens that are unimportant now will not hold significance at future decoding steps, a faulty conjecture for tasks that deviate from this pattern. For instance, using the popular sparse policy H_2O (Zhang et al., 2023b) on Falcon 7B (Almazrouei et al., 2023) to summarize an article (BBC, 2015; Narayan et al., 2018) produces a *factually incorrect* summary in Figure 3.2. For the full article, see Figure 3.13.

One way to combat information loss is to cache more tokens, but this is far from memory efficient. An ideal KV cache policy should **1)** minimize performance degradation from a full cache, **2)** scale at a much slower rate than the full KV cache, and **3)** be cheap to integrate into existing pretrained LLMs.

Fortunately, with some investigation into the *residual* between full and sparse attention

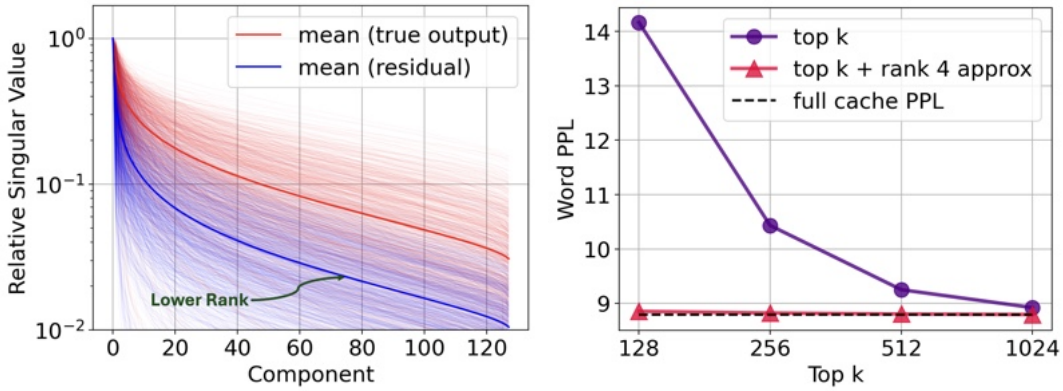

Figure 3.3: Attention residuals exploration in Llama 2 7B on WikiText (Merity et al., 2016). Mean and 1000 sample relative singular value plots of true attention outputs and residuals from top-512 sparse policy, showing the residual is much lower rank (left). End-to-end performance (lower is better) using top-k caching with and without low-rank approximations (right). A rank-4 approximation virtually recovers the original performance.

outputs, a better strategy emerges. First, define the residual as $\boldsymbol{\Delta}_{\boldsymbol{A}} = \boldsymbol{A} - \boldsymbol{A}_{\text{sparse}}$, where $\boldsymbol{A}$ and $\boldsymbol{A}_{\text{sparse}}$ are the full and sparse attention outputs, respectively. Using top-k selection as our sparse policy, we observe the residuals $\boldsymbol{\Delta}_{\boldsymbol{A}}$ are in fact *low-rank* — more so than $\boldsymbol{A}$ — based on Figure 3.3, a similar observation to Chen et al. (2021). Even a very low-rank approximation can nearly negate the performance degradation from sparse caching. *In turn, this finding motivates the use of low-rank methods to approximate the residuals for efficient caches.*

We propose LESS (**L**ow-rank **E**mbedding **S**idekick with **S**parse policy) to learn the *residual* between the original attention output and the attention output approximated by a sparse policy. LESS does this by accumulating information that would have been discarded by sparse policies into a constant-sized low-rank cache or state, allowing for queries to still access information to recover previously omitted regions in attention maps (see Figure 3.1).

We show that LESS makes significant progress towards an ideal cache:

1. **Performance Improvement:** LESS synthesizes sparse KV policies with low-rank states to bridge the performance gap on a variety of tasks where these sparse algorithms show cracks of weakness. In fact, LESS improves the performance much more than simply dedicating that memory to storing more KV pairs.

2. **Constant Low-rank Cache Size:** Low-rank caches in LESS occupy constant memory with respect to the sequence length, and in our experiments, the extra storage to accommodate LESS is *nearly free*, taking up the equivalent space of only 4 extra KV pairs in our experiments. Inspired by recurrent networks, the low-rank state stores new information by recursive updates rather than concatenation. As each sample has its own cache, LESS provides the same proportional cache reduction for small and large batch sizes.
3. **Cheap Integration:** Changes to the LLMs' architectures are small and do not perturb the original weights. The only modifications to LLMs will be the addition of tiny multilayer perceptions (MLPs) at each attention layer. For example, using LESS with Llama 2 13B adds fewer than 2% of the total number of parameters. In addition, we can train LESS at each attention layer independently from all others, bypassing expensive end-to-end training. Trained once, LESS can transfer to more relaxed settings while maintaining comparable performance, further extending its applicability.

Our comprehensive experiments on Llama 2 (Touvron et al., 2023) and Falcon (Almazrouei et al., 2023) with different sparse policies (Han et al., 2023; Xiao et al., 2023; Zhang et al., 2023b) on a variety of tasks demonstrates LESS as a highly performative method that reduces KV cache memory. For instance, LESS recovers more than 40% of the Rouge-1 degradation caused by a sparse policy on the CNN/DailyMail dataset (Hermann et al., 2015; See et al., 2017) with Falcon 7B. Finally, we provide an implementation of LESS that reduces the latency by up to 1.3× and increases the throughput by 1.7× from the full cache.

Next up, we introduce our hybrid KV caching algorithm in Section 3.2. Then, we showcase the performance of LESS in Section 3.3. Lastly, we explore other properties of our method in Section 3.4.

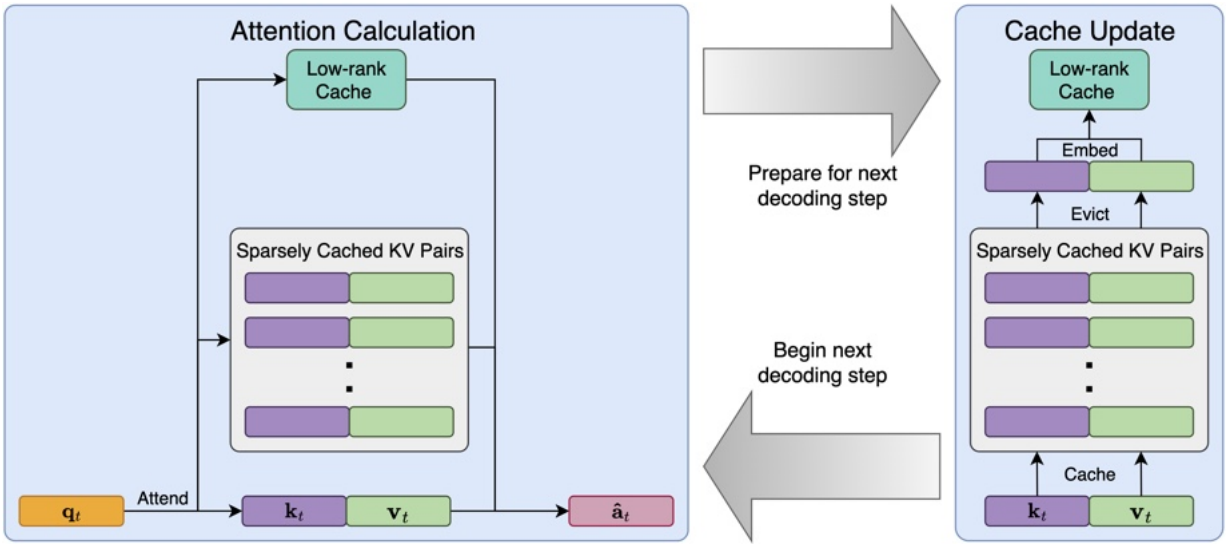

Figure 3.4: LESS algorithm during inference. At each decoding step, attention is calculated as in (3.3). To prepare for the next decoding step, the cache is updated by placing the most recent KV pair into the sparse policy cache, and if it has exceeded capacity, a KV pair will be evicted and integrated into the low-rank cache $\boldsymbol{H}_t$ before being deleted.

3.2 LESS: Sparse Eviction and Recurrence Synthesis

We propose LESS, a general method to synthesize low-rank caches with *any* eviction-based sparse KV cache policy, $\mathfrak{C}$, to close the performance gap from full KV caching while being efficient. Notably, our method only adds a constant sized cache which does not scale with the sequence length. For the sparse policy, $\mathfrak{C}$, we require that it can output the cached keys $\boldsymbol{K}_{\mathfrak{C},t} \in \mathbb{R}^{B_t \times D}$, the cached values $\boldsymbol{V}_{\mathfrak{C},t} \in \mathbb{R}^{B_t \times D}$, and the set of discarded KV pairs $\mathcal{D}_t$ at iteration t where $B_t \in \mathbb{N}$ is the number of cached pairs.

Letting $\cdot$ denote both ϕ and ψ, we define our kernels as

$$\phi(\boldsymbol{q}) = |\sigma_\phi(\sigma_\phi(\boldsymbol{q}\boldsymbol{W}_{\phi,1})\boldsymbol{W}_{\phi,2})| \tag{3.1}$$

$$\psi(\boldsymbol{k}) = |\sigma_\psi(\sigma_\psi(\boldsymbol{k}\boldsymbol{W}_{\psi,1})\boldsymbol{W}_{\psi,2})\boldsymbol{W}_{\psi,3}| \tag{3.2}$$

for activation functions $\sigma_\cdot$, $\boldsymbol{W}_{\cdot,1} \in \mathbb{R}^{D \times R'}$, $\boldsymbol{W}_{\cdot,2} \in \mathbb{R}^{R' \times R}$, and $\boldsymbol{W}_{\psi,3} \in \mathbb{R}^{R \times R}$. The element-wise absolute values ensure the inner product $\phi(\boldsymbol{q})\psi(\boldsymbol{k})^\top > 0$ to preserve the nonnegativity of attention probabilities. In the ideal case, if $\phi(\boldsymbol{q})\psi(\boldsymbol{k})^\top = e^{\boldsymbol{q}\boldsymbol{k}^\top/\sqrt{D}}$ for all $\boldsymbol{q}, \boldsymbol{k}$, then the

result would be the original attention probabilities.

Now, we outline our method, summarized in Algorithm 1 and illustrated in Figure 3.4, which split into two parts: attention calculation and cache updates.

Attention Calculation

At step t, we find the query-key-value triplet $(\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t)$ from the input token as usual. Recalling that we have cached $\boldsymbol{K}_{\mathfrak{C},t}$, $\boldsymbol{V}_{\mathfrak{C},t}$, $\boldsymbol{H}_t$, and $\boldsymbol{z}_t$ from the previous generation step, append $\boldsymbol{k}_t$ to $\boldsymbol{K}_{\mathfrak{C},t}$ and $\boldsymbol{v}_t$ to $\boldsymbol{V}_{\mathfrak{C},t}$ to obtain $\boldsymbol{K}'_{\mathfrak{C},t} \in \mathbb{R}^{(B_t+1)\times D}$ and $\boldsymbol{V}'_{\mathfrak{C},t} \in \mathbb{R}^{(B_t+1)\times D}$, respectively. Then, we can find $\widehat{\boldsymbol{a}}_t$, our approximation of the original attention $\boldsymbol{a}_t$, by computing

$$\widehat{\boldsymbol{a}}_t = \frac{\phi(\boldsymbol{q}_t)\boldsymbol{H}_t + \exp(\boldsymbol{q}_t(\boldsymbol{K}'_{\mathfrak{C},t})^\top/\sqrt{D})\boldsymbol{V}'_{\mathfrak{C},t}}{\phi(\boldsymbol{q}_t)\boldsymbol{z}_t^\top + \exp(\boldsymbol{q}_t(\boldsymbol{K}'_{\mathfrak{C},t})^\top/\sqrt{D})\mathbf{1}_{B\times 1}}. \tag{3.3}$$

During the prompt phase (i.e. $t = 0$), it is just regular attention since $\boldsymbol{H}_0 = \mathbf{0}_{R\times D}$ and $\boldsymbol{z}_0 = \mathbf{0}_{1\times R}$.

Cache Updates

With the attention computed, we need to prepare the necessary ingredients for iteration $t+1$ by finding $\boldsymbol{K}_{\mathfrak{C},t+1}$, $\boldsymbol{V}_{\mathfrak{C},t+1}$, $\boldsymbol{H}_{t+1}$, and $\boldsymbol{z}_{t+1}$. The first two are simple since the sparse policy will return $\boldsymbol{K}_{\mathfrak{C},t+1}$, $\boldsymbol{V}_{\mathfrak{C},t+1}$, and $\mathcal{D}_{t+1}$. Before freeing $\mathcal{D}_{t+1}$ from memory, we embed its information into $\boldsymbol{H}_{t+1}$ and $\boldsymbol{z}_{t+1}$:

$$\boldsymbol{H}_{t+1} = \boldsymbol{H}_t + \sum_{(\boldsymbol{k},\boldsymbol{v})\in\mathcal{D}_{t+1}} \psi(\boldsymbol{k})^\top \boldsymbol{v}, \tag{3.4}$$

$$\boldsymbol{z}_{t+1} = \boldsymbol{z}_t + \sum_{(\boldsymbol{k},\boldsymbol{v})\in\mathcal{D}_{t+1}} \psi(\boldsymbol{k}). \tag{3.5}$$

After this, $\mathcal{D}_{t+1}$ can be deleted, and we are prepared for the following generation step. Intuitively, $\boldsymbol{H}_{t+1}$ and $\boldsymbol{z}_{t+1}$ are updated recursively by keys and values that have been newly pruned at each decoding step. As such, they are constant size repositories of information

from all deleted KV pairs which becomes clear when we expand the recursion:

$$\boldsymbol{H}_{t+1} = \sum_{(\boldsymbol{k},\boldsymbol{v})\in\bigcup_{i=1}^{t+1}\mathcal{D}_i} \psi(\boldsymbol{k})^\top \boldsymbol{v}, \tag{3.6}$$

and similarly for $\boldsymbol{z}_{t+1}$.

Algorithm 1: Generation Step with LESS

Input: $\mathfrak{C}, \boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t$
Load $\boldsymbol{K}_{\mathfrak{C},t}, \boldsymbol{V}_{\mathfrak{C},t}, \boldsymbol{H}_t, \boldsymbol{z}_t$ from memory.
$\boldsymbol{K}'_{\mathfrak{C},t} \leftarrow \text{concatenate}(\boldsymbol{K}_{\mathfrak{C},t}, \boldsymbol{k}_t)$
$\boldsymbol{V}'_{\mathfrak{C},t} \leftarrow \text{concatenate}(\boldsymbol{V}_{\mathfrak{C},t}, \boldsymbol{v}_t)$
Obtain $\widehat{\boldsymbol{a}}_t$ via (3.3).
Obtain $\boldsymbol{K}_{\mathfrak{C},t+1}, \boldsymbol{V}_{\mathfrak{C},t+1}, \mathcal{D}_{t+1}$ from sparse KV cache algorithm $\mathfrak{C}$.
Update $\boldsymbol{H}_{t+1}$ via (3.4).
Update $\boldsymbol{z}_{t+1}$ via (3.5).
Save $\boldsymbol{K}_{\mathfrak{C},t+1}, \boldsymbol{V}_{\mathfrak{C},t+1}, \boldsymbol{H}_{t+1}, \boldsymbol{z}_{t+1}$.
Delete $\mathcal{D}_{t+1}$.
Return: $\widehat{\boldsymbol{a}}_t$

3.2.1 Implementation Details

Inexpensive Training

With our inference-time protocol outlined, we now describe how we can cheaply train our kernel functions ϕ and ψ. Because training end-to-end is time consuming and resource intensive, we choose to train ϕ and ψ at each layer independent of all other layers which already surprisingly gives strong results. The training objective is to minimize the ℓ_2 distance to the output projection of the original attention layer using that layer's inputs. All weights except for those in ϕ and ψ are frozen. As a result, the only computational requirements are the abilities to backpropagate through a single attention layer and run inference on the full model to collect a dataset of attention layer inputs and outputs, which for all models we experiment with, *can be done on a single GPU*. With more devices, training each layer can be parallelized. While inference follows recursive updates of $\boldsymbol{H}_t$ and $\boldsymbol{z}_t$, this does not impede parallelism along the sequence axis because we can just

construct the full attention matrix where entries not computed by sparsely cached KV pairs, as determined by whichever sparse policy we train on, will be found by the kernel functions.

LESS was trained using Adam (Kingma and Ba, 2014) for 40 epochs with an initial learning rate of 0.001 which halved every 10 epochs. We fixed the hidden layer dimension $R' = 512$, used a dropout rate of 0.3 within the kernels, and let all nonlinear functions σ_ϕ and σ_ψ to be GELUs. None of the original model's weights are updated. First, we sampled 512 sequences for Llama 2 models (Touvron et al., 2023) and 1024 sequences for Falcon (Almazrouei et al., 2023) from the C4 training set (Raffel et al., 2019). Since Falcon's context length is half of Llama 2's, the training sets have the same number of tokens. Next, queries, keys, and values at each layer would be collected as each sample propagated through the models. These collected features (fed in batches of 2) would be used to train the kernels at each layer independently using some sparse policy at some sparsity level. For multi-query attention (Shazeer, 2019), we extend H_2O to aggregate attention scores across all query attention heads to determine KV pairs to evict.

We find that the kernel initialization is critical. As we will show in our experiments (Section 3.3), the sparse policies already have decent performance which we want to use as a starting point. We add learnable scalars between layers in ψ initialized to 10^{-4}, so the influence of LESS during the first few gradient steps is small. This way, the sparse policy acts as a warm start, and we can immediately reduce the sparse policy's residual.

Efficient Generation

We also develop an implementation that enhances throughput and reduces the latency of LLM generation of LESS. For the sparse cache, we adapt the implementation from Zhang et al. (2023b) to support any KV cache eviction algorithm efficiently. To avoid data movement in memory, we directly replace the evicted KV pair with the newly-added one. As our kernels are small MLPs with GELUs, we implement a fused linear kernel that absorbs the activation into the layer before to avoid writing the intermediate results to DRAM for the low-rank cache.

Table 3.1: Token counts at different sparsity levels.

Model	Max Length	# Tokens at 2%/5%/10%
Llama 2	4096	80 / 204 / 408
Falcon	2048	40 / 102 / 204

3.3 Core Experiments

Here, we demonstrate the impressive performance of LESS across multiple datasets, models (Llama 2 and Falcon), sparse policies (Han et al., 2023; Oren et al., 2024; Xiao et al., 2023; Zhang et al., 2023b), and sparsity levels, despite allocating only approximately 4 tokens of storage to the low-rank state. In Section 3.3.1, LESS achieves the closest performance to the full cache in language modeling and classification tasks. For example, evaluated with 2% H_2O in Llama 2 7B, LESS reduces the word perplexities on WikiText and PG-19 by over 20% from H_2O alone, relative to the full cache performance. Section 3.3.2 shows similar gains in summarization. For example, LESS reduces Rouge-1 degredation by 10% H_2O in Falcon 7B on CNN/DailyMail by 41.4%. In Section 3.3.3, we note the lower latency ($1.1 - 1.3\times$ reduction) and higher throughput of LESS ($1.7\times$ higher) compared to full caching.

We explore three sparse policies: H_2O (Zhang et al., 2023b), Λ-masking from the infinite generation literature (Han et al., 2023; Xiao et al., 2023), and TOVA (Oren et al., 2024). When using H_2O, the sparse KV cache is equally split between the heavy hitter tokens and the recent tokens (e.g. 5% H_2O cache consists of 2.5% heavy hitters and 2.5% recent tokens). For Λ-masking, the cache consists of the first 4 and most recent tokens. The percentages represent how much of the model's max context length is cached, so regardless of input length, the cache size remains the same for fairness. Since the sparsity level can translate to different numbers of tokens among models based on the max input lengths, we include Table 3.1 as a quick reference for the models we evaluate on, Llama 2 and Falcon. The token count is rounded down to the nearest even number to make sure H_2O can have an even split.

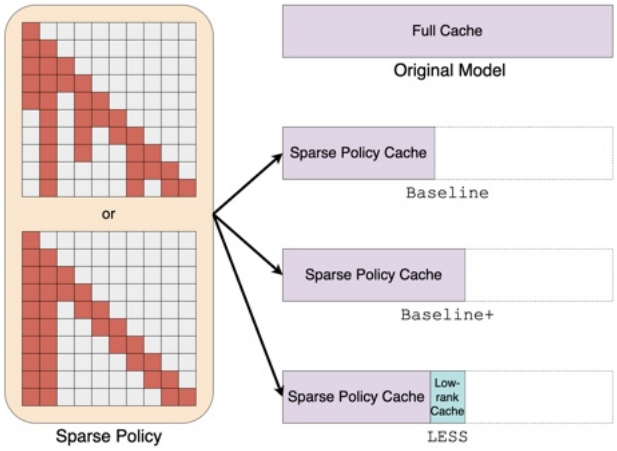

Figure 3.5: `LESS` experimental setup. First, a sparse policy is chosen as the underlying policy behind all methods. Then, we compare performance among the full cache model, `Baseline`, `Baseline+`, and `LESS`. `Baseline+` and `LESS` use the same amount of storage which is slightly larger than the requirements of `Baseline`.

For our experiments, we set the kernel size $R = 8$, unless otherwise stated. Thus, while minuscule, the size of $\boldsymbol{H}$ is nonzero, equivalent to caching 4 extra tokens. We ignore the influence of $\boldsymbol{z}$ since it only has R entries. As such, when evaluating on a task at $\alpha\%$ sparsity, we compare `LESS` with the sparse policy $\mathfrak{C}$ at $\alpha\%$ sparsity and at $\alpha\%$ sparsity plus additional tokens to match the extra space taken by $\boldsymbol{H}$ (e.g. 4 tokens in experiments where $R = 8$), which we denote as `Baseline` and `Baseline+`, respectively. Both are inherently sparse-only policies. A visual representation of the different baselines can be found in Figure 3.5. Note that the sparsity level and policy $\mathfrak{C}$ will vary throughout our experiments depending on the context. The purpose of evaluating `Baseline` is to compare the performance gain from extra tokens and the low-rank state $\boldsymbol{H}$. Additionally, we evaluate the full KV cache to observe how far we are from the unconstrained potential of the original model. For our method, we denote it as `LESS` ($\beta\%$) where β is the percent cache sparsity `LESS` was trained on with some sparse policy depending on the context.

Table 3.2: Llama 2 7B WikiText and PG-19 word perplexities with H_2O as the primary underlying sparse policy. Numeric column names indicate the sparsity levels during test time. Lower is better.

H_2O Method	2% H_2O	5% H_2O	10% H_2O
WikiText			
Full Cache	8.791	8.791	8.791
Baseline	13.333	9.863	9.295
Baseline+	12.718	9.842	9.288
H_2O+Performer	13.332	9.863	9.296
LESS (2%)	**10.745**	9.658	9.261
LESS (5%)	11.321	**9.657**	9.239
LESS (10%)	14.577	9.693	**9.230**
PG-19			
Full Cache	23.787	23.787	23.787
Baseline	37.013	27.939	25.451
Baseline+	35.832	27.829	25.429
H_2O+Performer	36.996	27.938	25.451
LESS (2%)	**32.157**	27.887	26.322
LESS (5%)	33.195	**27.089**	25.979
LESS (10%)	41.204	27.201	**25.134**

3.3.1 Language Modeling & Classification

We start with validating our method trained at different sparsity levels on some language modeling and classification tasks at different sparsity levels using Language Modeling Evaluation Harness (Gao et al., 2023). For these tasks, we use the same setup as in training by masking out query-key interactions depending on the sparse policy and having LESS capture the masked probabilities. In addition, we purposefully mismatch training and testing sparsity levels to uncover insight on the transferability between test sparsity levels. To illustrate why a learned kernel is necessary, we also evaluate H_2O with Performer kernels (Choromanski et al., 2020) based on random Fourier features (Rahimi and Recht, 2007), which we denote as H_2O+Performer.

Table 3.2 shows Llama 2 7B performance on WikiText (Merity et al., 2016) and PG-19 (Gao et al., 2020; Rae et al., 2019) using H_2O. Looking at the scenarios where training sparsity is equal to the test sparsity, our method is able to achieve much lower word

Table 3.3: Llama 2 7B performance on WikiText (word perplexity), MuTual (16-shot R@1), and BoolQ (10-shot accuracy) with 5% Λ-masking as the primary underlying sparse policy.

Λ Method	WikiText (↓)	MuTual	BoolQ
Full Cache	8.79	55.08	80.40
Baseline	10.66	53.50	77.28
Baseline+	10.64	53.27	77.46
LESS (5%)	**10.12**	**54.51**	**78.65**

Table 3.4: Llama 2 7B performance (word PPL) on WikiText and PG-19 at 5% token sparsity using different sparse policies. Lower is better.

Method	H_2O	Λ	TOVA
WikiText			
Full Cache	8.79	8.79	8.79
Baseline	9.86	10.66	9.97
Baseline+	9.84	10.64	9.95
LESS (5%)	**9.66**	**10.12**	**9.72**
PG-19			
Full Cache	23.79	23.79	23.79
Baseline	27.94	22K	27.88
Baseline+	27.83	22K	27.79
LESS (5%)	**27.09**	**3.9K**	**27.34**

perplexities than the baselines. Notably, LESS beats Baseline by a wider margin than Baseline+ and H_2O+Performer, indicating that LESS uses the space of 4 extra tokens most effectively. The lackluster performance of H_2O+Performer suggests that learned kernels are needed to make a noticeable improvement. Moreover, LESS trained at one sparsity level can often generalize reasonably to higher sparsity levels especially on WikiText, even sometimes matching the performance of ones trained at the test sparsity level. The reverse is less effective but can still be better than the baselines. However, all methods are still quite far from the full cache performance.

Evaluation results (Clark et al., 2019; Cui et al., 2020) with Λ-masking in Table 3.3 show LESS's benefit to a different sparse policy (though less performative than H_2O). Similar to the case with H_2O, LESS closes the gap from full caching but cannot match the

performance completely. TOVA also observes similar benefits in language modeling, shown Table 3.4, where we compare language modeling using LESS with H_2O, Λ-masking, and TOVA. We see that LESS makes a greater improvement on word perplexity than simply caching the equivalent number of extra tokens, though none of the methods can fully recover the original model's performance. While LESS is efficacious for language modeling and classification, we also want to assess its utility for generation where the KV cache storage becomes a critical bottleneck.

3.3.2 Summarization

Now, we move on to generation, specifically summarization, to test the ability to generate longer and coherent sequences by synthesizing numerous tokens. Unlike in our language modeling evaluations, the model will have access to all tokens during the prompting phase with the sparse policy and LESS only kicking in during the subsequent generation steps. Consequently, generation sparse policies are fundamentally different from the language modeling masks LESS is trained on, yet despite this, we show that our method maintains its superior performance.

In Tables 3.5 and 3.6, we see LESS achieves better ROUGE (Lin, 2004a) scores than purely H_2O on the CNN/DailyMail (Hermann et al., 2015; See et al., 2017), MultiNews (Fabbri et al., 2019), and XSum (Narayan et al., 2018) datasets. Even at exceptionally low sparsity levels, H_2O can capture a significant amount of the full cache's performance. This is even more surprising for Falcon models which already cache many times fewer tokens than Llama 2 due to the multi-query attention mechanism. Despite this, we observe LESS surpasses the already strong performance of H_2O across the board where H_2O underperforms compared to the full cache. Like in language modeling, we again see that the improvement from Baseline to Baseline+ pales in comparison to the improvement induced by LESS, sometimes even matching the full cache performance as in XSum. Again, we also see the transferability of LESS to other sparsity levels. See Section 3.4.5 for example generation outputs.

Table 3.5: Llama 2 13B and Falcon 7B generation quality comparison on CNN/DailyMail and XSum with 408 sparse tokens (10% and 20% of the context lengths of Llama 2 and Falcon, respectively) with H_2O as the primary underlying sparse policy. Llama 2 13B is given 5 shots while Falcon 7B is given 3 shots due to its shorter context length. Values are in the format [Rouge-1]/[Rouge-2]/[Rouge-L].

H_2O Method	CNN/DailyMail	XSum
Llama 2 13B		
Full Cache	27.55/9.96/25.80	33.14/13.05/27.33
Baseline	23.57/7.35/22.04	33.09/**13.09**/**27.44**
Baseline+	23.40/7.31/21.88	33.09/13.06/27.41
LESS (2%)	**25.27**/**7.76**/**23.64**	**33.40**/12.98/27.41
LESS (5%)	24.45/7.70/22.87	33.15/13.02/27.39
Falcon 7B		
Full Cache	25.92/8.52/24.15	27.17/8.83/22.67
Baseline	21.26/5.95/19.73	24.50/7.65/20.50
Baseline+	21.31/6.16/19.75	24.55/7.66/20.56
LESS (5%)	23.00/6.28/21.28	24.94/8.17/20.94
LESS (10%)	**23.22**/**6.37**/**21.53**	**25.21**/**8.28**/**21.17**

3.3.3 Efficiency

Following Sheng et al. (2023), we benchmark the generation throughput and latency of LESS on an NVIDIA A100 80G GPU using FP16 precision. We focus on the Llama 2 7B and 13B models, with all speedup results tested end-to-end with both prompting and generation phases. To measure its performance when generating long sequences or inputting large batch sizes, we use synthetic datasets where all prompts are padded to the same length and batched together. The same number of tokens are generated for each prompt. We test different combinations of prompt and generation lengths.

Table 3.7 shows results with sequence lengths from 4K to 10K. With the same batch size, LESS reduces the latency by $1.1 - 1.3\times$ compared to the full cache, though slightly slower than H_2O. Moreover, LESS saves memory to allow larger batch sizes with a $1.7\times$ improvement on generation throughput for Llama 2 7B, closely matching that of H_2O's.

We also show that maintaining and using the low-rank state in LESS has little overhead by determining the latencies of different operations in LESS. Quantified in Table 3.8, the to-

Table 3.6: Llama 2 7B performance on MultiNews (1-shot), CNN/DailyNews (5 shot), and XSum (5-shot) with 5% and 10% H_2O as the primary underlying test sparse policies. Values are in the format [Rouge-1]/[Rouge-2]/[Rouge-L].

H_2O Method	5% H_2O	10% H_2O
MultiNews		
Full Cache	23.79/6.87/21.35	23.79/6.87/21.35
Baseline	13.38/3.25/12.25	19.44/4.97/17.73
Baseline+	13.58/3.32/12.41	19.44/4.96/17.72
LESS (2%)	15.31/3.73/14.03	20.32/5.24/18.51
LESS (5%)	**15.42**/**3.80**/**14.14**	**20.55**/**5.29**/**18.70**
CNN/DailyMail		
Full Cache	26.25/9.34/24.40	26.25/9.34/24.40
Baseline	18.18/4.92/16.89	20.04/6.09/18.66
Baseline+	18.24/4.91/16.85	20.15/6.21/18.73
LESS (2%)	18.71/5.40/17.34	20.76/6.40/19.32
LESS (5%)	**19.21**/**5.44**/**17.80**	**22.29**/**6.85**/**20.69**
XSum		
Full Cache	30.65/11.11/25.40	30.65/11.11/25.40
Baseline	29.03/10.77/24.28	30.68/**11.54**/25.58
Baseline+	28.94/10.78/24.15	30.64/11.49/**25.59**
LESS (2%)	**30.72**/**11.53**/**25.57**	30.34/10.98/25.31
LESS (5%)	30.03/11.19/25.03	**30.82**/11.17/25.56

tal latency of LESS is much smaller than the full cache but slightly higher than Baseline+ due to the additional operations associated with the low-rank state. This overhead represents about 15% of the decoding time, meaning it does not significantly impact the overall efficiency.

3.4 Ablations & Analysis

In this section, we share some interesting characteristics of LESS beyond accuracy and efficiency. In particular, we show our method's improved quality of attention probability distribution reconstruction visually (Section 3.4.1) and quantitatively (Section 3.4.2), effect from larger kernels (Section 3.4.3), and robustness to long sequences (Section 3.4.4). Example generations can be found in Section 3.4.5.

Table 3.7: Llama 2 7B and 13B's generation throughput (tokens/s) and latency (s) on an A100 GPU. In the sequence length column, we use "$P + G$" to denote a prompt length of P and a generation length of G. "OOM" stands for out-of-memory.

Seq. length	Model size	Batch size	Full cache	Baseline+	LESS (5%)
Latency					
5000+5000	13B	4	257.3	185.2	204.7
2048+2048	7B	24	116.7	78.3	95.1
Throughput					
2048+2048	7B	24	421.2	627.7	516.9
2048+2048	7B	64	OOM	819.2	699.2

Table 3.8: Latency (s) breakdown for Llama 2 7B on an A100 GPU, with a batch size of 64, prompt length of 512, and generation length of 512 with 5% H_2O as the underlying sparse policy. "Eviction" refers to H_2O's KV eviction algorithm, "Kernels" refers to (3.1) and (3.2), "Attn Synth" refers to (3.3), and "LR Cache" refers to (3.4) and (3.5).

	Decoding	Eviction	Kernels	Attn Synth	LR Cache	Total
Full Cache	50.71	N/A	N/A	N/A	N/A	50.71
Baseline+	23.53	4.52	N/A	N/A	N/A	28.05
LESS (5%)	23.61	4.39	0.87	1.35	1.51	32.96

3.4.1 Attention Matrix Visualizations

This section provides some qualitative results on attention matrix approximations by sparse policies and LESS. While low-rank caches LESS cannot perfectly recover all the missing information, it visually is able to reconstruct a patterns that are completely ignored by sparse policies. We can also see the idiosyncrasies of the sparse policies and LESS, such as H_2O's bias towards keeping early tokens, as shown in Figures 3.6 and 3.7, and Λ-masking's tendency to miss influential tokens which are captured by LESS, as show in Figure 3.8.

3.4.2 Reconstructing Attention Probabilities

Numerically, we measure the similarity of each row in the attention matrix with corresponding rows produced by H_2O and LESS with the Hellinger distance, which for two discrete

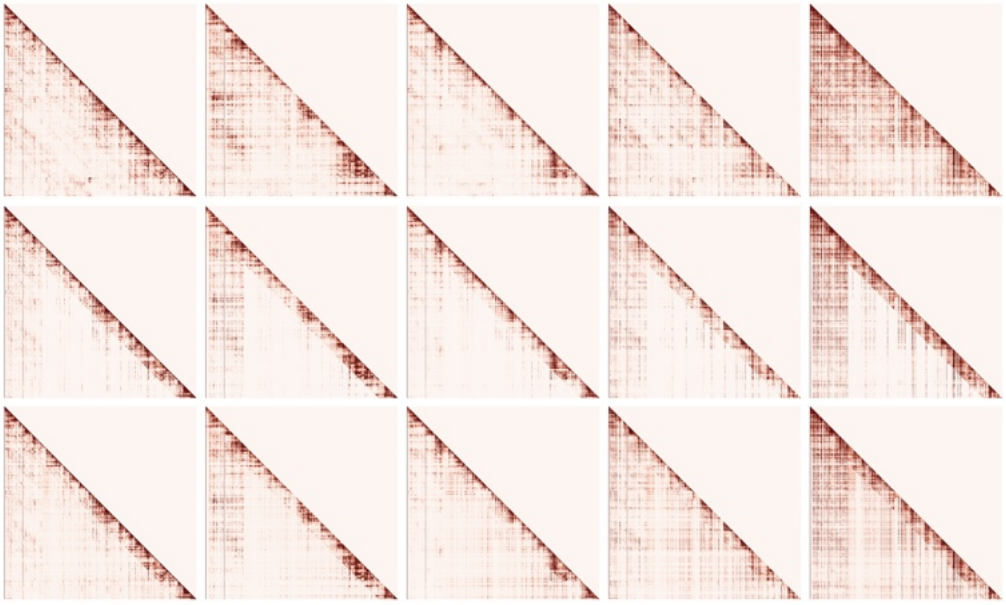

Figure 3.6: Example attention probability matrices from passing a single input into Falcon 7B. From top to bottom, the rows consist of attention maps from the original model, 10% H_2O (204 tokens), and `LESS` (10% H_2O). Darker pixels indicate larger probability weights. Only the first 1024 tokens are displayed.

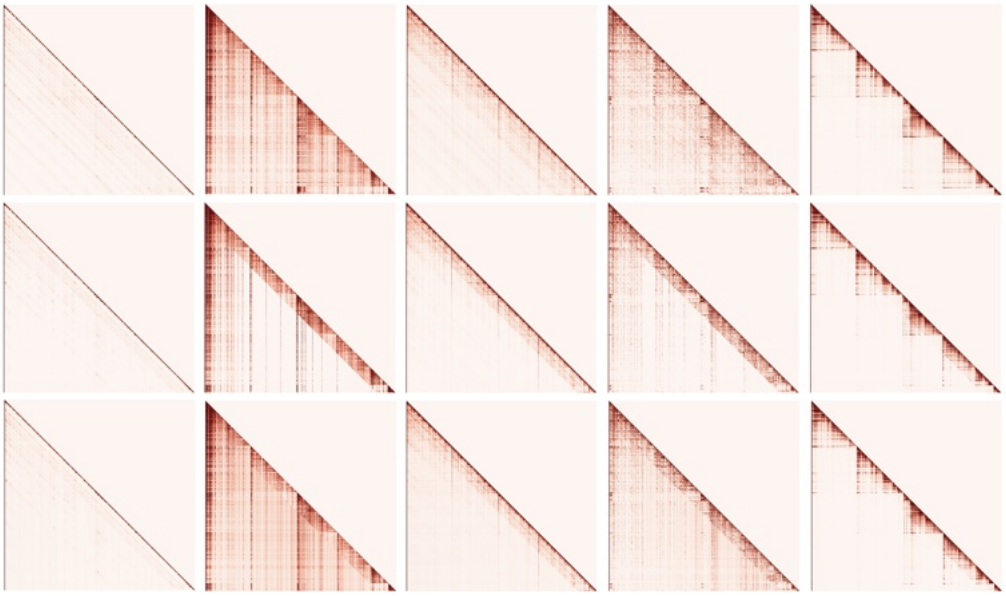

Figure 3.7: Example attention probability matrices from passing a single input into Llama 2 7B. From top to bottom, the rows consist of attention maps from the original model, 5% H_2O (204 tokens), and `LESS` (5% H_2O). Darker pixels indicate larger probability weights. Only the first 1024 tokens are displayed.

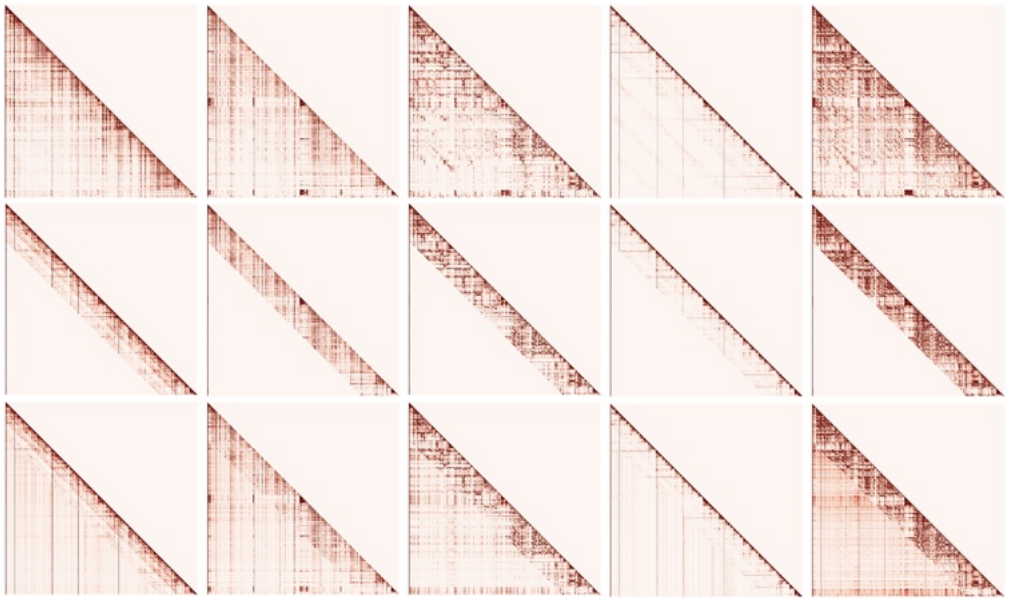

Figure 3.8: Example attention probability matrices from passing a single input into Llama 2 7B. From top to bottom, the rows consist of attention maps from the original model, 5% Λ-masking (204 tokens), and `LESS` (5% Λ). Darker pixels indicate larger probability weights. Only the first 1024 tokens are displayed.

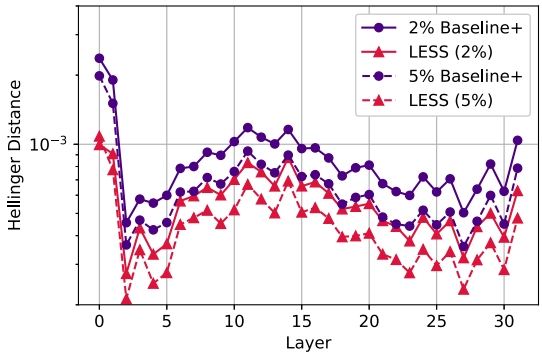

Figure 3.9: Layer-wise Llama 2 7B mean Hellinger distance from original attention probabilities, aggregated across WikiText evaluation samples. The underlying sparse policy is H_2O. Here, `LESS` is evaluated based on their training sparsity percentages.

probability vectors, $\boldsymbol{p}$ and $\boldsymbol{q}$, is defined as

$$\mathcal{H}(\boldsymbol{p}, \boldsymbol{q}) \coloneqq \|\sqrt{\boldsymbol{p}} - \sqrt{\boldsymbol{q}}\|_2 / \sqrt{2} \tag{3.7}$$

where the square root is elementwise. The value of $\mathcal{H}(\boldsymbol{p}, \boldsymbol{q})$ ranges from 0 to 1, where a lower value indicates greater similarity. In Figure 3.9, we see that our method more accurately

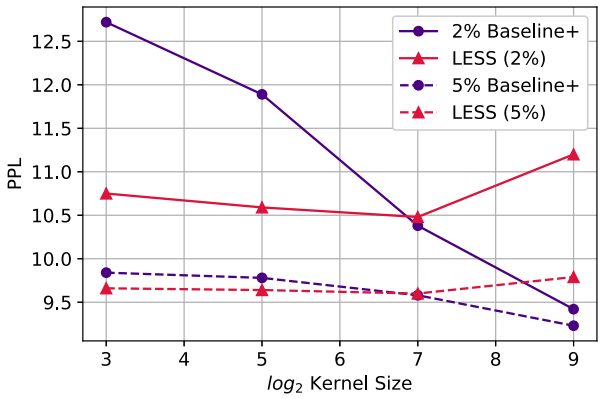

Figure 3.10: Llama 2 7B WikiText word perplexity (lower is better) as the kernel size quadruples, compared against `Baseline+` which occupies the same space. The sparse KV cache policy is H_2O.

replicates the original attention probability distributions as measured by the Hellinger distance. We choose to aggregate each layer separately since the attention distribution patterns tend to vary dramatically throughout the model.

3.4.3 Larger Kernels

In our experiments, we fixed $R = 8$, and as we show in Figure 3.10, the performance generally increases as R increases. However, at a certain point, the marginal benefit derived from increasing R is less than shifting more of the KV cache to the sparse policy, suggesting that a small low-rank cache is enough.

3.4.4 Providing Hope for Long Sequences

Model performance appears to be highly correlated with the input sequence length regardless of the caching method. As shown in Figure 3.11, even the full cache model performance drops dramatically and immediately as the prompt length increases. `Baseline+` and `LESS` (1% H_2O) appear to perform similarly for shorter sequences but diverge for longer sequences where we see `LESS` is more performative. This follows our intuition since for sparse cache policies, a smaller fraction of KV pairs is saved as the sequence length increases, so more

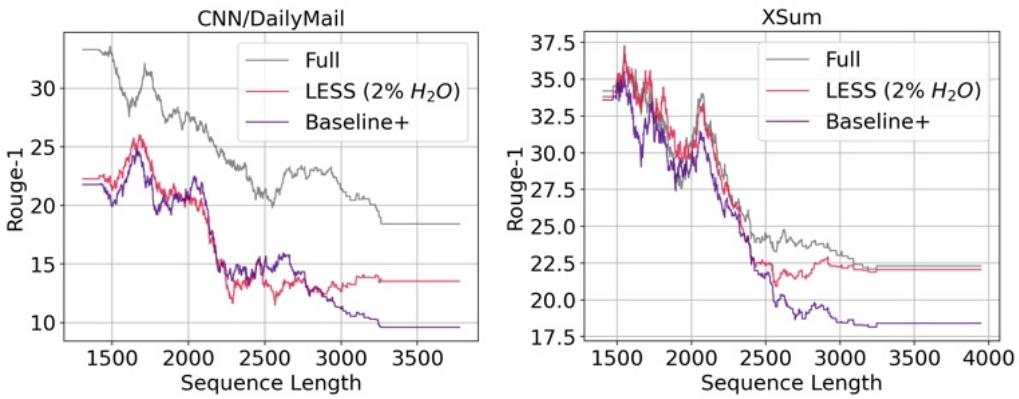

Figure 3.11: Relationship between Rouge-1 score and prompt length for Llama 2 7B with different cache methods on CNN/DailyMail (left) and XSum (right). The test sparse KV cache policy is 5% H_2O for all models. As these results can be fairly noisy, the lines are k-nearest regression lines where k is 10% of the dataset size.

information is omitted. This is where a low-rank state can help to recover some of this information.

3.4.5 Example Generation Outputs

We include a couple examples of generation outputs in Figure 3.12 and Figure 3.13. In both cases, the full cache, `LESS`, and `Baseline+` models attempt to summarize news articles. We see in Figure 3.12 that `LESS` is able to produce the same concise summary as the full cache while `Baseline+` produces rambling text. In Figure 3.13, we observe that `LESS` completely changes the meaning of the summary from H_2O alone–`Baseline+` is factually incorrect based on the article.

Prompt

Article: (CNN)Residents of central Sanaa, the Yemeni capital, have learned the hard way that key strategic bombing targets are located in their neighborhoods: Detonating ordnance has been shattering their windows and doors. ... Hadi's government had cooperated with the United States to fight AQAP, but with the Houthi takeover, that arrangement has evaporated, and the terror group operates generally unchecked.

Summarize the above article in 3 sentences.

Llama 2 7B: Full Cache

Saudi-led coalition bombs Yemen's Defense Ministry Central Command in capital Sanaa . Fighting has killed hundreds of people in less than two weeks . Yemen's deposed President Abdu Rabu Mansour Hadi heaped scorn on top of the airstrikes .

Llama 2 7B: `LESS` (2% H_2O)

Saudi-led coalition bombs Yemen's Defense Ministry Central Command in capital Sanaa . Fighting has killed hundreds of people in less than two weeks . Yemen's deposed President Abdu Rabu Mansour Hadi heaped scorn on top of the airstrikes .

Llama 2 7B: `Baseline+`

The Houthis have taken over Yemen's capital of Sanaa . The Saudi-led coalition has been bombing the Houthis and their allies . The fighting has killed hundreds of people . The Houthis are Shiite Muslims and the Saudis are Sunni Muslims . The Houthis have been fighting the government of Yemen's former President, Abed Rabbo Mansour Hadi, who is now in Saudi Arabia . The Saudi-led coalition is fighting to restore Hadi to power . The Houthis have been fighting the government of Yemen'*<cut off before reaching the stop token>*

Figure 3.12: Example 5-shot (not shown) CNN/DailyMail summary generation results produced by variations of Llama 2 7B with an underlying sparse policy of 2% H_2O. For brevity, only the start and end of the article are shown with the middle omitted with an ellipsis. `LESS` produces the same concise summary as the full cache while `Baseline+` produces rambling text, exceeding the 3 sentence requirement by the prompt. The original article is from (Brumfield, 2015).

Prompt

Article: The UK government has announced measures to speed up development.
Natural Resources Minister Carl Sargeant has written to councils reminding them of the temporary ban on fracking in Wales.
He said fracking technology was "unproven", saying Welsh ministers preferred to look at renewable energy.
Fracking is a process of using high pressure water to break up rocks deep underground to release gas and pipe it to the surface.
It is a major industry in the United States, but there are concerns about safety, the environment and underground water.
There have been protests against applications for test drilling for gas in both the Vale of Glamorgan and Wrexham, as well as sites in England.
Energy Secretary Amber Rudd has stressed the UK government's support for fracking, and frustration at delays in its development.
She has told councils in England to make decisions on applications within 16 weeks, to stop the process being "dragged out for months".
In his letter to Welsh councils, Mr Sargeant said the new guidance only applied to England and that the moratorium in Wales remained in place.
"The UK government's general support for oil and gas applications is contrary to the approach of the Welsh Government of promoting renewable and low carbon forms of energy through the planning system and other measures," he said.
"We still see renewable energy as a key element in ensuring that Wales achieves sustainable development for the benefit of future generations.
"Local planning authorities must ensure that planning applications for renewable energy projects are determined within statutory timescales."
Plans to devolve control over fracking were confirmed in the Queen's Speech in June following the Conservatives' general election victory.

Summarize the above article in 1 sentences.

Falcon 7B: Full Cache

The UK government has announced measures to speed up development of fracking in Wales.

Falcon 7B: `LESS` (10% H_2O)

The Welsh government has announced that it will not allow fracking in Wales.

Falcon 7B: `Baseline+`

The Welsh government has announced measures to speed up the development of fracking.

Figure 3.13: Example 3-shot (not shown) XSum summary generation results produced by variations of Falcon 7B. Models were evaluated with 20% H_2O. The summary by `Baseline+` is factually incorrect based on the article, while `LESS` preserves the meaning better. The original article is from (BBC, 2015).

3.5 Chapter Summary

To tackle the KV cache bottleneck, we introduced `LESS` which has demonstrated itself to be an effective way to boost eviction-based KV cache algorithms. Motivated by the necessity to maintain information that would have been discarded, the constant-sized `LESS` recovers a significant portion of the performance lost due to maintaining a small cache across a variety of scenarios and intensities, despite being cheap to train and deploy. There are many exciting avenues of work that can enhance `LESS` or build upon it, such as improving kernel design and investigating the residual of `LESS`. Such directions will further push the performance of a condensed KV cache to that of a complete cache, allowing LLMs to accomplish the same tasks with less. In the next chapter, we will look at how to improve inference efficiency in small batch regimes, common in local models.

Chapter 4

Adaptive Structured Pruning

As we look at other sources of inefficiencies, FF blocks jump out as wasteful layers which perform large dense operations to produce incredibly sparse features. However, their seemingly chaotic patterns have been a major roadblock for practically efficient methods. In this chapter, we demonstrate that a novel transformation of FF features illuminates predictable structure which can be extrapolated for fast inference. This chapter explores Dong et al. (2024a) and uses some observations from Dong et al. (2023a).

4.1 Chasing Sparsity in FF Blocks

FF blocks consist of approximately 2/3 of the total number of parameters in LLMs. Using all of these parameters for every input can be serious memory and compute bottlenecks. These inefficiencies are highly problematic in latency-sensitive scenarios like in chatbots and autonomous vehicles. It turns out that LLM FF blocks waste computation on intermediate features with little to no impact on the final result due to the existence of sparse structures within them (Dettmers et al., 2022; Geva et al., 2020; Li et al., 2022; Liu et al., 2023c). For instance, it has been observed that in OPT-175B (Zhang et al., 2022), fewer than 5% of neurons in FF blocks have nonzero values per token (Liu et al., 2023c), meaning 95% of the compute in each FF block is wasted.

There have been many methods to exploit sparsity in LLMs for efficiency gains, such

as pruning model weights and constructing mixtures of experts (MoEs). Pruning removes low-impact pre-trained weights to reduce storage, yet this often does not translate to real speed-ups in practice, unless the pruning is done in a structured and hardware-friendly manner (Li et al., 2023; Ma et al., 2023; Santacroce et al., 2023; Xia et al., 2023, 2022) which typically causes greater performance deterioration. MoEs better preserve the original performance by adaptively selecting subsets of the model to use per input but also come with drawbacks. Unless the model has been trained in this fashion (Fedus et al., 2022; Jiang et al., 2024; Sadhukhan et al., 2026), it needs to learn a cheap yet effective gating function (expert selection mechanism) and sometimes even require full fine tuning. Perhaps an even bigger weakness of many of these methods is the limited effectiveness and special considerations to strictly enforce ReLU-like activation functions (Csordás et al., 2023; Li et al., 2022; Zhang et al., 2021). In summary, pruning and MoEs provide enormous benefits but also come with steep challenges:

1. Structured sparsity is difficult and costly to enforce without serious performance degradation.
2. Adaptive selection of model subsets need to be cheap and accurate.
3. The method should be effective with various activation functions.

Daunting at first, these challenges become surmountable because of a simple observation of a phenomenon we call *flocking*, highly consistent sparse activations that persist throughout a sequence observed in many LLMs. *Flocking emerges in FF activations (inputs into the FF down projection) when we focus on a sequence's relative activation magnitudes instead of the absolute values.* Examples from Llama 2 7B and Gemma 7B are shown in Figure 4.1. *The key takeaway is that neurons which produce high relative magnitudes are naturally shared across tokens within a sequence*, as seen with the distinct vertical streaks. Increasingly bizarre, the two models have different architectures and different non-ReLU activation functions.

Unlike existing pruning or MoE methods, we exploit flocking in our design of `GRIFFIN` (**G**ating by **R**epetition **I**n **F**eed**f**orward **I**ntermediate **N**eurons), *a highly performative and efficient training-free method to adaptively activate FF neurons.* `GRIFFIN` does this by

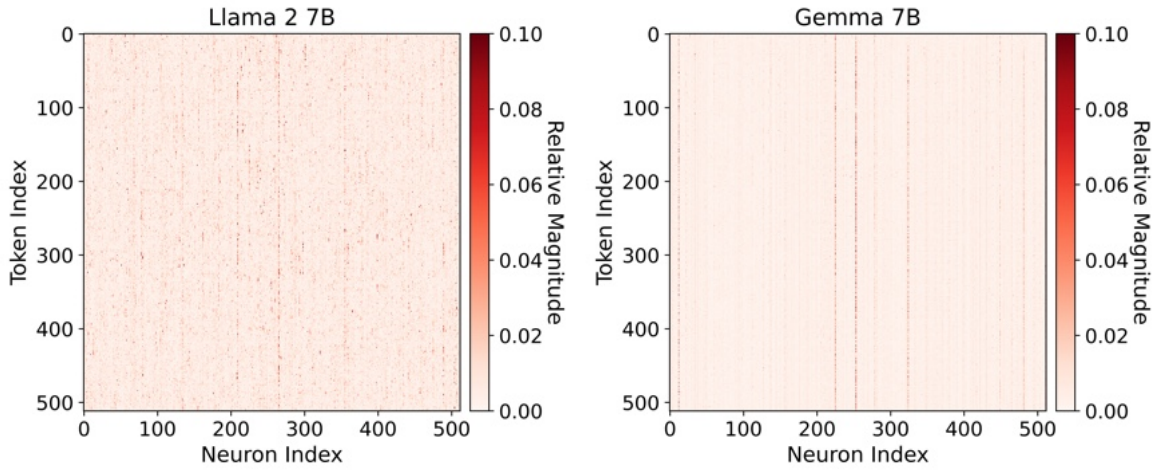

Figure 4.1: Relative FF activation magnitudes of the first 512 features and tokens across a sequence from PG-19 (Gao et al., 2020; Rae et al., 2019) in layer 10 of Llama 2 7B (left) and Gemma 7B (right). These heatmaps show flocking, where relative activation magnitudes are shared within a sequence, illustrated with the distinct dark vertical streaks. More examples in Section 4.6.3.

using a sequence's prompt to determine the experts to activate during generation, allowing it to overcome all of the aforementioned challenges:

1. **No Preparation:** Our no-cost method is completely training-free and requires no calibration. Moreover, the simple implementation of GRIFFIN means it can be dropped into any FF block and instantly deployed.

2. **Simple Expert Selection:** Flocking in the prompt reveals the most relevant FF neurons for generation with little to no performance loss. The selection process is parameter-free and adds negligible overhead.

3. **Model & Activation Function Diversity:** Thorough experimentation demonstrates the efficacy of `GRIFFIN` on numerous models, including Llama 2 (Touvron et al., 2023), Gemma (Team et al., 2024b), Mistral (Jiang et al., 2023), OPT (Zhang et al., 2022), and ReluLlama (Team, 2023). Together, the tested activation functions include ReLU, SwiGLU, GEGLU, and ReGLU (Shazeer, 2020).

In this chapter, we show `GRIFFIN` is a simple and strong adaptive structured pruning method because of flocking. Next, we formalize the problem we are trying to solve, along with its motivation in Section 4.2. Then, we present our novel approach in Section 4.3.2,

which requires a thorough examination of the surprising phenomenon of flocking shared by many LLMs in Section 4.3.1. Our rigorous experiments demonstrate GRIFFIN preserves performance on classification and generation even after removing 50% of FF neurons (Section 4.4.1), all while having lower latency (Section 4.4.2). For instance, GRIFFIN reduces the number of active parameters in Llama 2 13B from 13B to 8.8B during generation to improve latency by 1.25× with almost no loss in performance. For a demonstration of our method's scalability and robustness in several ablation studies, see Section 4.5. Additionally, for a deeper exploration on flocking, see Section 4.6.

4.2 Formulating FF Variants

This section contains a formulation of the FF compression problem which our method aims to tackle. With bias terms omitted for brevity, recall from Section 2.1 that an FF block operates on the input $\boldsymbol{X} \in \mathbb{R}^{S \times D}$ like so:

$$\mathrm{FF}(\boldsymbol{X}) = \mathrm{FF}_2(\underbrace{\mathrm{FF}_1(\boldsymbol{X})}_{\boldsymbol{Z}}) \tag{4.1}$$

where $\mathrm{FF}_2(\boldsymbol{Z}) = \boldsymbol{Z}\boldsymbol{W}_2$ is a linear transformation and FF_1 is nonlinear. FF blocks usually take the form of

$$\mathrm{FF}_1(\boldsymbol{X}) = \sigma(\boldsymbol{X}\boldsymbol{W}_1), \tag{4.2}$$

like in OPT or

$$\mathrm{FF}_1(\boldsymbol{X}) = \sigma(\boldsymbol{X}\boldsymbol{W}_g) \odot (\boldsymbol{X}\boldsymbol{W}_1), \tag{4.3}$$

for GLU variants like in Llama 2 and Gemma. For all, $\boldsymbol{W}_1, \boldsymbol{W}_g \in \mathbb{R}^{D \times D_{\mathrm{FF}}}$ and $\boldsymbol{W}_2 \in \mathbb{R}^{D_{\mathrm{FF}} \times D}$ where typically, $D_{\mathrm{FF}} \gg D$.

A popular method to compress FF blocks (which is also the goal of GRIFFIN, adaptive neuron pruning, and many MoE methods) is to find $\widehat{\boldsymbol{W}}_1 \in \mathbb{R}^{k \times D}$ and $\widehat{\boldsymbol{W}}_2 \in \mathbb{R}^{D \times k}$ (addi-

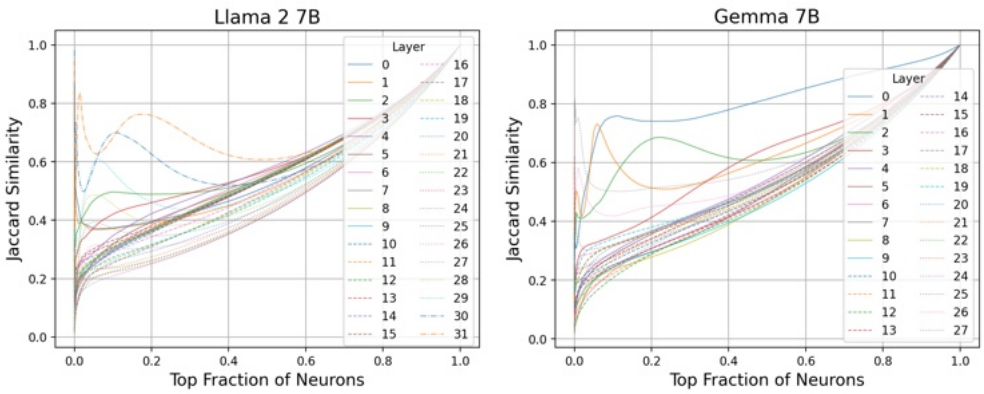

Figure 4.2: Average Jaccard similarity between WikiText samples' top FF neuron activations in Llama 2 7B (left) and Gemma 7B (right). Higher values indicate greater similarity.

tionally $\widehat{\boldsymbol{W}}_g \in \mathbb{R}^{k \times D}$ for GLU networks) where $k < D_{\text{FF}}$ such that when the FF block is reparameterized with these matrices, the output value is preserved. Namely, for

$$\widehat{\boldsymbol{Z}} = \widehat{\text{FF}}_1(\boldsymbol{X}) = \sigma(\boldsymbol{X}\widehat{\boldsymbol{W}}_g) \odot (\boldsymbol{X}\widehat{\boldsymbol{W}}_1), \tag{4.4}$$

$$\widehat{\text{FF}}_2(\widehat{\boldsymbol{Z}}) = \widehat{\boldsymbol{Z}}\widehat{\boldsymbol{W}}_2, \tag{4.5}$$

$\text{FF}(\boldsymbol{X}) \approx \widehat{\text{FF}}_2(\widehat{\text{FF}}_1(\boldsymbol{X}))$, and similarly for FF blocks with non-GLU functions. For instance, in the MoE setting, these smaller matrices can vary from token to token and are actually selections of chunks of the original structures. Crucially, a solution to this problem leads to multiplication with smaller matrices which are naturally more efficient on GPUs and TPUs (Fatahalian et al., 2004; Wang et al., 2019).

4.3 GRIFFIN: Adaptive Test-time Structured Pruning

Here, we take deeper dive into the phenomenon of flocking and describe the intuitive algorithm of GRIFFIN which is directly inspired by it.

4.3.1 Observing Flocking

Flocking arises when we look at the relative impact of each neuron per token within a sequence. To see this, we normalize rows of $\boldsymbol{Z}$ to be unit vectors to construct $\overline{\boldsymbol{Z}} \in \mathbb{R}^{S \times D_{\text{FF}}}$ (i.e. $[\overline{\boldsymbol{Z}}]_i = [\boldsymbol{Z}]_i / \|[\boldsymbol{Z}]_i\|_2$), the *relative activations*. We show example relative activation magnitudes for a sequence in Llama 2 7B and Gemma 7B in Figure 4.1. Since there are distinct vertical streaks, this intriguingly implies that activations that have relatively greater weight are common across all tokens in a sequence. Notably, Llama 2 7B and Gemma 7B use SwiGLU and GEGLU activations, respectively, along with other major architecture differences. We call this phenomenon flocking, like highly organized groups of birds, and we observe this in virtually all FF layers (see Section 4.6.3) and randomized inputs (see Section 4.6.2).

While relative activations magnitudes are shared within a sequence, they are not generally shared between sequences. We show this by taking the ℓ_2-norm of $\overline{\boldsymbol{Z}}$ along the token axis to obtain a length D_{FF} vector for each sample or sequence, roughly capturing the contribution of each FF neuron throughout a sequence. Taking the top-k of this for each sample at each layer, we find the Jaccard similarity between two sequences based on the indices selected for different k. In other words, we compute the intersection over union of every unique pair of top-k sets. Higher values indicate more similar top-k sets. From Figure 4.2 where we aggregate Jaccard similarities across WikiText (Merity et al., 2016) samples, we observe a lack of inter-sample activation similarities for the vast majority of layers in Llama 2 7B and Gemma 7B, unless the sets of selected neurons are large. *This lack of consistency implies statically pruning entire FF neurons without retraining would be less effective than a more adaptive method.*

4.3.2 `GRIFFIN` Algorithm

Using our insight on flocking, we introduce `GRIFFIN` as a simple general purpose and training-free adaptive structured pruning method for efficient generation, captured in Figure 4.3. In a nutshell, we select neurons during the prompt phase of each sample which are then used for the entire duration of the generation phase. This effective approach is based

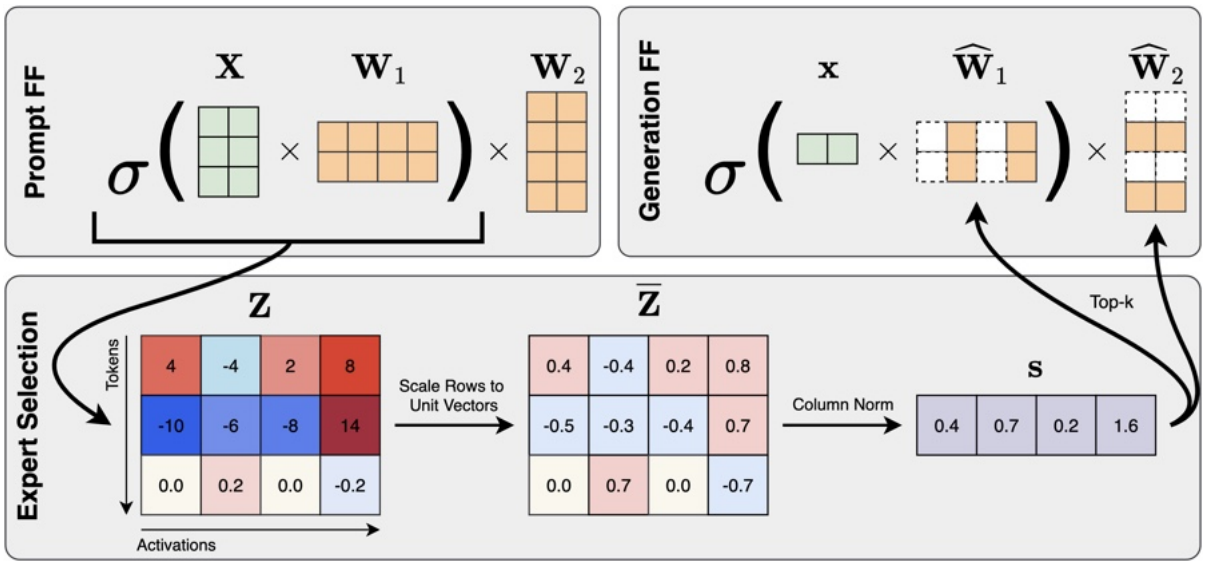

Figure 4.3: `GRIFFIN` overview. Relative activations from the prompt determine expert neurons to use for generation.

on a key observation on flocking: *since tokens within a sequence share activation patterns, the prompt and generated tokens will also share activation patterns.*

Prompt Phase Expert Neuron Selection

Our expert neurons are chosen at the sequence level, so we need to consider the dynamics of the entire input sequence rather than just a single token when choosing our neurons. To select expert neurons, we need a statistic $\boldsymbol{s} \in \mathbb{R}^{D_{\text{FF}}}$ to inform us of the importance of each neuron. At the prompt phase, we do this by taking the ℓ_2-norm of $\overline{\boldsymbol{Z}}$ along the token axis:

$$\boldsymbol{s} = \begin{bmatrix} \|[\overline{\boldsymbol{Z}}]_{\cdot,1}\|_2 & \cdots & \|[\overline{\boldsymbol{Z}}]_{\cdot,D}\|_2 \end{bmatrix}^\top. \tag{4.6}$$

Taking the indices of the top-k across $\boldsymbol{s}$ gives us the neurons we will use for this sample's generation phase which make up the set $\mathcal{E}$. Using the expert neurons in $\mathcal{E}$, we can find $\widehat{\boldsymbol{W}}_1, , \widehat{\boldsymbol{W}}_g$, and $\widehat{\boldsymbol{W}}_2$ by selecting corresponding rows and columns in $\boldsymbol{W}_1, \boldsymbol{W}_g$, and $\boldsymbol{W}_2$, respectively. Similarly, we can prune bias terms $\boldsymbol{b}_1, \boldsymbol{b}_g \in \mathbb{R}^{D_{\text{FF}}}$ to obtain $\widehat{\boldsymbol{b}}_1, \widehat{\boldsymbol{b}}_g \in \mathbb{R}^{D_k}$. This is done for every FF block during the prompt phase. Illustrated in Section 4.6.1, $\boldsymbol{s}$ highlights neurons consistently activated at relatively high intensities.

Generation with Expert Neurons

When generating tokens, we directly use the pruned layers which contain the expert neurons, $\widehat{\text{FF}}_1$ and $\widehat{\text{FF}}_2$, to estimate $\text{FF}(\boldsymbol{X}) \approx \widehat{\text{FF}}_2(\widehat{\text{FF}}_1(\boldsymbol{X}))$ for all future tokens. In Llama 2 13B and Gemma 7B, this reduces the active number of parameters from 13B to 8.8B and from 8.5B to 5.4B, respectively, during generation.

4.4 Core Experiments

We showcase the superb performance of `GRIFFIN` on numerous tasks and models (Section 4.4.1) while achieving lower latency (Section 4.4.2). All experiments are run on NVIDIA L40 GPUs.

4.4.1 Performance

Using various models, we evaluate on several generation and classification tasks. For generation, we evaluate on XSum (Narayan et al., 2018), CNN/DailyMail (Nallapati et al., 2016), COQA (Reddy et al., 2019), and SCROLLS QASPER (Dasigi et al., 2021; Shaham et al., 2022). For classification, we evaluate on HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), COPA (Roemmele et al., 2011), ARC-Easy/Challenge (Clark et al., 2018), and BoolQ (Clark et al., 2019). With the exception of XSum and CNN/DailyMail, we use LM Evaluation Harness for our experiments (Gao et al., 2023). Aside from comparing with the original LLM, we also compare `GRIFFIN` with a static neuron pruning method based on neuron magnitudes. Similar to neuron magnitude pruning, this baseline selects expert neurons based on neuron magnitudes in $\boldsymbol{W}_1$ for the generation phase but uses the entire FF blocks for prompting like `GRIFFIN`. In the case of GLU variants, the neuron-wise norms of $\boldsymbol{W}_1$ and $\boldsymbol{W}_g$ are elementwise multiplied to produce the pruning metric. As we will see, this straightforward baseline achieves great classification results but falters for generation. Another baseline we consider also uses the full model for the prompt and Wanda (Sun et al., 2023a) in FF blocks for generation, using the prompt activations to prune FF weights. We refer to this adaptive *unstructured* pruning baseline as Adaptive

Table 4.1: Classification tasks (0-shot unnormalized accuracy) at 50% FF sparsity.

Model	HellaSwag	PIQA	COPA	ARC-e	ARC-c	BoolQ
Llama 2 7B	57.16	78.07	87.00	76.35	43.34	77.71
Magnitude	57.12	77.31	84.00	70.33	40.27	66.54
GRIFFIN	57.11	77.69	86.00	74.54	42.75	73.15
Llama 2 13B	60.06	79.05	90.00	79.46	48.46	80.61
Magnitude	60.00	79.00	88.00	74.07	46.25	70.52
GRIFFIN	60.10	79.11	89.00	77.19	46.84	78.50
Gemma 7B	60.61	80.30	88.00	82.74	50.09	83.49
Magnitude	46.24	73.12	57.00	45.20	32.76	62.84
GRIFFIN	60.62	79.98	88.00	81.65	50.09	81.90
Mistral 7B	61.21	80.58	92.00	80.89	50.43	83.61
Magnitude	61.15	80.36	86.00	74.20	48.89	60.40
GRIFFIN	61.18	80.52	91.00	79.25	50.00	80.06
OPT 6.7B	50.48	76.28	81.00	65.53	30.55	66.12
Magnitude	49.21	72.63	79.00	47.60	27.13	40.15
GRIFFIN	50.44	75.63	80.00	63.93	30.55	65.44
OPT 13B	52.46	75.90	86.00	67.05	32.94	65.81
Magnitude	51.31	74.21	81.00	49.41	28.07	38.75
GRIFFIN	52.42	76.17	86.00	66.92	33.19	67.65

Wanda. Note that Adaptive Wanda is completely unstructured and does not reduce the FF activation dimension at all.

As our method is designed specifically for generation, we alter classification evaluations to simulate generation. In typical classification tasks, LLMs do not enter the generative phase since the final token output of the prompt phase indicates the class. Consequently, directly applying GRIFFIN for classification tasks trivially yields the exact performance of the original model. Therefore, we treat all tokens but the final input token as the prompt. Then, the model is forced to go into the generation phase for one step.

We start with a look into the relationship between the sparsity levels and performance degradation. This translates to varying k when we select the top-k of our statistic $\boldsymbol{s}$. To compare the performance degradation across multiple tasks, we plot the ratio of the final performance metrics between GRIFFIN and the full model in Figure 4.4. We see most of

Table 4.2: Generation tasks XSum (1-shot), CNN/DailyMail (1-shot), CoQA (0-shot), SCROLLS QASPER (0-shot) at 50% FF sparsity. Magnitude neuron pruning fails in almost every case while GRIFFIN and Adaptive Wanda effectively preserve performance, though unlike GRIFFIN, Adaptive Wanda is completely unstructured.

Model	XSum (Rouge-1/2/L)	CNN/DailyMail (Rouge-1/2/L)	CoQA (F1/EM)	QASPER (F1)
Llama 2 7B	27.15/9.06/22.62	10.08/0.13/9.55	77.35/63.88	26.31
Magnitude	9.71/1.31/8.59	9.66/0.63/9.32	56.59/39.93	12.93
Adaptive Wanda	25.59/8.18/21.34	9.90/0.26/9.39	77.16/63.53	27.61
GRIFFIN	24.75/7.41/20.55	10.97/0.66/10.37	77.18/63.58	25.76
Llama 2 13B	26.90/9.45/22.09	2.51/0.22/2.34	79.18/66.37	28.32
Magnitude	5.72/0.78/5.06	0.02/0.00/0.02	65.69/47.87	15.55
Adaptive Wanda	24.65/8.08/20.29	1.13/0.11/1.07	79.43/67.43	27.98
GRIFFIN	25.69/7.85/20.89	3.31/0.78/3.07	79.22/66.62	27.91
Gemma 7B	26.86/9.15/22.03	17.45/4.15/15.94	79.04/65.25	30.78
Magnitude	1.49/0.01/1.47	0.00/0.00/0.00	2.92/1.50	7.02
Adaptive Wanda	24.76/7.79/20.39	12.10/2.21/11.20	75.79/61.87	30.62
GRIFFIN	25.86/7.81/20.93	18.26/4.75/16.58	78.52/64.62	27.37
Mistral 7B	28.67/10.21/23.64	0.28/0.01/0.28	80.70/67.30	24.56
Magnitude	3.58/0.27/3.31	0.26/0.03/0.26	61.99/45.93	17.18
Adaptive Wanda	25.42/8.40/21.16	0.16/0.00/0.14	80.68/67.60	24.73
GRIFFIN	26.59/8.70/22.17	1.26/0.21/1.17	80.15/66.50	23.92
OPT 6.7B	23.60/7.04/19.46	13.85/1.54/13.04	68.70/54.98	18.53
Magnitude	1.63/0.00/1.54	1.20/0.00/1.17	31.53/16.52	7.28
Adaptive Wanda	22.40/6.04/18.57	12.94/1.14/12.23	69.01/55.22	18.47
GRIFFIN	21.17/5.42/17.58	13.01/1.06/12.26	68.99/55.00	17.40
OPT 13B	25.14/7.93/20.80	13.22/1.18/12.46	69.51/55.67	20.58
Magnitude	1.23/0.00/1.21	1.29/0.00/1.29	39.38/27.07	8.87
Adaptive Wanda	23.68/6.97/19.71	13.58/1.48/12.82	69.60/56.03	21.30
GRIFFIN	22.11/6.28/18.29	12.92/1.13/12.20	69.07/54.83	20.16
ReluLlama 2 7B	25.10/7.81/20.76	20.95/6.79/19.24	78.49/66.73	23.31
Magnitude	9.09/0.22/8.20	8.50/0.14/8.17	19.43/6.48	7.21
Adaptive Wanda	22.62/6.47/18.82	17.46/5.56/16.13	78.96/66.97	21.38
GRIFFIN	21.83/5.88/18.09	16.85/4.96/14.69	78.35/67.10	22.29

the performance is preserved at 50% FF sparsity in Llama 2 7B, Gemma 7B, and Mistral 7B. Different tasks have different tipping points where performance sharply drops, which may be related to the difficulty of the task (Yin et al., 2024).

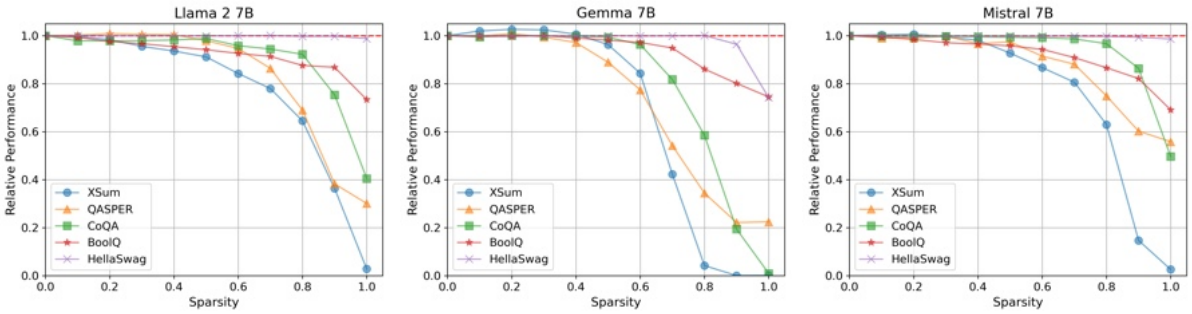

Figure 4.4: Relative performance of GRIFFIN for Llama 2 7B (left), Gemma 7B (center), and Mistral 7B (right) as we enforce varying degrees of sparsity per FF block. For all tasks, the original model's performance for each task is normalized to 1.

Fixing FF sparsity to be 50%, we evaluate on more tasks and models. Table 4.1 and Table 4.2 show the performance of GRIFFIN on classification and generation, respectively. We see that magnitude neuron pruning achieves reasonable results for classification but completely annihilates the original performance in most generation settings. For generation, Adaptive Wanda and GRIFFIN perform similarly, preserving most of the performance for all models. However, GRIFFIN is able to accomplish this in a structured manner which allows it to be more efficient in practice. For example outputs, see Section 4.5.4. Base models Mistral 7B and Llama 2 13B perform poorly on 1-shot CNN/DailyMail, as they often repeated "\n" or items from the example shot. With 3 shots, both obtain Rouge-1 scores above 25.

4.4.2 Efficiency

We now present efficiency metrics of GRIFFIN. We collect synthetic datasets with samples having identical lengths and average results across samples. Like many MoE methods, GRIFFIN is ideal for single sample inputs since it is adaptive, such as in the case of personal devices, so we use batch size 1 for these experiments. Using Hugging Face implementations of Llama 2 13B and Gemma 7B at FP16 precision, we measure the latency in different scenarios on an NVIDIA L40 GPU.

Recalling that our magnitude selection baseline is essentially neuron pruning at generation, this has the best possible speed-up since there is no expert neuron selection overhead

Table 4.3: Generation phase latency (s). We denote "$P + G$" as the task of generating G tokens from a length P prompt. When relevant, times are in the format 50% / 75% FF sparsity.

Model	Setup	Prompt	Full	Magnitude	GRIFFIN
Llama 2 13B	2048+128	0.5	6.8	5.4 / 5.0	5.4 / 5.1
	2048+2048	0.5	119.1	95.0 / 83.4	94.9 / 82.8
Gemma 7B	2048+128	0.3	4.5	4.1 / 4.2	4.2 / 4.1
	2048+2048	0.3	87.1	67.7 / 65.0	67.4 / 65.0

per sample. From Table 4.3, GRIFFIN matches the best case, producing up to a 1.29× and 1.25× improvement in latency for long generation at 50% FF sparsity in Gemma 7B and Llama 2 13B, respectively. This illustrates that our method is as fast as a static neuron pruned LLM during generation while being adaptive to preserve the accuracy of the full model. In offloading settings with large models, our method has the potential to further accelerate inference. For a prompt, GRIFFIN essentially performs structured pruning on the massive network, and if this pruned model can fit on a single device, it will avoid offloading for the entirety of generation.

4.5 Ablations & Analysis: GRIFFIN

Given GRIFFIN's computational benefit, we now explore its behavior in different settings, including the tradeoff between prompt and generation length in Section 4.5.1, the performance with larger batch sizes in Section 4.5.2, and a comparison with stochastic methods in Section 4.5.3. In addition, we include example outputs in Section 4.5.4.

4.5.1 Prompt vs. Generation Length

We find that GRIFFIN can potentially be made more robust for long generation by lengthening the prompt. To see this, we use language modeling on the concatenated version of WikiText to simulate generation. For a length S input into the FF block, we designate the first P tokens as the prompt and the last G tokens as the generated portion such that

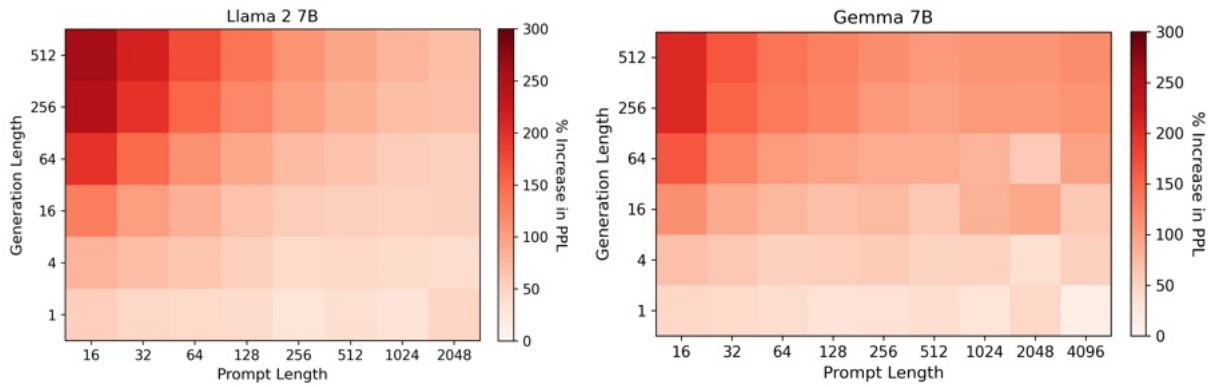

Figure 4.5: Prompt length vs. generation length for Llama 2 7B (left) and Gemma 7B (right) as measured by increase in perplexity (PPL) from the full model on concatenated WikiText at 50% FF sparsity.

Table 4.4: Rouge-1 scores of 1-shot XSum with varying ways to use the same selected neurons across multiple samples. From left to right, the scores are the result of the full model, FF neurons selected based on the prompt, FF neurons selected based on the entire dataset, and `GRIFFIN` in batch sizes of 1, 4, and 16. All pruning methods use the full model for the prompt and 50% of the FF neurons during generation.

Model	Full	Shot	Global	`GRIFFIN` (1)	`GRIFFIN` (4)	`GRIFFIN` (16)
Llama 2 7B	27.15	21.11	23.28	24.75	24.26	23.75
Gemma 7B	26.86	19.76	22.48	25.86	25.01	24.37

$P + G = S$. The prompt partition is used to calculate our statistic $\boldsymbol{s}$ and determine the expert neurons. The prompt partition uses the full FF block while the generation partition only uses the selected neurons. When comparing the original model with `GRIFFIN`, we only compute the perplexity of the outputs from the generation partition since the other outputs will be identical. Based on Figure 4.5, `GRIFFIN` gets closer to the full model outputs when the prompt length increases and generation length decreases, meaning the difficulty with long generation can be suppressed with longer prompts.

4.5.2 Sharing Selected FF Neurons & Batching

We further verify that `GRIFFIN` is simultaneously robust to batching and more performative than fixed neuron pruning methods using similar pruning metrics to $\boldsymbol{s}$ in (4.6). As such,

GRIFFIN maintains its reliability as we extend beyond single inputs.

For static pruning, we test two methods. One is to use the example shot in the prompt to compute $\boldsymbol{s}$. Then, for all samples with the same shot, we select the same FF neurons for generation. With the other method, we can extend (4.6) such that $\bar{\boldsymbol{s}}$ can be a pruning metric for multiple inputs (i.e. the same FF neurons will be pruned). Suppose that $\boldsymbol{s}_i$ is the metric for sample i with prompt length S_i. Then, we use

$$\bar{\boldsymbol{s}} = \sum_i \frac{\boldsymbol{s}_i}{\sqrt{S_i}} \tag{4.7}$$

to select FF neurons. Finding a global $\bar{\boldsymbol{s}}$ across all prompts in a dataset is the second method. Therefore, for a particular dataset, both methods have fixed expert neurons across samples.

To extend GRIFFIN to larger batch sizes, we use $\bar{\boldsymbol{s}}$ from (4.7) where the samples in a batch are aggregated. Thus, adaptability is maintained. From Table 4.4, GRIFFIN (even with batch size >1) achieves better performance than the static methods, reinforcing the performance benefit of adaptability. Interestingly, even though performance decays as we increase the batch size, the decay is slow.

4.5.3 Sampling-based Selection

Here, we verify that given the statistic $\boldsymbol{s}$, top-k expert selection produces better results than sampling-based methods. The methods we compare against include sampling based on the weights in $\boldsymbol{s}$ and combining top-k selection for half of the experts followed by weighted sampling. Based on Table 4.5, we can see that sampling generally degrades performance much more.

4.5.4 Generation Examples

We show example generated text in Figure 4.6. Adaptive Wanda and GRIFFIN produce similar summaries compared to the those produced by the full models and the target.

Table 4.5: Comparison between different expert selection methods at 50% FF sparsity.

Method	XSum (Rouge-1/2/L)	CNN/DailyMail (Rouge-1/2/L)	CoQA (F1/EM)	QASPER (F1)
Llama 2 7B				
Full	27.15/9.06/22.62	10.08/0.13/9.55	77.35/63.88	26.31
Top-k	**24.75/7.41/20.55**	**10.97/0.66/10.37**	**77.18**/63.58	**25.76**
Sample	21.04/5.22/17.12	8.78/0.49/8.28	76.15/62.53	24.46
Top-k+Sample	24.35/7.08/20.07	10.45/0.48/9.88	77.12/**64.17**	25.22
Gemma 7B				
Full	26.86/9.15/22.03	17.45/4.15/15.94	79.04/65.25	30.78
Top-k	**25.86/7.81/20.93**	**18.26/4.75/16.58**	**78.52/64.62**	**27.37**
Sample	20.25/5.16/16.79	8.34/1.71/7.72	75.02/59.93	24.97
Top-k+Sample	24.47/7.43/19.98	10.93/2.60/9.98	76.76/62.12	27.09

4.6 Ablations & Analysis: Flocking

Furthermore, we investigate various properties of the flocking phenomenon, whose utility could extend beyond `GRIFFIN`. We explore the dynamics of gating statistic $\boldsymbol{s}$ (Section 4.6.1), random sequences (Section 4.6.2), and layer-to-layer visualizations (Section 4.6.3).

4.6.1 Gating Statistic

Here we present visualizations of our gating statistic $\boldsymbol{s}$ from (4.6). For a single sample, we find $\boldsymbol{s}$ and sort the entries normalized between 0 and 1 in Figure 4.7. In both models, values in $\boldsymbol{s}$ are heavily concentrated in a handful of features. Since $\boldsymbol{s}$ aggregates the relative activation magnitudes across tokens, this implies $\boldsymbol{s}$ can capture heavily and frequently activated neurons.

4.6.2 Sparsity in Random Sequences

As further exploration into flocking, we investigate this phenomenon with random inputs. As input sequences, we use a sample from concatenated WikiText, a permuted version of that sample, and completely random sequence where tokens are uniformly sampled from the vocabulary. Seen in Figure 4.8, this structure exists in permuted and random inputs,

Prompt

<shot 1>

###
Article: The systems, at Kentucky Methodist Hospital, Chino Valley Medical Center and Desert Valley Hospital, California, are now running normally again.
None of the hospitals is believed to have paid the ransom.
And the cases are now being investigated by the FBI.
The Kentucky Methodist Hospital had to shut down all of its desktop computers and activate a back-up system.
A message on its homepage said: "Methodist Hospital is currently working in an internal state of emergency due to a computer virus that has limited our use of electronic web-based services.
"We are currently working to resolve this issue, until then we will have limited access to web-based services and electronic communications."
It later said no patient data or care had been affected.
Fred Ortega, a spokesman for Prime Healthcare Services, which owns Chino Valley Medical Center and Desert Valley Hospital, said: "It did cause significant disruptions of our IT systems.
"However, most of the systems and the critical infrastructure has been brought back online."
The attack comes weeks after it was revealed Hollywood Presbyterian Medical Centre in Los Angeles had been attacked by ransomware.
In that case, it paid $17,000 to get access to files back.
Kentucky Methodist Hospital information systems director Jamie Reid named the malware involved as Locky, a new bug that encrypts files, documents and images and renames them with the extension .locky.
The most common way Locky gets itself on machines is via a spam email with an attached document that looks like nonsense and advises readers to enable macros "if the data encoding is incorrect".
Once the malware is downloaded, it sends a message to desktops with instructions about how users can pay to have files unlocked.
In November, a report from Intel's McAfee labs said the number of ransomware attacks was expected to grow in 2016.
Security expert Brian Krebs said: "It's a fair bet that as ransomware attacks and attackers mature, these schemes will slowly become more targeted.
"I also worry that these more deliberate attackers will take a bit more time to discern how much the data they've encrypted is really worth, and precisely how much the victim might be willing to pay to get it back."

Summarize the above article in 1 sentences.

Target

The IT systems of three US hospitals have been infected with ransomware, which encrypts vital files and demands money to unlock them.

Llama 2 7B: Full

Hospitals in California and Kentucky have been hit by a ransomware attack.

Llama 2 7B : Adaptive Wanda (50%)

Three hospitals in the US have been hit by ransomware.

Llama 2 7B: GRIFFIN (50%)

Three US hospitals were hit by ransomware, which has affected their computer systems.

Gemma 7B: Full

A computer virus has hit three hospitals in the US, causing them to shut down their systems and activate back-up servers.

Gemma 7B: Adaptive Wanda (50%)

Three hospitals in the US have been hit by ransomware attacks.

Gemma 7B: GRIFFIN (50%)

Hospitals in the US have been hit by a ransomware attack, which has encrypted their files and demanded a ransom.

Figure 4.6: Generation examples by Llama 2 7B and Gemma 7B, given a prompt (BBC, 2016) from XSum (1-shot).

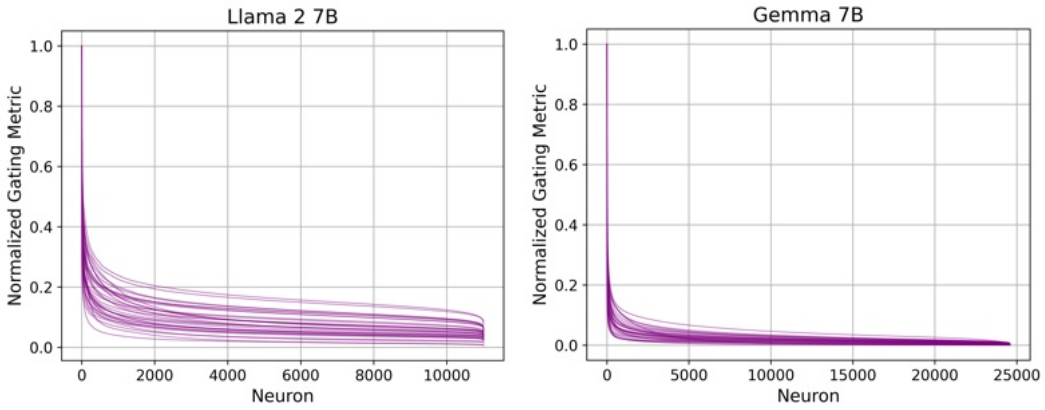

Figure 4.7: Sorted entries of $\boldsymbol{s}$ for each layer of Llama 2 7B (left) and Gemma 7B (right) with a sequence from PG-19 as the input. Each line is a layer.

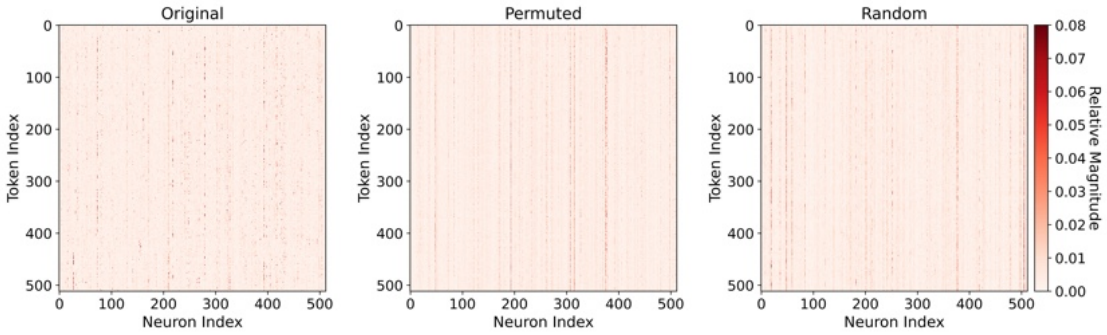

Figure 4.8: First 512 tokens and their relative FF activation magnitudes in layer 18 of Gemma 7B when inputting the original WikiText sequence (left), permuted sequence (center), and random tokens (right). Best viewed zoomed in.

perhaps even more consistently than in unperturbed sequences. This suggests something within language actually diversifies the activations, the cause of which would be of interest for future work.

4.6.3 Flocking Examples

We provide more example of flocking across different layers of the LLM. Figure 4.9 and Figure 4.10 show flocking in Gemma 7B. Figure 4.11 and Figure 4.12 show flocking in Llama 2 7B. Flocking in Gemma 7B is more visually distinct while activations in Llama 2 7B are more distributed.

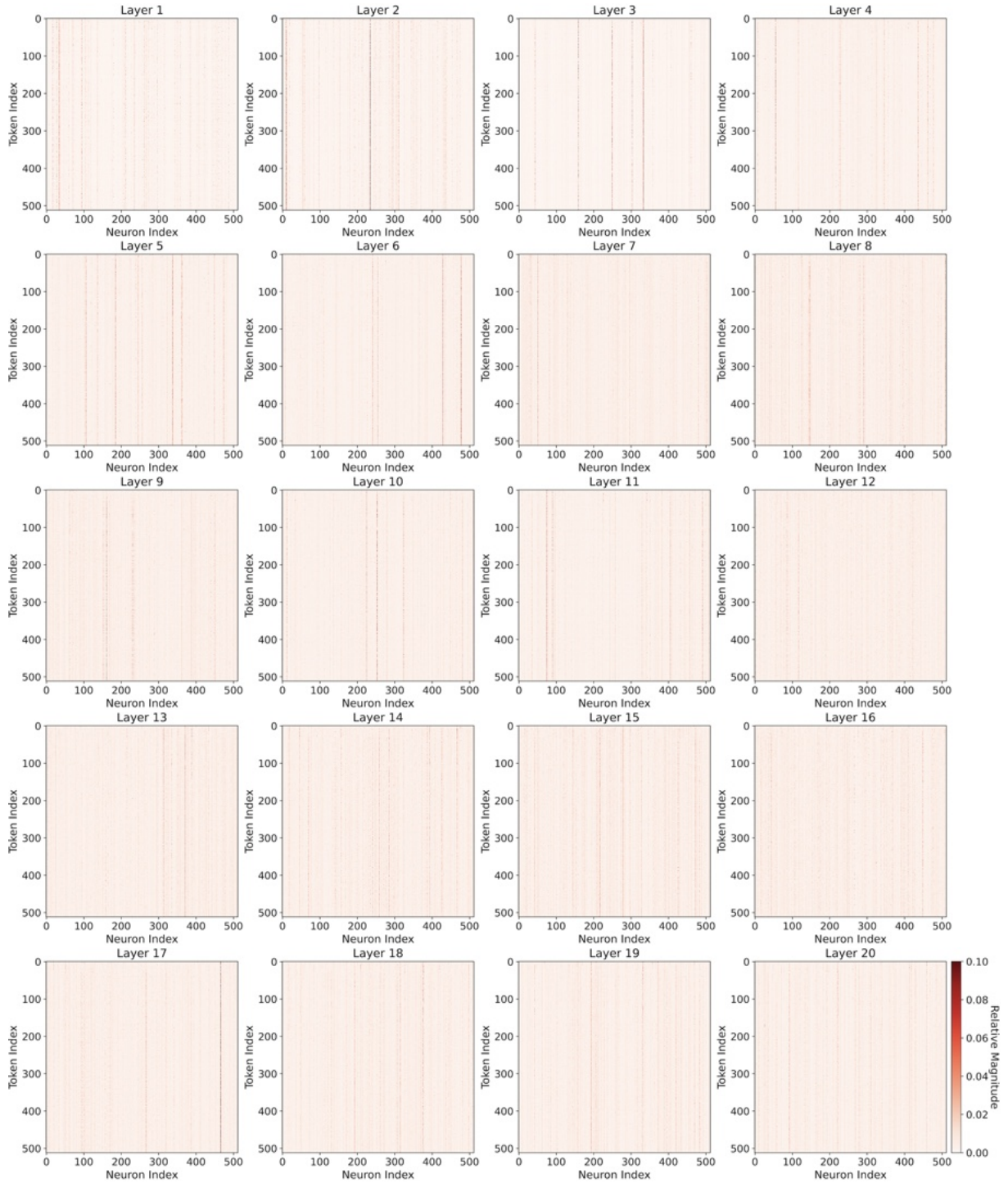

Figure 4.9: First 512 tokens and their relative FF activation magnitudes in layers 1 to 20 of Gemma 7B when inputting a sequence from PG-19.

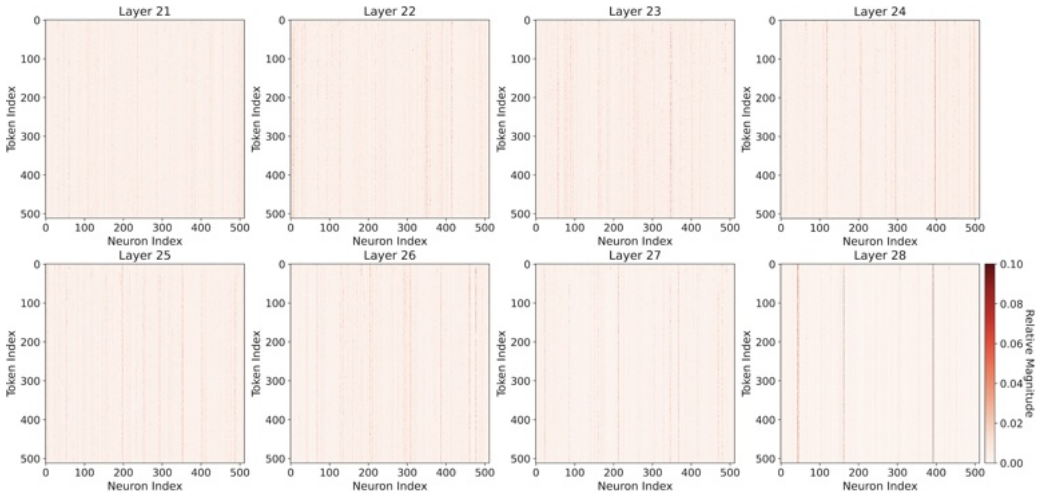

Figure 4.10: Continuation of Figure 4.9 for layers 21 to 28 of Gemma 7B.

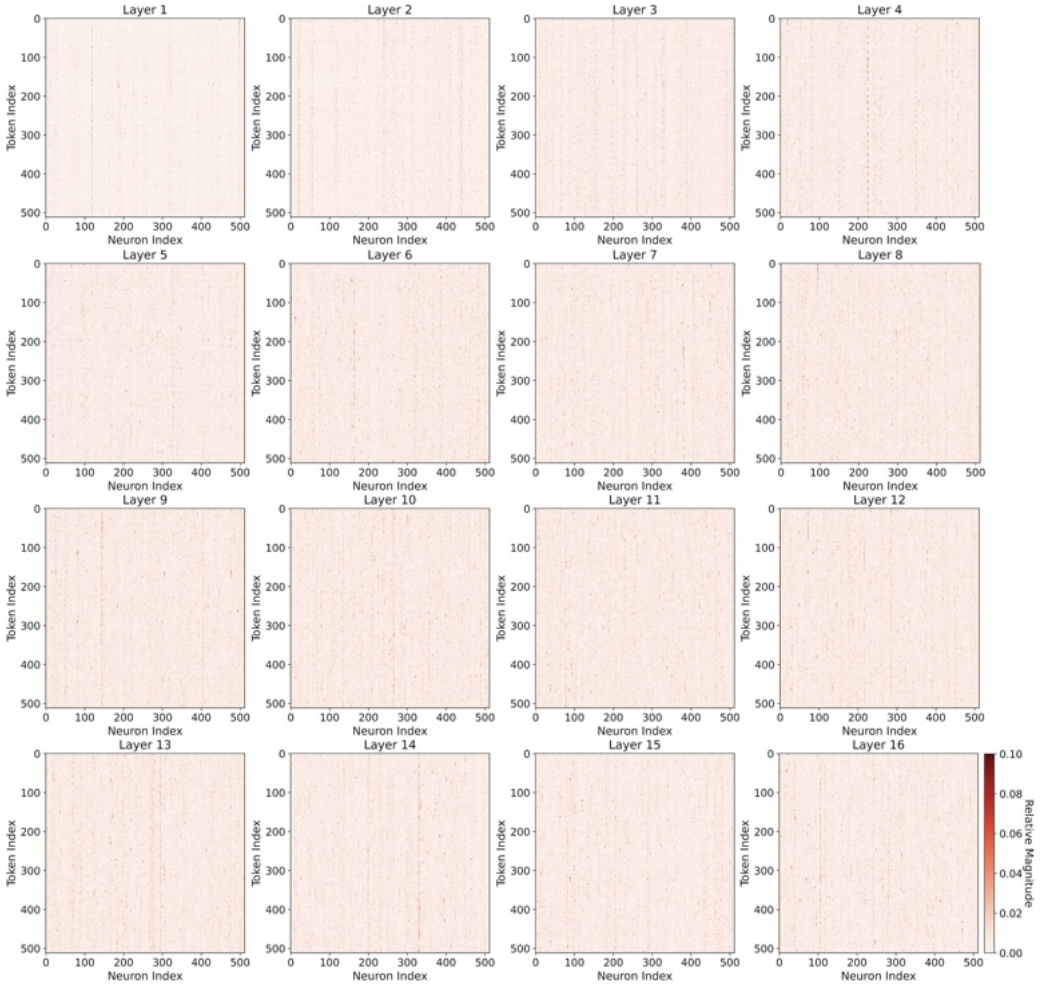

Figure 4.11: First 512 tokens and their relative FF activation magnitudes in layers 1 to 16 of Llama 2 7B when inputting a sequence from PG-19.

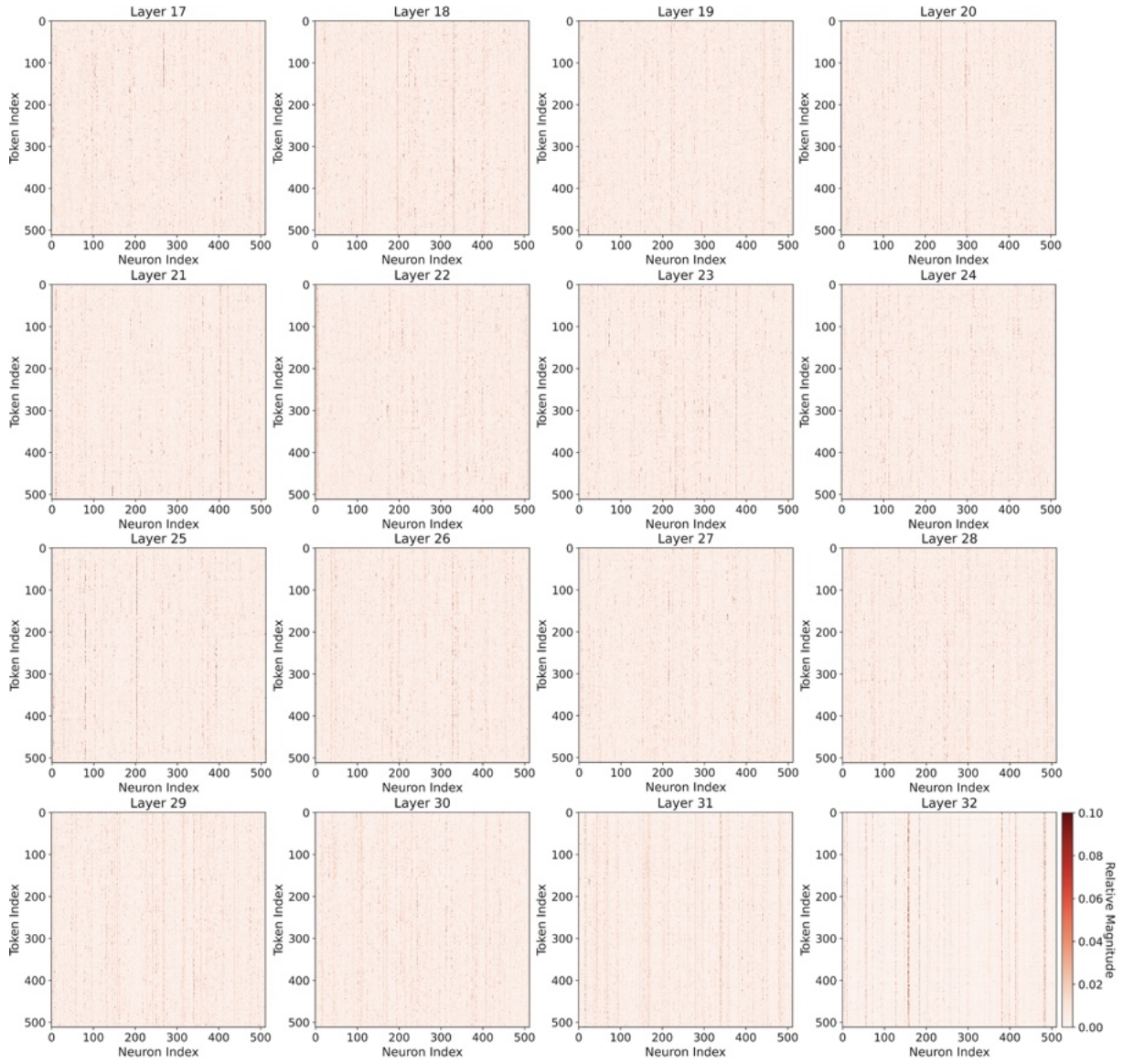

Figure 4.12: First 512 tokens and their relative FF activation magnitudes in layers 17 to 32 of Llama 2 7B when inputting a sequence from PG-19.

4.7 Chapter Summary

In this chapter, we have shown a special form of sparsity in FF layers and a simple method to exploit it. Flocking is a curious phenomenon present in many LLMs where tokens within a sequence activate neurons at similar intensities. This structure motivated the design of `GRIFFIN`, a learning-free adaptive structured pruning mechanism to remove FF neurons during inference at the sequence level which preserves the full model's performance on a large collection of classification and generative tasks at 50% FF sparsity while achieving lower latency. Furthermore, its applicability extends beyond just ReLU-based LLMs. With its straightforward algorithm and no-cost deployment, `GRIFFIN` expands the accessibility of numerous LLMs for generative inference. In the following chapter, we will build upon `GRIFFIN` and other efficient FF methods for reasoning.

Chapter 5

Extending Efficient Algorithms to Reasoning

A more recent focus on instilling reasoning capabilities into LLMs pushes the limits of existing inference methods to the extreme due to the large amount of computation required for these tasks. For the next two chapters, we will pivot our attention from usual language tasks (e.g., trivia, reading comprehension, and language modeling) to reasoning tasks (e.g., math, science, and logic). Here, new challenges arise, and old assumptions break, yet the lessons from before will still inspire the designs of our next methods. This chapter revolves around work presented in Dong et al. (2026a).

5.1 Reasoning (and Inference Scaling) as a Stress Test

With the increasing capabilities of LLMs, outputs are also becoming increasingly sophisticated, typically involving multi-step reasoning such as in math problem solving. These tasks tend to demand long generation which drives up latency, making efficiency a dire issue. Fortunately, many sparsity-based efficient LLM inference algorithms have shown great promise, slashing expensive computational bottlenecks with little damage to the original performance on a variety of language-based tasks like reading comprehension and summarization. *However, for reasoning tasks, many of these algorithms begin to break down,*

decimating performance, despite their robustness in language settings. Thus, there is a need to design efficient inference algorithms that simultaneously maintain language and reasoning capabilities.

One of the main differences between reasoning and many language tasks is the generation length. Reasoning typically involves long generations from chain-of-thoughts (CoTs) (Wei et al., 2022), which often far surpass the length of the input query. However, CoT performance falls apart with efficient algorithms that introduce approximation errors which build up over time. For instance, deploying CATS (Lee et al., 2024), a sparse thresholding method, on Gemma 2 2B (Team et al., 2024c) has little impact on language generation performance, but knocks GSM8K (Cobbe et al., 2021) accuracy *from 51.02% to 34.42%* (Table 5.5). Similarly, for reasoning models: applying `GRIFFIN` (Dong et al., 2024a), an adaptive structured pruning method, on the first half of DeepSeek-R1-Distill-Qwen 1.5B (Guo et al., 2025) drops MATH-500 (Lightman et al., 2023) accuracy *from 79.40% to 42.00%* (Table 5.2). Moreover, since generation is much more computationally demanding than prefill, there is a dire need to make long reasoning CoTs efficient, which is made more apparent with the rise of test-time scaling. Thus, this poses two key inference challenges with math reasoning. First, even single token mistakes can sometimes drive the generation trajectory off course, leading to an incorrect response (Zhou et al., 2024). Second, the already computationally expensive LLM autoregressive decoding process is accentuated with generating CoTs. This problem is further exaggerated as CoTs scale in length in reasoning models (Guo et al., 2025) and as the number of repeated generations scale to elicit higher quality answers (Brown et al., 2024; Snell et al., 2024; Wu et al., 2024b). *An ideal method should be efficient, performance preserving, and easy to integrate.*

Thankfully, low-rank structure in the feedforward (FF) output features can help make this possible. (We choose to focus on FF blocks since they contribute around 2/3 of an LLM's parameters and about 50% of the generation latency.) Because of the success of works that exploit contextual sparsity in FF blocks on language tasks for efficiency, like CATS and `GRIFFIN`, we peek into the residuals of FF sparsity-based efficient methods. Using an oracle top-k filter on FF nonlinearity output magnitudes, we observe huge reduc-

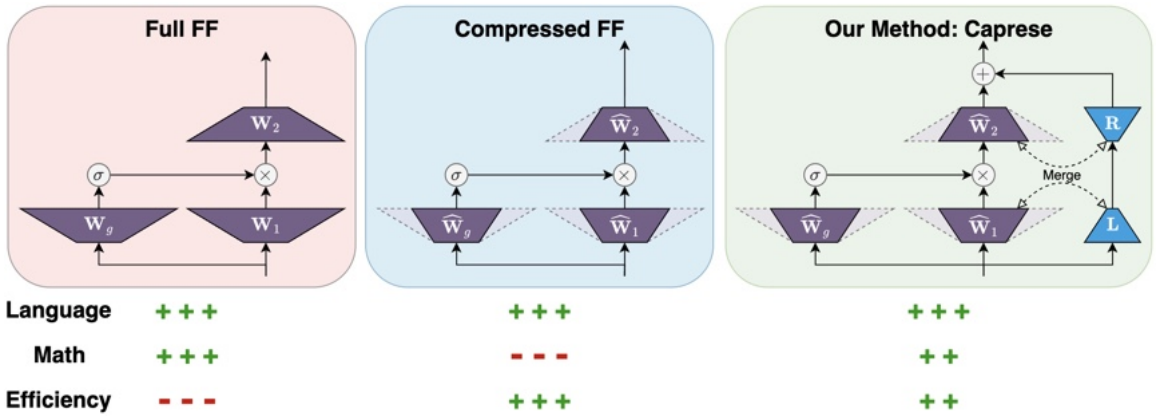

Figure 5.1: A full FF block maximizes accuracy without any benefit to efficiency. Compressed FF algorithms can be very efficient by using subsets of the FF block but harm math performance. Our method, `Caprese`, uses a compressed FF algorithm and a small distilled low-rank linear layer, which can be merged with existing FF weights, for performative inference in language and math settings while being efficient. Layers are drawn as trapezoids to highlight the expansion of the intermediate feature size compared to the input size.

tions in error with a low-rank approximation to the FF output residuals (Figure 5.2). Since FF intermediate feature sizes can be on the order of 10^5, adding 256 for a low-rank approximation is comparatively tiny. *This observation motivates us to estimate the residual from FF compression methods with low-rank layers.*

We introduce `Caprese` (**CAP**ability **RE**covery with **S**calable **E**fficiency) to learn FF residuals from a compressed FF algorithm using small low-rank linear layers for improved performance (Figure 5.1) (Dong et al., 2026a). While `Caprese` works for any task, we focus on reasoning (e.g., math) in this paper, as current efficient compression methods can obliterate an LLM's reasoning capabilities. Through distillation of math knowledge into these low-rank layers, `Caprese` is able to overcome the aforementioned challenges of reasoning and makes significant progress towards an ideal efficient method:

1. **Performance Enhancement:** Demonstrated across a variety of instruct and reasoning models, `Caprese` *recovers much if not all of the reasoning performance lost from deploying a compressed FF algorithm without harming language tasks.* Also, `Caprese` maintains its performance benefit when applying different techniques of test-time scaling.

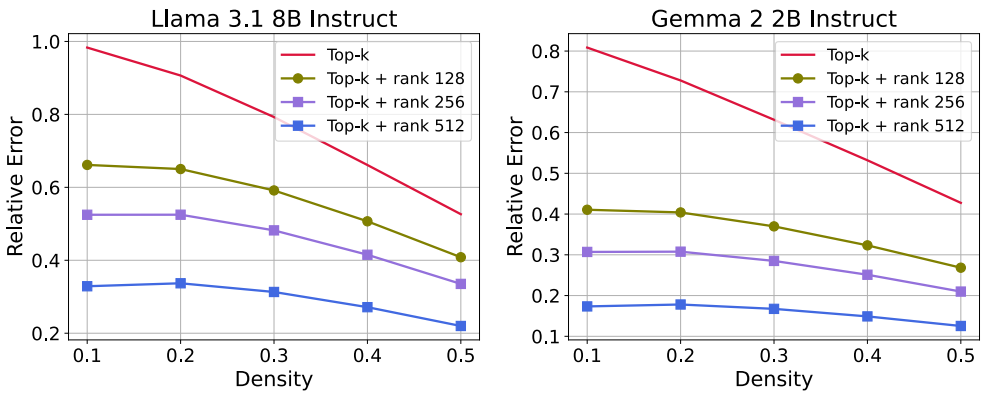

Figure 5.2: Average relative FF output error of generated tokens with varying top-k densities and low-rank approximations. The density is the fraction of non-zero intermediate FF features maintained by top-k. Samples consist of 16 random MATH generations by the original model. *A relatively small low-rank approximation to the top-k residual reduces error more effectively than increasing density.*

2. **Efficiency:** Due to its parallelizability with existing FF layers, `Caprese` adds negligible overhead, preserving latency reductions of the underlying method.
3. **Low Budget:** The distillation process of `Caprese` is cheap since we are able to see significant gains in performance by training on only 20K synthetic math samples. Moreover, with a low-rank layer size of only 256, this equates to adding roughly 0.8% additional parameters into Llama 3.1 8B and Gemma 2 9B, dwarfed by savings in active parameters (∼2B cut for both models with `Caprese` (CATS)).

Our extensive experiments on `Caprese` indicate strong performance on various tasks and models. For example, DeepSeek-R1-Distill-Qwen 7B with a sparse method drops AMC 2023 accuracy from 75.00% to 62.50%, but `Caprese` is able to bring it to 78.13%, beyond the original accuracy. Furthermore, scaling generation quantity with `Caprese` on Llama 3.2 3B Instruct improves Pass@100 by 7.0% from the original model while using 15.8% fewer active parameters. We also show `Caprese` reduces latency by 16% using the Qwen 2.5 14B architecture.

Section 5.2 covers related works and describes `GRIFFIN` and CATS, two efficient sparse FF compression algorithms that will serve as baselines. Section 5.3 details `Caprese`'s architecture and distillation procedure. Then, we showcase the strong performance of

`Caprese` in Section 5.4: scaling generation length with reasoning LLMs (Section 5.4.1), scaling generation outputs (Section 5.4.2), and instruct LLM inference (Section 5.4.3). We quantify efficiency improvements in Section 5.4.4. Finally, we explore some properties of `Caprese` in Section 5.5.

5.2 Revisiting Adaptive FF Methods

We briefly describe the inner workings of `GRIFFIN` and CATS. (`GRIFFIN` is described in more detail in Chapter 4.) Letting $\boldsymbol{X} \in \mathbb{R}^{S \times D}$ be the input into the FF block during the prefill phase with sequence length S and feature size D, recall that $\text{FF}(\boldsymbol{X}) = \text{FF}_2(\text{FF}_1(\boldsymbol{X}))$ such that

$$\boldsymbol{Z} = \text{FF}_1(\boldsymbol{X}) = \sigma(\boldsymbol{X}\boldsymbol{W}_g) \odot \boldsymbol{X}\boldsymbol{W}_1, \tag{5.1}$$

$$\text{FF}_2(\boldsymbol{Z}) = \boldsymbol{Z}\boldsymbol{W}_2, \tag{5.2}$$

where $\boldsymbol{W}_g, \boldsymbol{W}_1 \in \mathbb{R}^{D \times D_{\text{FF}}}$, $\boldsymbol{W}_2 \in \mathbb{R}^{D_{\text{FF}} \times D}$, σ is a nonlinear function, $\odot$ is an element-wise multiplication operator, and D_{FF} is the FF intermediate feature size. Typically, $D_{\text{FF}} \gg D$. Although recent LLMs use this architecture, for LLMs without gated functions (e.g., OPT (Zhang et al., 2022)), $\boldsymbol{X}\boldsymbol{W}_1$ can be removed. Bias terms are omitted for brevity.

`GRIFFIN` Overview

`GRIFFIN` (Dong et al., 2024a) adaptively prunes columns and rows in the FF block weights, using FF activation statistics from the prefill phase, namely the flocking patterns. `GRIFFIN` calculates $[\overline{\boldsymbol{Z}}]_i = |[\boldsymbol{Z}]_i| / \|[\boldsymbol{Z}]_i\|_2$ for each token index i, followed by an aggregation across the FF feature axis: $[\boldsymbol{s}]_j = \|[\overline{\boldsymbol{Z}}]_{\cdot,j}\|_2$. The result $\boldsymbol{s} \in \mathbb{R}^{D_{\text{FF}}}$ gives a metric to perform top-k selection across corresponding columns of $\boldsymbol{W}_g$ and $\boldsymbol{W}_1$, and rows of $\boldsymbol{W}_2$ to produce $\widehat{\boldsymbol{W}}_g, \widehat{\boldsymbol{W}}_1 \in \mathbb{R}^{D \times k}$, and $\widehat{\boldsymbol{W}}_2 \in \mathbb{R}^{k \times D}$. Then, for the generation phase, the following FF block is used for input $\boldsymbol{x} \in \mathbb{R}^D$:

$$\widehat{\boldsymbol{z}} = \widehat{\text{FF}}_1(\boldsymbol{x}) = \sigma(\boldsymbol{x}\widehat{\boldsymbol{W}}_g) \odot \boldsymbol{x}\widehat{\boldsymbol{W}}_1, \tag{5.3}$$

$$\widehat{\mathrm{FF}}_2(\widehat{\boldsymbol{z}}) = \widehat{\boldsymbol{z}}\widehat{\boldsymbol{W}}_2. \tag{5.4}$$

Recall that the compressed FF blocks are fixed throughout generation but are dynamic across prompts.

CATS Overview

CATS (Lee et al., 2024) uses hard thresholding to skip computation in part of the FF block. Letting T_τ be the hard thresholding function with threshold τ, CATS computes

$$\widehat{\boldsymbol{z}} = \widehat{\mathrm{FF}}_1(\boldsymbol{x}) = T_\tau(\sigma(\boldsymbol{x}\boldsymbol{W}_g)) \odot \boldsymbol{x}\boldsymbol{W}_1, \tag{5.5}$$

$$\widehat{\mathrm{FF}}_2(\widehat{\boldsymbol{z}}) = \widehat{\boldsymbol{z}}\boldsymbol{W}_2. \tag{5.6}$$

The weights $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ should be sparsified into $\widehat{\boldsymbol{W}}_1$ and $\widehat{\boldsymbol{W}}_2$, respectively, based on the non-zero entries of $T_\tau(\sigma(\boldsymbol{x}\boldsymbol{W}_g))$ for latency improvement, which can vary from token to token and require a custom kernel for wall clock speed-up. The parameter τ is calibrated to be a desired percentile on a dataset. In this paper, to avoid calibration, we set τ based on prefill features and only threshold during the generation phase, analogous to `GRIFFIN`.

5.3 `Caprese`: Low-rank Recovery of FF Algorithms

Motivated by the low-rank structure of residuals in Figure 5.2, we introduce `Caprese` which distills approximation errors in embeddings into low-rank linear layers in FF blocks. See Figure 5.1 for an illustration of our method.

5.3.1 Layer-wise Distillation

Inference efficiency algorithms often introduce feature approximation errors in favor of faster generation, which we mitigate with distillation. Let the efficient and approximate FF block be $\widehat{\mathrm{FF}}(\boldsymbol{x}) = \widehat{\mathrm{FF}}_2(\widehat{\mathrm{FF}}_1(\boldsymbol{x}))$. In our design of `Caprese`, we do not constrain $\widehat{\mathrm{FF}}$ to be any specific method or architecture. For instance, $\widehat{\mathrm{FF}}$ could be an FF block with

GRIFFIN (Dong et al., 2024a), CATS (Lee et al., 2024), or quantized weights. Then, the error is

$$\|\mathrm{FF}(\boldsymbol{x}) - \widehat{\mathrm{FF}}(\boldsymbol{x})\|_2^2.$$

We choose to reduce this residual with a low-rank linear layer, meaning we want to solve

$$\min_{\boldsymbol{L},\boldsymbol{R}} \frac{1}{|\mathcal{X}|} \sum_{\boldsymbol{x}\in\mathcal{X}} \|\mathrm{FF}(\boldsymbol{x}) - \widehat{\mathrm{FF}}(\boldsymbol{x}) - \boldsymbol{x}\boldsymbol{L}\boldsymbol{R}\|_2^2 \tag{5.7}$$

for input set $\mathcal{X}$, $\boldsymbol{L} \in \mathbb{R}^{D\times r}$, and $\boldsymbol{R} \in \mathbb{R}^{r\times D}$ where $r \ll D_{\mathrm{FF}}$. The optimal solution can be computed analytically since this is a reduced rank regression problem, but the size of $|\mathcal{X}|$ and D may make it prohibitively expensive. Therefore, we opt to learn $\boldsymbol{L}$ and $\boldsymbol{R}$ independently for every FF block (i.e., previous FF blocks are assumed to be from the original model), allowing for parallel layer-wise training. This takes inspiration from LESS (Dong et al., 2024c) which uses layer-wise training on attention residuals for key-value cache compression. Each $\boldsymbol{R}$ is initialized as a zero matrix since the efficient approximation is assumed to be of good quality, and original model weights are frozen. We also distill end-to-end (E2E) to further improve the performance.

5.3.2 End-to-end Distillation

Using the learned low-rank layers as an initialization, we put them all together to distill the final model embedding before the linear head into the efficient model, again using MSE:

$$\min_{(\boldsymbol{L}_i,\boldsymbol{R}_i)_{i=1,\ldots,L}} \frac{1}{|\mathcal{X}|} \sum_{\boldsymbol{x}\in\mathcal{X}} \|\mathrm{M}(\boldsymbol{x}) - \mathrm{M}_{\mathrm{student}}(\boldsymbol{x})\|_2^2 \tag{5.8}$$

where M and $\mathrm{M}_{\mathrm{student}}$ are the original LLM and efficient LLM with distillation layers, respectively, excluding the final linear head. L is the number of layers in the model. All original weights are frozen, so the only tunable parameters are the $\boldsymbol{L}$ and $\boldsymbol{R}$ of each FF block.

5.3.3 Parallel Inference Computation

The computation of $\boldsymbol{xLR}$ can be done in parallel with the original FF operations. In fact, $\boldsymbol{L}$ and $\boldsymbol{R}$ can be concatenated with the up and down projection matrices, respectively. In other words, the `Caprese` FF block is $\widehat{\mathrm{FF}}^+(\boldsymbol{x}) = \widehat{\mathrm{FF}}_2^+(\widehat{\mathrm{FF}}_1^+(\boldsymbol{x}))$, such that

$$\widehat{\boldsymbol{z}}^+ = \widehat{\mathrm{FF}}_1^+(\boldsymbol{x}) = \begin{bmatrix}\sigma(\boldsymbol{x}\widehat{\boldsymbol{W}}_g) & \mathbf{1}_r\end{bmatrix} \odot \boldsymbol{x} \begin{bmatrix}\widehat{\boldsymbol{W}}_1 & \boldsymbol{L}\end{bmatrix}, \tag{5.9}$$

$$\widehat{\mathrm{FF}}_2^+(\widehat{\boldsymbol{z}}^+) = \widehat{\boldsymbol{z}}^+ \begin{bmatrix}\widehat{\boldsymbol{W}}_2^\top & \boldsymbol{R}^\top\end{bmatrix}^\top, \tag{5.10}$$

where $\mathbf{1}_r$ is a one-vector with length r. In practice, to save memory and time, we do not materialize $\mathbf{1}_r$ but directly assign the product with $\sigma(\boldsymbol{x}\widehat{\boldsymbol{W}}_g)$ to corresponding entries of $\boldsymbol{x}\widehat{\boldsymbol{W}}_1^+$. Recall that the prefill stage still just uses the original model.

5.3.4 Training Details

We set the inner dimension of our low-rank layer to $r = 256$ (to see the effect of different r, see Appendix 5.5.2). In comparison to the large inner dimension of FF layers (e.g., $D_{\mathrm{FF}} = 14336$ for Llama 3.1 8B and Gemma 2 9B), our choice of r is relatively minuscule, adding only $\sim 1\%$ new parameters for all tested models (Figure 5.3). We use a 20K subset of a common synthetic math training set for training. For a fair comparison, the same subset is used for both layer-wise and E2E distillation for every model. Each training sample is prepended with a CoT instruction: "Please reason step by step." At test time, the actual instructions may be vastly different. Layer-wise and E2E training consists of 20 epochs and 3 epochs, respectively. Training and inference are done in BF16. Table 5.1 lists the hyperparameter settings for training `Caprese` layers. The E2E learning rates lie in the interval [4e-6, 2e-4], where larger models tend to learn better with smaller learning rates.

Table 5.1: `Caprese` layer-wise and E2E training hyperparameters.

	Layer-wise	End-to-end
Optimizer	Adam	Adam
Learning rate	1e-3	[4e-6, 2e-4] (varies)
Batch size	128	16
Epochs	20	3
Training samples	2e5	2e5
Scheduler	Linear	Linear
Warmup	2%	2%

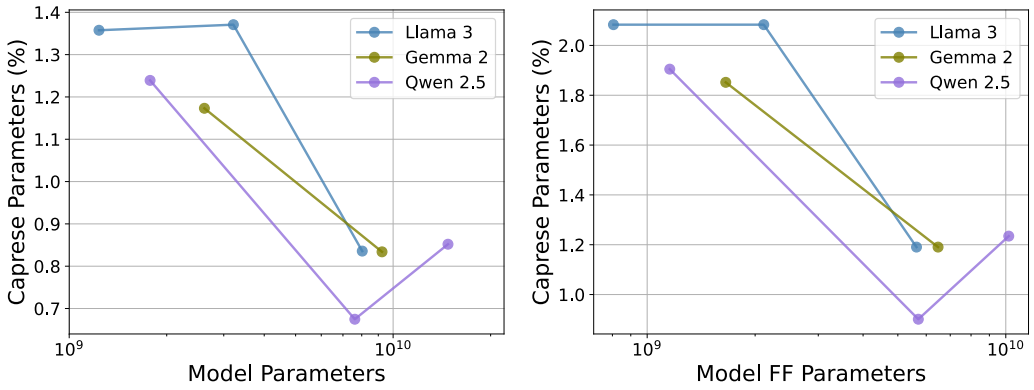

Figure 5.3: The percent of new parameters that `Caprese` ($r = 256$) adds relative to the entire model (left) and relative to only the FF parameters (right) for the Llama 3, Gemma 2, and Qwen 2.5 model families.

5.3.5 Comparison with LoRA

Our method derives inspiration from and shares a slight connection with low-rank adaptation (LoRA), a widely used method for efficient fine tuning with the addition of low-rank parameters (Hu et al., 2022), but remains distinct since they target different inefficiencies. To illustrate, recall a sparse algorithm constructs $\widehat{\boldsymbol{W}}_1 \in \mathbb{R}^{D \times k}$ by selecting columns from $\boldsymbol{W}_1$. Alternatively, construct $\widetilde{\boldsymbol{W}}_1 \in \mathbb{R}^{D \times D_{\text{FF}}}$ by setting unwanted columns to zero, so nonzero columns in $\widetilde{\boldsymbol{W}}_1$ match columns in $\widehat{\boldsymbol{W}}_1$ and vice versa. Doing the same to columns of $\boldsymbol{W}_g$ and rows of $\boldsymbol{W}_2$, LoRA learns $\boldsymbol{A}_1, \boldsymbol{A}_g, \boldsymbol{B}_2^\top \in \mathbb{R}^{D \times r}$ and $\boldsymbol{B}_1, \boldsymbol{B}_g, \boldsymbol{A}_2^\top \in \mathbb{R}^{r \times D_{\text{FF}}}$ to

form a new FF block with LoRA weights:

$$\widetilde{\mathrm{FF}}(\boldsymbol{x}) = \Big(\sigma\big(\boldsymbol{x}(\widetilde{\boldsymbol{W}}_g + \boldsymbol{A}_g\boldsymbol{B}_g)\big) \odot \big(\boldsymbol{x}(\widetilde{\boldsymbol{W}}_1 + \boldsymbol{A}_1\boldsymbol{B}_1)\big)\Big)\,(\widetilde{\boldsymbol{W}}_2 + \boldsymbol{A}_2\boldsymbol{B}_2). \tag{5.11}$$

From this, we see *LoRA is not designed to be applied to sparse efficient inference algorithms.* First, this is because in its compressed form, LoRA requires two sequential operations per linear layer. In contrast, `Caprese` parameters can be appended to existing ones, so additional computation is all parallel to the original operations (Section 5.3.3). The real latency impact of this difference is quantified later in Table 5.8. `Caprese`'s architecture also allows efficient training, but that is merely a byproduct of our design, not its purpose. Second, for a method like GRIFFIN which reduces the intermediate FF feature size from D_{FF} to k, adding LoRA will negate this benefit while `Caprese` will preserve the reduced feature size. Due to this difference, we de-prioritize comparisons with LoRA in the main text, but include comparisons with LoRA in Section 5.4.3 for interested readers.

5.4 Core Experiments

We showcase the effectiveness of `Caprese` at recovering much, if not all, of the reasoning performance lost from efficient inference algorithms without sacrificing efficiency or performance on language tasks. Beginning with sequence length scaling in Section 5.4.1, `Caprese` is able to significantly or completely recover lost reasoning performance from existing efficient algorithms in reasoning LLMs. Next, we demonstrate `Caprese`'s benefit when scaling the number of generations, another axis of tes-time scaling, in Section 5.4.2. Then in Section 5.4.3, we investigate typical instruct LLM settings without test-time scaling where we again observe better math performance without losing their generalizability with explorations into quantization. Finally, we highlight `Caprese`'s latency and length improvements in Section 5.4.4. When ambiguous, we denote `Caprese` (CATS) to be our method with CATS as the underlying compression method and similarly for `GRIFFIN`. Otherwise, we use "`Caprese`" for brevity. Unless specified, FF intermediate feature sparsity is set at 50%. *For a more meaningful baseline, we only apply `GRIFFIN` to the first half of*

Table 5.2: Reasoning models' 0-shot accuracies on reasoning tasks. Sparsity is set at 50%, and $r = 256$. AIME 2024, AMC 2023, and BRUMO 2025 columns also include the sample standard deviation across 4 runs. Further improvements with reselection are shown in Table 5.4.

Model	MATH-500	AIME 2024	AMC 2023	BRUMO 2025	GPQA
DS-Qwen 1.5B	79.40	30.00 ± 2.72	60.83 ± 7.47	25.83 ± 3.19	18.69
GRIFFIN	42.00	2.50 ± 1.67	16.88 ± 1.25	1.67 ± 1.92	11.62
Layer-wise Caprese	47.20	2.50 ± 1.67	28.13 ± 2.39	0.00 ± 0.00	13.13
E2E Caprese	**60.40**	**6.67** ± 2.72	**34.38** ± 6.57	**4.17** ± 1.67	**16.67**
CATS	72.00	11.67 ± 6.94	37.50 ± 3.54	10.83 ± 4.19	11.62
Layer-wise Caprese	73.80	15.83 ± 3.19	**48.13** ± 3.75	13.33 ± 2.72	16.67
E2E Caprese	**74.80**	**20.83** ± 1.67	47.50 ± 3.54	**20.00** ± 2.72	**22.22**
DS-Qwen 7B	90.20	47.50 ± 7.87	75.00 ± 4.08	45.83 ± 4.19	38.38
GRIFFIN	80.60	21.67 ± 4.30	62.50 ± 8.90	25.00 ± 8.39	21.72
Layer-wise Caprese	84.80	29.17 ± 3.19	64.38 ± 7.18	27.50 ± 4.19	**27.78**
E2E Caprese	**85.40**	**30.00** ± 6.09	**71.88** ± 5.54	**29.17** ± 5.69	23.74
CATS	89.20	34.17 ± 7.88	62.50 ± 2.04	37.50 ± 7.39	32.83
Layer-wise Caprese	87.00	**35.00** ± 4.30	70.00 ± 10.21	39.17 ± 3.19	**38.38**
E2E Caprese	**90.00**	33.33 ± 4.71	**78.13** ± 5.15	**47.50** ± 5.00	35.86
DS-Qwen 14B	92.80	63.33 ± 2.72	90.63 ± 2.39	60.83 ± 4.19	52.53
GRIFFIN	89.80	32.50 ± 7.39	80.63 ± 3.75	33.33 ± 3.85	41.92
Layer-wise Caprese	**90.80**	**41.67** ± 5.77	**86.88** ± 4.27	**43.33** ± 4.71	**55.05**
E2E Caprese	89.20	39.17 ± 9.18	84.38 ± 6.57	40.00 ± 2.72	43.43
CATS	**92.80**	58.34 ± 1.92	**88.75** ± 1.44	**55.00** ± 3.33	44.95
Layer-wise Caprese	91.00	**61.67** ± 5.77	**88.75** ± 1.44	53.00 ± 7.20	50.51
E2E Caprese	92.00	58.34 ± 9.62	87.50 ± 2.04	49.17 ± 4.19	**51.01**

models as we observe steeper drops in math performance if used for all layers. CATS is applied to all layers.

5.4.1 Scaling Length: Reasoning Models

Reasoning models scale inference by augmenting the CoT length before giving a final answer. Since this entails long generation lengths, it is critical that error accumulation across tokens be minimized. We test on DeepSeek-R1-Distill-Qwen (which we abbreviate to DS-Qwen) models (Guo et al., 2025; Yang et al., 2024a) using Open R1 (Face, 2025) prompt

Table 5.3: Average reasoning model performance standard deviation on AIME 2024, AIME 2023, and BRUMO 2025 from Table 5.2. There is no clear consistent relationship between `Caprese` and accuracy variance.

	DS-Qwen 1.5B	DS-Qwen 7B	DS-Qwen 14B	Mean
Full Model	4.46	5.38	3.10	4.31
`GRIFFIN`	1.61	7.20	5.00	4.60
Layer-wise `Caprese`	1.35	4.85	4.92	3.71
E2E `Caprese`	3.65	5.77	6.16	5.19
CATS	4.89	5.77	2.23	4.30
Layer-wise `Caprese`	3.22	5.90	4.80	3.97
E2E `Caprese`	2.64	4.95	5.28	4.29

templates and configurations (temperature and top-p set to 0.6 and 0.95, respectively). We test 0-shot performance on math and science tasks MATH-500 (Hendrycks et al., 2021; Lightman et al., 2023), AIME 2024 (AIME, 2025), AMC 2023 (AMC, 2023), BRUMO 2025 (BRUMO, 2025), and GPQA (Rein et al., 2024), for a max generation length of 32768. Due to the small size of AIME 2024, AMC 2023, and BRUMO 2025, we evaluate them 4 times, reporting the average accuracy and sample standard deviation. We observe none of the methods have any significant effect on standard deviation in Table 5.3.

From Table 5.2, `Caprese` is the most performative in most cases. DS-Qwen 1.5B and 7B see the greatest benefit with E2E `Caprese` usually achieving the highest accuracy, sometimes even exceeding the full model's performance. DS-Qwen 7B with CATS brings AMC 2023 accuracy down to 62.50% from 75.00%, but `Caprese` lifts performance above the original model's to 78.13%, and similarly with DS-Qwen 7B with CATS and BRUMO 2025. All methods are more robust as model size increases. AIME 2024 and BRUMO 2025 challenge CATS and `GRIFFIN` with severe degradation and partial recovery with `Caprese`. *In the next section, we show a simple way to push this recovery even further with reselection.*

Enhanced Performance with Reselection

We can push the performance of `Caprese` with neuron reselection. For a sample, `GRIFFIN` and CATS calculate metrics ($\boldsymbol{s}$ and τ) to determine subsets of the FF block to use, but

Table 5.4: E2E `Caprese` AMC 2023 accuracies and standard deviations when recalculating `GRIFFIN` pruning metrics and reselecting pruned neurons every ρ decode steps. No reselection and the full model are special cases where $\rho = \infty$ and $\rho = 0$, respectively.

Model	No Reselect	$\rho = 1024$	$\rho = 256$	$\rho = 64$	Full
DS-Qwen 1.5B	34.38	41.87	45.00	**58.75**	60.83
DS-Qwen 7B	71.88	69.25	73.13	**73.75**	75.00
DS-Qwen 14B	84.38	88.13	88.75	**91.88**	90.63

these metrics are fixed during generation. Updating them mid-generation can benefit downstream performance.

For `GRIFFIN`, updating the metric $\boldsymbol{s}$ entails integrating the FF feature statistics of generated tokens into $\boldsymbol{s}$. While this can be done by re-running prefill on all tokens, there is a more efficient way by passing in the generated tokens following the last reselection through the full model. As these tokens propagate through each layer, we find the selection metric for the generated tokens $\boldsymbol{s}_G$ and update corresponding KV pairs. This is like verification in speculative decoding (Chen et al., 2023a; Leviathan et al., 2023). As $\boldsymbol{s}$ and $\boldsymbol{s}_G$ are ℓ_2-norms along the token axis, we define the updated metric as $\sqrt{(\boldsymbol{s} \odot \boldsymbol{s}) + (\boldsymbol{s}_G \odot \boldsymbol{s}_G)}$ and use that to reselect different subsets of the FF block to use (as described in Section 5.2). *This updates the pruned layers yet avoids prefill for all tokens.* Table 5.4 shows a clear benefit of reselection (even if infrequent compared to speculative decoding) in `Caprese` by pushing the performance much closer to the full model's as we decrease steps between reselection rounds. Although `GRIFFIN` is generally more harmful to accuracy than CATS due to its structured pruning, with reselection, `Caprese` (`GRIFFIN`) can exceed the performance of `Caprese` (CATS) and reach the full model performance.

Reselection is also possible with CATS but requires recomputing prefill for all tokens. The parameter to update is the hard thresholding parameter τ, but since this requires finding the desired percentile of intermediate values in the FF block, we would need to access to all values, which likely cannot be fully stored in memory. In turn, prefill will need to be redone anytime we want to update τ. Since τ represents a cutoff for a desired

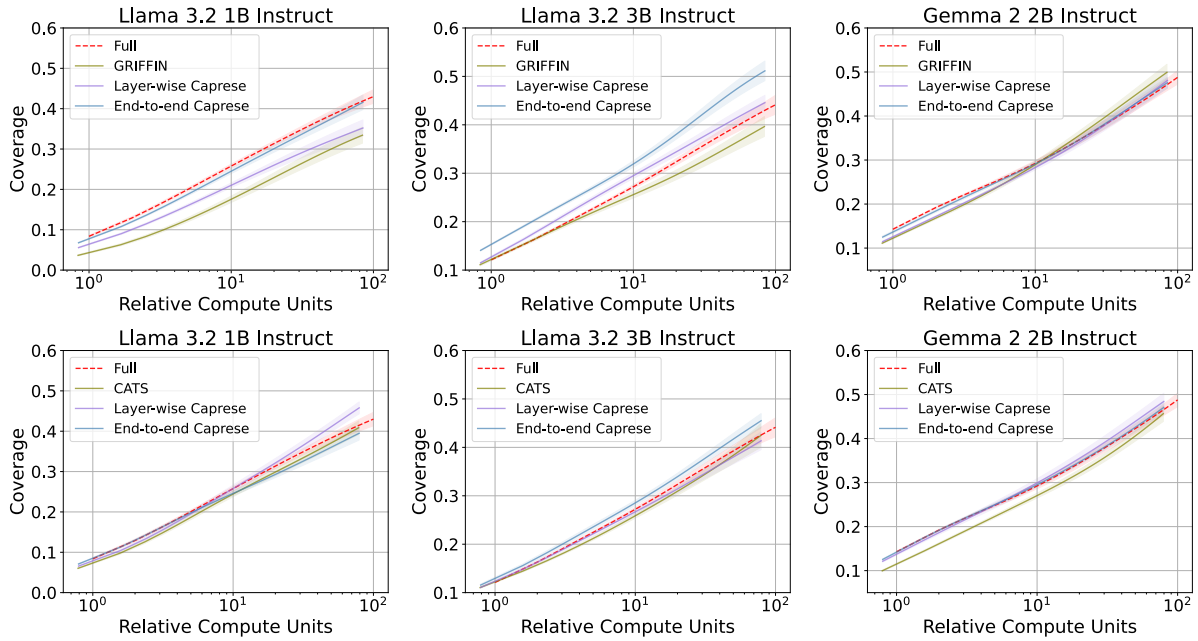

Figure 5.4: Coverage and standard deviation of 140 samples from MATH as the number of generation attempts, N, scales. We define Relative Compute Units $= N \times A$ where A is the fraction of total parameters activated per input. FF sparsity set to 50%. Best viewed zoomed.

percentile (e.g., median), it is fairly robust to new observations. Given this and the lack of computational incentive, we focus primarily on reselection for `GRIFFIN`.

5.4.2 Scaling Best-of-N: Coverage

Now, we see the generalizability of `Caprese` on the second axis of test-time scaling: sampling multiple responses and selecting the best one, known as Best-of-N. Increasing the number of generations per prompt, N, increases the probability of generating a correct answer, even for non-reasoning models (e.g., instruct models). To evaluate, we find the coverage (Pass@K) on 140 samples from MATH. We use an oracle verifier to accurately assess the quality of the pool of generated responses. To combat the high variance when K is close to N, we calculate the average coverage for $K = 1, \ldots, 100$ across 10 independent pools of 100 generations for each sample. *Factoring in the saved compute, E2E* ***`Caprese`*** *is able to have similar or better coverage scaling compared to the original model*, as shown in Figure 5.4. Notably, E2E `Caprese` (`GRIFFIN`) on Llama 3.2 3B Instruct improves Pass@100 by 7.0% from the original model while using 15.8% less compute.

Table 5.5: Instruct models' 0-shot accuracies on mathematical reasoning (GSM8K and MATH) and language generation tasks (CoQA, QASPER, XSum, and CNN/DailyMail). FF sparsity is set to 50%.

Model	GSM8K	MATH	CoQA	QASPER	XSum	CNN/DM
Llama 3.2 1B Instruct	22.44	10.66	55.43	14.43	21.65	25.60
GRIFFIN	7.13	5.42	56.05	14.11	21.13	25.47
Layer-wise Caprese	13.72	6.62	56.07	13.40	20.65	26.18
E2E Caprese	**21.00**	**8.44**	56.55	13.88	20.71	26.18
CATS	19.18	7.54	54.40	13.88	21.10	24.73
Layer-wise Caprese	18.65	8.28	55.58	14.35	20.94	25.35
E2E Caprese	**20.39**	**9.04**	56.12	13.75	20.40	25.56
Llama 3.2 3B Instruct	51.55	14.32	63.95	12.45	23.22	26.20
GRIFFIN	28.96	10.98	64.52	12.52	22.09	25.49
Layer-wise Caprese	40.18	13.70	64.33	11.60	21.56	25.90
E2E Caprese	**44.66**	**16.96**	64.83	12.35	21.26	26.04
CATS	41.24	12.04	58.87	11.36	22.23	25.40
Layer-wise Caprese	45.49	13.62	60.53	12.26	22.50	25.90
E2E Caprese	**46.85**	**14.40**	61.72	12.36	21.58	25.40
Llama 3.1 8B Instruct	53.98	13.94	63.88	15.16	21.97	25.98
GRIFFIN	20.47	6.16	63.37	12.63	21.53	25.89
Layer-wise Caprese	37.60	9.72	65.05	14.21	21.92	26.31
E2E Caprese	**51.40**	**12.64**	65.50	15.35	22.05	26.42
CATS	50.95	11.80	58.85	12.26	21.43	25.67
Layer-wise Caprese	51.93	13.66	63.92	14.34	22.58	25.79
E2E Caprese	**58.00**	**13.88**	64.27	14.42	22.08	26.13
Gemma 2 2B Instruct	51.02	16.06	63.77	10.96	22.17	26.01
GRIFFIN	33.74	11.32	63.28	11.07	18.27	22.24
Layer-wise Caprese	42.53	12.32	63.77	10.75	21.63	26.37
E2E Caprese	**48.14**	**13.70**	63.37	11.05	22.48	27.16
CATS	34.42	10.56	61.53	10.11	21.97	26.46
Layer-wise Caprese	**46.32**	13.90	61.92	10.82	22.10	26.23
E2E Caprese	46.17	**14.16**	63.92	11.03	22.15	26.52
Gemma 2 9B Instruct	78.17	27.64	63.78	9.91	23.98	26.46
GRIFFIN	59.21	25.22	63.82	10.14	24.66	26.77
Layer-wise Caprese	**76.72**	**25.84**	64.20	9.82	24.88	26.88
E2E Caprese	76.65	25.30	64.42	9.92	24.89	25.47
CATS	76.50	27.32	64.37	9.78	24.74	26.64
Layer-wise Caprese	**77.18**	**28.16**	64.52	10.46	24.54	26.45
E2E Caprese	**77.18**	28.00	64.87	10.02	24.65	26.33

Table 5.6: 0-shot accuracies on mathematical reasoning (GSM8K and MATH) and language generation tasks (CoQA, QASPER, XSum, and CNN/DailyMail) with LoRA. FF sparsity is set to 50%, $r = 256$, and LoRA ranks are set to match the number of parameters that `Caprese` adds.

Model	GSM8K	MATH	CoQA	QASPER	XSum	CNN/DM
Llama 3.2 1B Instruct	22.44	10.66	55.43	14.43	21.65	25.60
`GRIFFIN`	7.13	5.42	56.05	14.11	21.13	25.47
Layer-wise LoRA	8.11	5.60	56.00	14.65	21.08	25.51
E2E LoRA	20.47	**10.10**	56.10	14.41	21.09	25.62
Layer-wise `Caprese`	13.72	6.62	56.07	13.40	20.65	26.18
E2E `Caprese`	**21.00**	8.44	56.55	13.88	20.71	26.18
Llama 3.2 3B Instruct	51.55	14.32	63.95	12.45	23.22	26.20
`GRIFFIN`	28.96	10.98	64.52	12.52	22.09	25.49
Layer-wise LoRA	26.46	11.28	63.88	12.55	22.12	25.96
E2E LoRA	42.91	16.50	64.67	12.63	21.81	26.05
Layer-wise `Caprese`	40.18	13.70	64.33	11.60	21.56	25.90
E2E `Caprese`	**44.66**	**16.96**	64.83	12.35	21.26	26.04
Gemma 2 2B Instruct	51.02	16.06	63.77	10.96	22.17	26.01
`GRIFFIN`	33.74	11.32	63.28	11.07	18.27	22.24
Layer-wise LoRA	39.58	14.66	62.98	10.54	21.58	26.64
E2E LoRA	44.66	**16.71**	63.03	10.81	22.45	26.35
Layer-wise `Caprese`	42.53	12.32	63.77	10.75	21.63	26.37
E2E `Caprese`	**48.14**	**13.70**	63.37	11.05	22.48	27.16

5.4.3 Instruct Models

Next, we look into the performance of `Caprese` in the absence of inference scaling methods on instruct LLMs whcih are primarily tailored for language tasks. We show `Caprese` is able to preserve math performance without sacrificing quality on language tasks like question answering. We test Llama 3 (Dubey et al., 2024) and Gemma 2 (Team et al., 2024c) models on 0-shot GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), CoQA (Reddy et al., 2019), QASPER (Dasigi et al., 2021), XSum (Narayan et al., 2018), and CNN/DailyMail (Hermann et al., 2015; See et al., 2017). We use CoT prompts for math tasks.

Table 5.5 shows ***`Caprese`*** *preserves most if not all of the math capabilities in the original*

Table 5.7: 0-shot accuracies on mathematical reasoning (GSM8K and MATH) and language generation tasks (CoQA, QASPER) with QLoRA as the backend method. FF weights are quantized to 4 bits and $r = 256$.

Model	GSM8K	MATH	CoQA	QASPER
Llama 3.2 1B Instruct	22.44	10.66	55.43	14.43
QLoRA	12.05	6.56	48.70	12.37
Layer-wise `Caprese`	15.01	6.88	49.60	12.65
E2E `Caprese`	**21.61**	**9.98**	53.52	13.54
Llama 3.2 3B Instruct	51.55	14.32	63.95	12.45
QLoRA	45.41	11.98	62.22	12.87
Layer-wise `Caprese`	45.56	11.78	64.25	12.41
E2E `Caprese`	**48.82**	**14.14**	64.05	12.26

models without damaging performance on the language tasks, despite the distillation dataset being all math. In most cases, CATS and `GRIFFIN` severely harm GSM8K and MATH accuracy, but `Caprese` effectively recovers the lost performance. E2E `Caprese` is the most performative in the majority of math scenarios. All methods have little impact on the accuracy of language tasks (the main purpose of these tasks is to show no degradation in language-related cases).

Additionally, we report the performance of using LoRA in place of `Caprese` while keeping parameter count constant in Table 5.6. `Caprese` achieves the highest accuracy in slightly more cases than LoRA, but as per our discussion on LoRA in Section 5.3.5 and efficiency analysis in Section 5.4.4, LoRA's main benefit is efficient training, not efficient inference.

To demonstrate recovery generalizability beyond sparse FF methods as the backend, we also equip QLoRA (Dettmers et al., 2023) with `Caprese`. Table 5.7 shows that E2E `Caprese` is also able to nearly recover the performance degraded by quantization in both math and language tasks.

Table 5.8: End-to-end generation latency (s) and average time to next token (ms) for Qwen 2.5 14B. For the "Setup" column, $P+G$ indicates input and generation lengths of P and G tokens, respectively. As before, `GRIFFIN` is applied to the first half of the model, sparsity is 50%, and $r = 256$.

Setup	End-to-end Latency (s)				Avg. Time to Next Token (ms)			
	Full	`GRIFFIN`	LoRA	`Caprese`	Full	`GRIFFIN`	LoRA	`Caprese`
2048+256	10.5	8.7	9.5	8.7	41	34	37	34
2048+2048	84.4	70.3	76.5	70.4	41	34	37	34
2048+8192	344.9	287.7	312.5	288.8	42	35	38	35

5.4.4 Efficiency

`Caprese` reduces generation latency (Table 5.8). `Caprese` cuts total latency and time to next token by >16% for the Qwen 2.5 14B architecture. Moreover, the latency differences between `GRIFFIN` and `Caprese` are exceptionally small, suggesting that `Caprese` has *minimal overhead*. We also include latencies of `GRIFFIN` paired with LoRA weights, which although faster than the base model, negates >40% of the time savings that `GRIFFIN` has provided. This highlights the efficiency suboptimality of LoRA during inference. Metrics were collected on an NVIDIA L40 GPU using BF16 precision.

Bonus: Effect on Natural Response Length

`Caprese` implicitly encourages brevity, often even producing shorter responses than from the full reasoning model. Intriguingly, this behavior arises despite any enforcement or regularization on response lengths anywhere during training or inference. Shown in Figure 5.5, the shortest mean response length for all MATH-500 problem difficulties is either the full model or `Caprese`. CATS consistently outputs the longest responses, averaging ∼1K more across every sample. With increasing difficulty, all models and methods naturally allocate more inference tokens towards the output. Given CATS and `Caprese` (CATS) achieve similar MATH-500 accuracies to the full DS-Qwen 7B and 14B models, the ability of `Caprese` to cut down the lengthy responses of CATS down to the response lengths of the original model or shorter without compromising performance is a direct memory and latency ben-

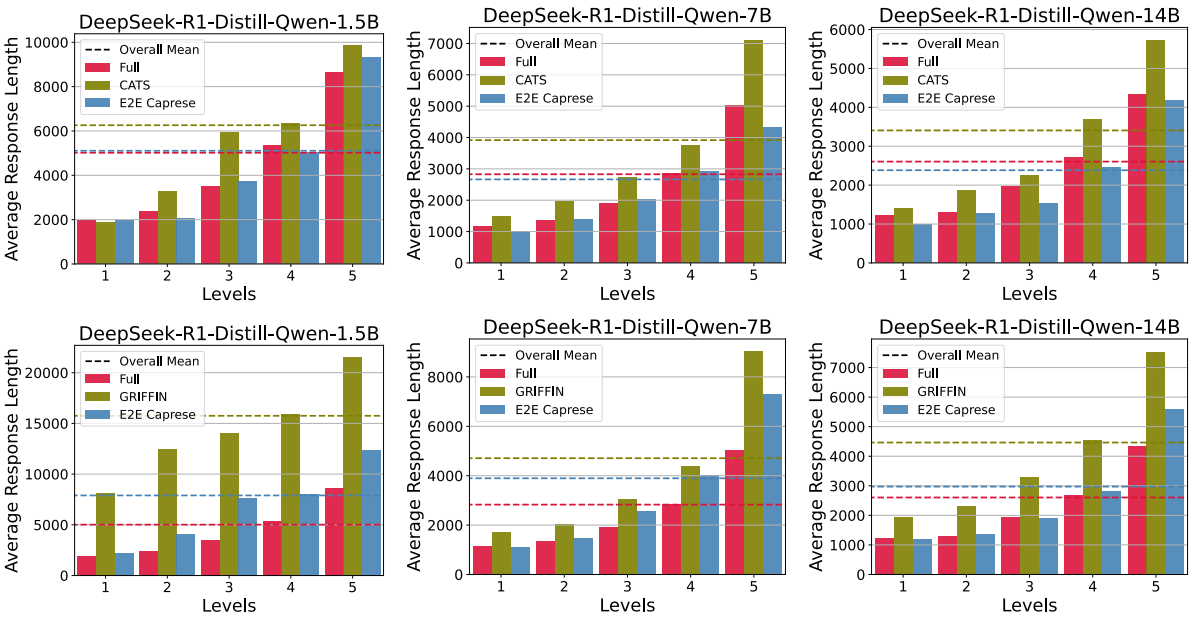

Figure 5.5: Average number of response tokens for MATH-500 queries with increasing problem difficulty for CATS (top) and `GRIFFIN` (bottom). The global averages are indicated by the dashed lines. Sparsity is set at 50%.

efit. This brevity of `Caprese` is more pronounced in larger models, proportionally, from a 5.8% reduction in tokens compared to the full DS-Qwen 7B model to an 8.5% reduction compared to the full DS-Qwen 14B model. For `GRIFFIN`, the difference between `Caprese` (`GRIFFIN`) and the full model is slightly larger but decreasing with model size. Even so, response lengths of `Caprese` is still significantly shorter than `GRIFFIN`.

5.5 Ablations & Analysis

Now, we explore a couple characteristics of `Caprese`. In Section 5.5.1, we show the effective rank of the learned parameters, and in Section 5.5.2, we take a look at the interaction between the sparsity level of the FF method and the rank of `Caprese`.

5.5.1 Rank of Learned Parameters

In Figure 5.6, we plot the relative singular values for each learned product $\boldsymbol{LR}$ from (5.7) in Llama 3.2 3B Instruct. Although we set the inner rank to be 256, we see that some

layers in `Caprese`(CATS) are still very low rank. In comparison, `Caprese`(`GRIFFIN`) layers are relatively high rank, likely to due to the fact that `GRIFFIN` performs highly structured FF sparsification. From this, we hypothesize that `GRIFFIN` may better utilize an increased `Caprese` inner dimension.

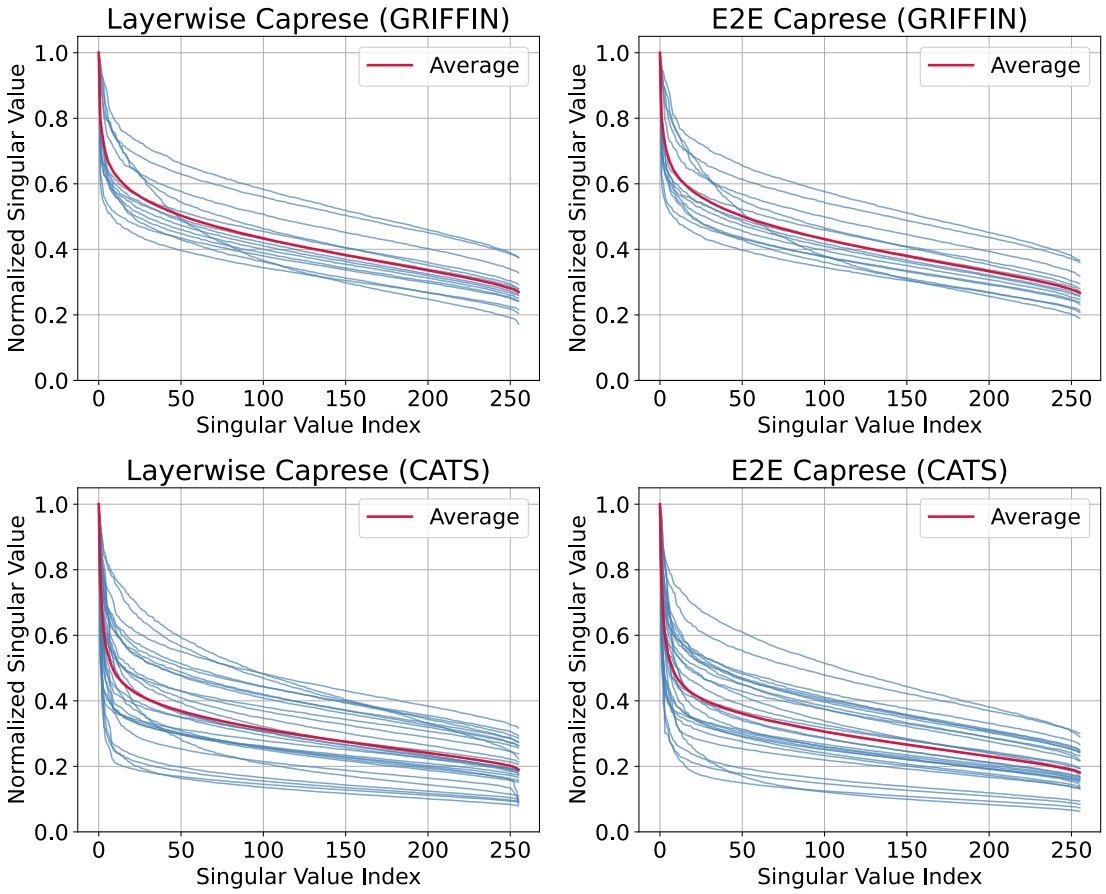

Figure 5.6: Relative singular values of learned Llama 3.2 3B Instruct `Caprese` layers. Blue lines are individual layers; red lines are averaged across layers.

5.5.2 Effect of Rank & Sparsity

Here, we ablate the relationship between varying ranks in `Caprese` and sparsity levels in CATS. Using Llama 3.2 1B Instruct, we show the test performance of MATH in Figure 5.7. The same training procedure and data are used as outlined in Section 5.3. For all ablated ranks, `Caprese` consistently outperforms pure CATS by a large margin when

CATS performs poorly relative to the full model. With a couple of exceptions (perhaps due to randomness in the generation process), greater performance is correlated with higher rank.

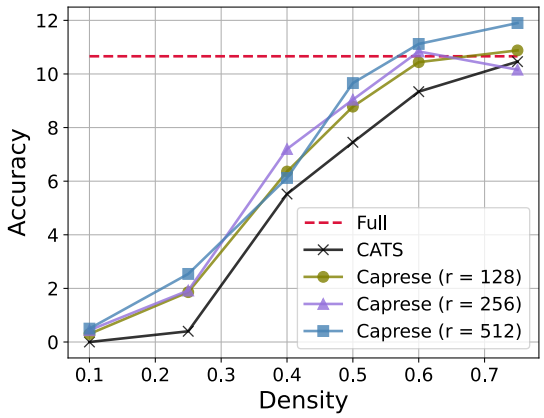

Figure 5.7: Llama 3.2 1B Instruct's performance on MATH with varying densities of CATS and ranks in `Caprese` with end-to-end training.

5.6 Chapter Summary

To combat the inefficiency of the long and brittle generation process associated with reasoning, we introduce `Caprese`, a performant and efficient method with strong reasoning and language capabilities that is compatible with a broad class of efficient FF algorithms. In the future, it would be interesting to evaluate its benefit in more reasoning domains, explore input-dependent low-rank layers, and investigate the knowledge contained in these low-rank layers. `Caprese`, along with future developments, pushes towards the long-term goal of degradation-free efficient LLM inference. Staying the realm of reasoning, we will take a complementary approach in the next chapter where we strive to maximize the potential of each unit of additional compute.

Chapter 6

Underutilized Information in Parallel Scaling

Whereas Chapters 3, 4, and 5 showed we could improve efficiency with little effect on accuracy, this chapter will go in the complementary direction towards the inference compute-performance Pareto frontier. In other words, this chapter introduces a method to boost performance with equivalent computational resources by leveraging underutilized features unique to parallel inference scaling. In doing so, we will also generalize parallel generation for LLMs by elevating computation to a higher dimension. This chapter covers material from Dong et al. (2026b).

6.1 Parallel Scaling Features as Tensors

Now we focus on improving parallel scaling by enhancing reasoning performance of LLMs. Many scaling methods concentrate resources to generate a *single* high-quality response such as with CoTs (Wei et al., 2022) and decompositions of a problem into parallel substeps (Rodionov et al., 2025; Yang et al., 2025b). However, there are also instances where a high-quality *set* of responses for each input is needed, such as in the case of output synthesis, best-of-N selection, and synthetic data generation. Scaling this in a parallel manner is traditionally done by sampling independent generations. Consequently, each generation is

ignorant of the other rollouts, despite answering the same prompt. Independent generations for the same prompt leave potentially useful information derived from other responses unutilized, limiting the performance ceiling. In contrast, sequentially scaling CoTs ensures each sampled token can play a role in the final output. Motivated by the potential of shared information across parallel generations stemming from the same prompt, we aim to leverage these interactions to enhance and generalize parallel inference scaling.

There has been progress in integrating some form of parallel dependence for inference. A line of work explores breaking down reasoning steps into parallel paths with great success (Hsu et al., 2025; Jin et al., 2025; Pan et al., 2025a; Rodionov et al., 2025; Yang et al., 2025b). In these N-to-1 cases, parallel computation is funneled into a single output, useful for generating one high-quality response but not a high-quality *set* of responses. Even so, they highlight the potential of mid-generation interactions between sequences. We seek to generalize parallel scaling with *interdependence*, which allows all N output sequences for one prompt to use all the compute and information available, not just a single isolated partition (N-to-N). *Thus, the challenge is finding a totally parallel method that uses N simultaneous threads to generate N responses with interdependence without extensive post-training.*

Looking at the operations on LLM hidden states reveals a clue to overcome these challenges (Figure 6.1). For batch size B, sequence length S, and hidden dimension D, the hidden states per forward pass has 3-D shape $B \times S \times D$. Attention and feedforward blocks blend information throughout each $S \times D$ slice with the batch dimension kept independent. Even minor inter-sample interactions like Batch Normalizations (Ioffe and Szegedy, 2015) were substituted with Layer Normalizations (Ba et al., 2016). While this is natural for highly heterogeneous batches where samples with wildly different inputs can be fed together in the same forward pass without interference, *parallel scaling, which draws many responses from a single input, exhibits uniquely homogeneous structure since each output stems from the same input.* Hence, there is the potential for useful information transfer during the generation process which we exploit.

We introduce `Bridge` (**B**atch **r**easoning with **i**nter**d**ependent **ge**nerations), a method

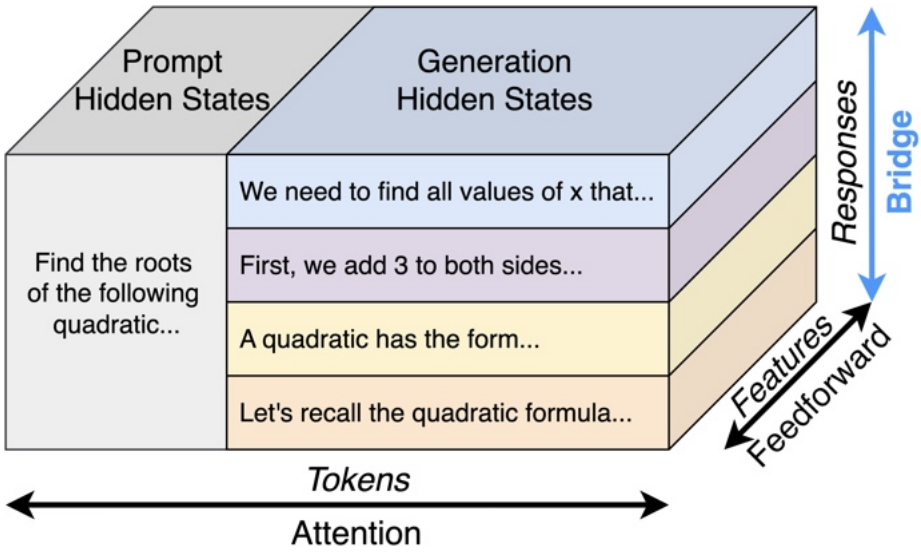

Figure 6.1: LLM hidden states are 3-D tensors, where attention and feedforward blocks explicitly transfer information between tokens and features, respectively. By instead treating parallel scaling generations as a single tensor rather than independent slices, our method, `Bridge`, operates along the batch axis, so that tokens from all sequences that share the same prompt can share information throughout generation.

that shares information across tokens that stem from the same prompt in a batch for parallel scaling with interdependent generations (Dong et al., 2026b). With a minor architectural change to LLMs, each token generated in a batch can depend on tokens in other generation threads with the same prompt. In turn, our method improves reasoning performance evaluated both at the individual response level (accuracy) and response set level (G-Pass@k_τ (Liu et al., 2025)). Furthermore, our method focuses on generation, so any post-generation aggregation technique can be used. We push the following advancements towards parallel scaling:

1. **Parallelism with Dependence:** Instead of generating in isolated silos, `Bridge` allows information to flow between sequences while maintaining complete generation parallelism. Thus, inference compute is pooled together for all tokens, rather than being partitioned. We show `Bridge` significantly increases the final performance after reinforcement learning with verifiable rewards (RLVR) on 7 math and 5 non-math benchmarks using multiple reasoning models.
2. **Low Cost:** By adding only 2.8% to 5.1% additional parameters, and warming up

on a small supervised fine tuning (SFT) dataset (e.g. GSM8K (Cobbe et al., 2021)), `Bridge` already significantly improves the effectiveness of RLVR.

3. **Versatility:** `Bridge` has no restriction on the width of parallelism and is robust to train-time and test-time width discrepancies. Trained once, all tested widths outperform independent generations in terms of accuracy, coverage, and consistency. Furthermore, `Bridge` does not rely on any heuristics or interventions at any point in the generation process.

Our extensive experiments on multiple models and tasks show that `Bridge` effectively shares information across multiple generations for the same input. For example, our method improves the relative benefit of RLVR on DeepSeek-R1-Distill-Qwen-7B by 39% averaged over 7 math tasks, compared to the next best method. With the same model, `Bridge` also increases the rate at which all responses to a single competition math problem are correct from 15.0% to 17.8%.

In Section 6.2, we introduce `Bridge`, detailing the algorithm, the training pipeline, and its implications. We demonstrate `Bridge`'s efficacy on a variety of reasoning datasets, evaluated both on sample-wise accuracy and on global response set quality in Section 6.3. Finally, in Section 6.4, we highlight some important characteristics of our method including the versatility of generation width, length extrapolation, an example of post-generation aggregation, our design choices, and additional effects on features and outputs.

6.2 `Bridge`: Latent Information Exchange

Sharing information between samples mid-generation in the latent space gives rise to a couple technical challenges. One is finding an effective and efficient way to achieve this. Attention and feedforward blocks already pose serious static and dynamic memory bottlenecks, which we want to avoid accentuating while still improving accuracy. Second, we also need versatility to allow for any number of parallel generations at test-time. `Bridge` overcomes these challenges with small attention-like blocks that fit into any LLM. We begin with a description of `Bridge`, its connections, and its implications, followed by details

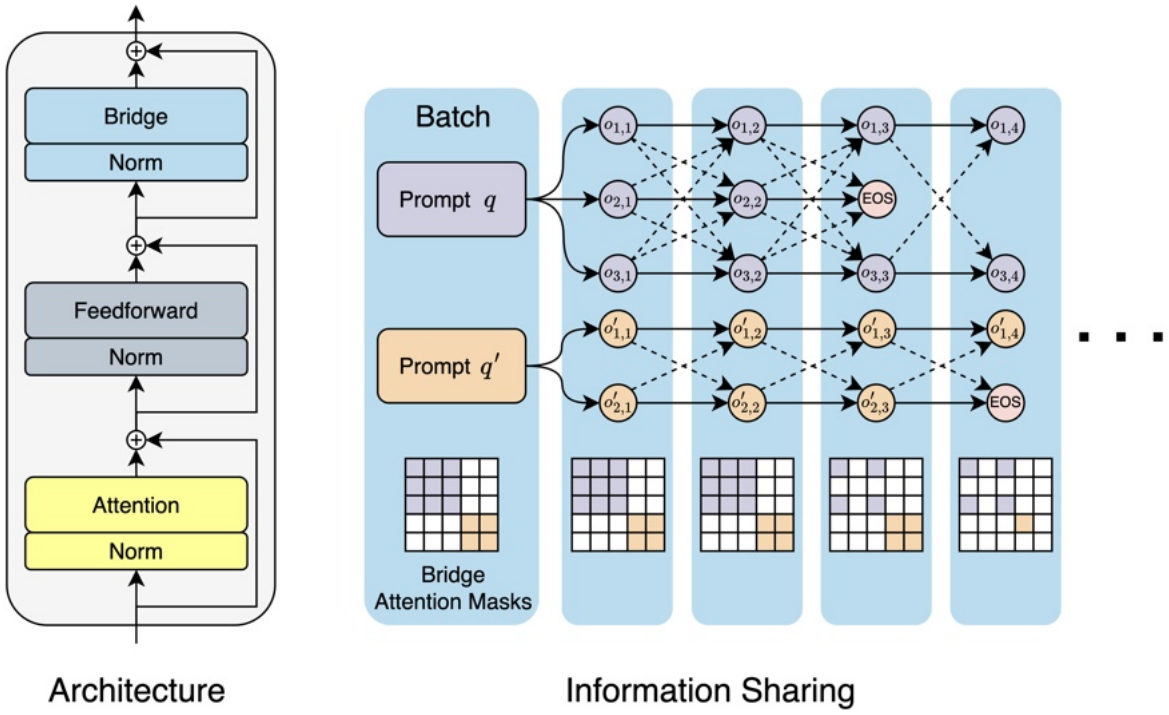

Figure 6.2: `Bridge` design. **(Left)** A `Bridge` block and input normalization layer are added after each feedforward block. **(Right)** A timestep's tokens stemming from the same input prompt attend to each other in `Bridge` blocks, denoted by the arrows. Dotted arrows illustrate all the locations of information transfer to different sequences in a Markovian fashion (token features only at the current timestep are shared to predict the next timestep's tokens). Attention is masked for tokens from different prompts and from completed generations. White squares are masked cells.

on SFT warm up (Section 6.2.2) and RLVR (Section 6.2.3).

6.2.1 `Bridge` Architecture

We introduce `Bridge`, a new transformer block that introduces dependence between samples in a batch. At a high level, `Bridge` performs attention between tokens, which share the same prompt and do not come from completed generations, in a batch at each timestep. We summarize our method in Figure 6.2 and Algorithm 2.

To start, we first describe self-attention layers. Define hidden states $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{B \times S \times D}$ for batch size B, sequence length S, and hidden dimension D. Let $[\boldsymbol{\mathcal{X}}]_{b,\cdot,\cdot}$ and $[\boldsymbol{\mathcal{X}}]_{\cdot,s,\cdot}$ be the b-th and s-th 2-D slices along the batch and sequence axes, respectively. Self-attention,

parameterized by $\boldsymbol{W}_{\mathrm{A,Q}}, \boldsymbol{W}_{\mathrm{A,K}} \in \mathbb{R}^{D \times D_{\mathrm{QK}}}$ and $\boldsymbol{W}_{\mathrm{A,V}}, \boldsymbol{W}_{\mathrm{A,O}}^{\top} \in \mathbb{R}^{D \times D_{\mathrm{VO}}}$, is calculated independently for each sample b:

$$\boldsymbol{Q}_{\mathrm{A},b} = [\boldsymbol{\mathcal{X}}]_{b,\cdot,\cdot} \boldsymbol{W}_{\mathrm{A,Q}}, \qquad \boldsymbol{K}_{\mathrm{A},b} = [\boldsymbol{\mathcal{X}}]_{b,\cdot,\cdot} \boldsymbol{W}_{\mathrm{A,K}}, \qquad \boldsymbol{V}_{\mathrm{A},b} = [\boldsymbol{\mathcal{X}}]_{b,\cdot,\cdot} \boldsymbol{W}_{\mathrm{A,V}},$$

$$[\mathrm{Attn}(\boldsymbol{\mathcal{X}})]_{b,\cdot,\cdot} = \underbrace{\mathrm{Softmax}(\mathrm{Mask}_{\mathrm{A}}(\boldsymbol{Q}_{\mathrm{A},b} \boldsymbol{K}_{\mathrm{A},b}^{\top}))}_{\in \mathbb{R}^{S \times S}} \boldsymbol{V}_{\mathrm{A},b} \boldsymbol{W}_{\mathrm{A,O}}. \tag{6.1}$$

`Bridge` blocks are similar, but attention between samples is calculated independently for each token index s. Letting $\boldsymbol{W}_{\mathrm{B,Q}}, \boldsymbol{W}_{\mathrm{B,K}} \in \mathbb{R}^{D \times D_{\mathrm{QK}}}$ and $\boldsymbol{W}_{\mathrm{B,V}}, \boldsymbol{W}_{\mathrm{B,O}}^{\top} \in \mathbb{R}^{D \times D_{\mathrm{VO}}}$,

$$\boldsymbol{Q}_{\mathrm{B},s} = [\boldsymbol{\mathcal{X}}]_{\cdot,s,\cdot} \boldsymbol{W}_{\mathrm{B,Q}}, \qquad \boldsymbol{K}_{\mathrm{B},s} = [\boldsymbol{\mathcal{X}}]_{\cdot,s,\cdot} \boldsymbol{W}_{\mathrm{B,K}}, \qquad \boldsymbol{V}_{\mathrm{B},s} = [\boldsymbol{\mathcal{X}}]_{\cdot,s,\cdot} \boldsymbol{W}_{\mathrm{B,V}},$$

$$[\texttt{Bridge}(\boldsymbol{\mathcal{X}})]_{\cdot,s,\cdot} = \underbrace{\mathrm{Softmax}(\mathrm{Mask}_{\mathrm{B}}(\boldsymbol{Q}_{\mathrm{B},s} \boldsymbol{K}_{\mathrm{B},s}^{\top}))}_{\in \mathbb{R}^{B \times B}} \boldsymbol{V}_{\mathrm{B},s} \boldsymbol{W}_{\mathrm{B,O}}. \tag{6.2}$$

Algorithm 2 sketches the pseudocode for `Bridge` blocks, following (6.2). Like normal self-attention, this can easily be extended to multiple heads. However, there are 3 key differences between usual self-attention and `Bridge` beyond a transposition of $\boldsymbol{\mathcal{X}}$:

- Instead of a decoder mask, `Bridge` applies an attention mask that omits attention to tokens from sequences stemming from different prompts and sequences that have completed generation. See Figure 6.2 for an example.
- No positional encoding is used to preserve sample position invariance.
- Without attention to previous tokens, `Bridge`'s Markovian design *does not maintain a key-value cache.*

We place a `Bridge` block after each feedforward block with a residual stream and input normalization layer that mimics existing blocks, shown in Figure 6.2. `Bridge` is active during the prefill stage too, but since all hidden states for the same input are identical, `Bridge` blocks act as linear layers.

Algorithm 2: `Bridge` Block

Input: $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{B \times S \times D}$
Parameters: $\boldsymbol{W}_{\mathrm{Q}}, \boldsymbol{W}_{\mathrm{K}} \in \mathbb{R}^{D \times D_{\mathrm{QK}}}; \boldsymbol{W}_{\mathrm{V}}, \boldsymbol{W}_{\mathrm{O}}^{\top} \in \mathbb{R}^{D \times D_{\mathrm{VO}}}$
Output: $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{B \times S \times D}$
$\boldsymbol{Q}_s \leftarrow [\boldsymbol{\mathcal{X}}]_{\cdot,s,\cdot} \boldsymbol{W}_{\mathrm{Q}}$ for $s = 1, \ldots, S$
$\boldsymbol{K}_s \leftarrow [\boldsymbol{\mathcal{X}}]_{\cdot,s,\cdot} \boldsymbol{W}_{\mathrm{K}}$ for $s = 1, \ldots, S$
$\boldsymbol{V}_s \leftarrow [\boldsymbol{\mathcal{X}}]_{\cdot,s,\cdot} \boldsymbol{W}_{\mathrm{V}}$ for $s = 1, \ldots, S$
Construct mask $\boldsymbol{M}_s \in \mathbb{R}^{B \times B}$ for $s = 1, \ldots, S$:
  $[\boldsymbol{M}_s]_{b_1,b_2} = 0$ if generations b_1, b_2 have the same prompt and are incomplete at token s.
  $[\boldsymbol{M}_s]_{b_1,b_2} = -\infty$ otherwise.
$[\boldsymbol{\mathcal{Y}}]_{\cdot,s,\cdot} \leftarrow \mathrm{Softmax}\left(\frac{\boldsymbol{Q}_s \boldsymbol{K}_s^{\top}}{\sqrt{D_{\mathrm{QK}}}} + \boldsymbol{M}_s\right) \boldsymbol{V}_s \boldsymbol{W}_{\mathrm{O}}$ for $s = 1, \ldots, S$
Return: $\boldsymbol{\mathcal{Y}}$

Connection to Efficient Attention for Tensors

`Bridge` unlocks the ability for an LLM to treat a batch of LLM hidden states as a 3-D ($B \times S \times D$) structure rather than a stack of independent 2-D slices. In this way, the inputs are analogous to images, and the decoding process is like autoregressively generating additional columns. With this interpretation, `Bridge` applying attention operations on different axes of an input is similar to axial attention (Ho et al., 2019) which was introduced first in computer vision to accelerate encoder attention but has since seen wide success in various applications such as in medicine (Azad et al., 2024), materials science (Dong et al., 2023b), and algorithm discovery (Fawzi et al., 2022).

Generation Interdependence

For B independent rollouts we sample the next token $[\boldsymbol{o}_b]_{s+1}$ from the distribution,

$$p([\boldsymbol{o}_b]_{s+1} | \boldsymbol{q}, [\boldsymbol{o}_b]_{1:s}),$$

for response sample b, timestep s, input vector of prompt tokens $\boldsymbol{q}$, and previously generated tokens $[\boldsymbol{o}_b]_{1:s}$. With `Bridge`, the next token distribution becomes

$$p([\boldsymbol{o}_b]_{s+1} | \boldsymbol{q}, \{[\boldsymbol{o}_{b'}]_{1:s}\}_{b'=1}^{B})$$

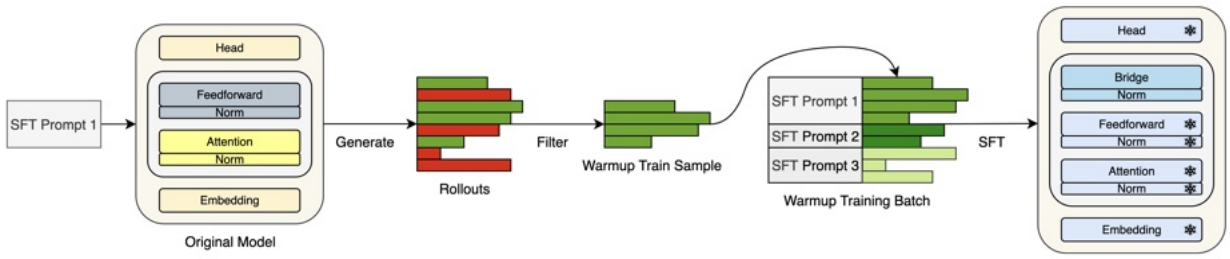

Figure 6.3: `Bridge` and P-Match warm up procedure. The original LLM generates candidate traces which are filtered by correctness and compiled into a dataset. SFT on this generated dataset only updates new parameters. The P-Match baseline substitutes `Bridge` blocks with MLPs matched in parameter count.

for each sample b. Conditioned on past tokens, `Bridge` preserves independence between tokens at the same timestep, which allows next token sampling to still be performed in parallel:

$$([\boldsymbol{o}_{b_1}]_{s+1} \perp\!\!\!\perp [\boldsymbol{o}_{b_2}]_{s+1}) | (\boldsymbol{q}, \{[\boldsymbol{o}_{b'}]_{1:s}\}_{b'=1}^{B}) \text{ for } b_1 \neq b_2.$$

6.2.2 SFT Warm up

While RLVR can be immediately applied with `Bridge` since these new blocks are initialized to have no contribution, we can also optionally warm them up with SFT for more sufficient training and better downstream performance. A desirable SFT dataset would include many reasoning traces to one prompt. To stay close to the original LLM's generation distribution, we create SFT datasets by first responding to prompts from an existing math dataset. Then, traces are filtered for correctness. During training, these correct traces are fed together in the same batch to warm up `Bridge` blocks with SFT. All other parameters are frozen. Figure 6.3 illustrates the warm up procedure, and Table 6.6 explores more in-depth on the benefits of warm up.

6.2.3 RLVR Objective

We train LLMs with `Bridge` using GRPO (Shao et al., 2024). We use a variant specified by Yu et al. (2025) which performs token-level normalization to reduce length bias. Letting

the group size be G, the advantage of the i-th output $\boldsymbol{o}_i$ to input $\boldsymbol{q}$ with reward r_i is $\widehat{A}_i = \frac{r_i - \text{mean}(r_1,\ldots,r_G)}{\text{std}(r_1,\ldots,r_G)}$. Then, for clipping threshold ϵ, hyperparameter β, and policy π_θ parameterized by θ, the objective is

$$\mathcal{J}(\theta) = \frac{1}{\sum_{i=1}^{G}|\boldsymbol{o}_i|}\sum_{i=1}^{G}\sum_{s=1}^{|\boldsymbol{o}_i|}\left\{\min\left[R_{i,s}(\theta)\widehat{A}_i, \text{clip}(R_{i,s}(\theta), 1-\epsilon, 1+\epsilon)\widehat{A}_i\right] - \beta D_{\text{KL}}(\pi_\theta||\pi_{\theta_{\text{ref}}})\right\}, \tag{6.3}$$

where

$$R_{i,s}(\theta) = \frac{\pi_\theta([\boldsymbol{o}_i]_s|\boldsymbol{q}, \{[\boldsymbol{o}_j]_{1:s-1}\}_{j=1}^{G})}{\pi_{\theta_{\text{old}}}([\boldsymbol{o}_i]_s|\boldsymbol{q}, \{[\boldsymbol{o}_j]_{1:s-1}\}_{j=1}^{G})},$$
$$D_{\text{KL}}(\pi_\theta||\pi_{\theta_{\text{ref}}}) = \frac{\pi_{\theta_{\text{ref}}}([\boldsymbol{o}_i]_s|\boldsymbol{q}, \{[\boldsymbol{o}_j]_{1:s-1}\}_{j=1}^{G})}{\pi_\theta([\boldsymbol{o}_i]_s|\boldsymbol{q}, \{[\boldsymbol{o}_j]_{1:s-1}\}_{j=1}^{G})} - \log\frac{\pi_{\theta_{\text{ref}}}([\boldsymbol{o}_i]_s|\boldsymbol{q}, \{[\boldsymbol{o}_j]_{1:s-1}\}_{j=1}^{G})}{\pi_\theta([\boldsymbol{o}_i]_s|\boldsymbol{q}, \{[\boldsymbol{o}_j]_{1:s-1}\}_{j=1}^{G})} - 1.$$

The key differences from the GRPO objective (and its variants) and our objective are not formulaic but rather inherently induced from the architecture of `Bridge`*.* Namely, the ratio and KL divergence terms now contain inter-sample dependence between relevant samples, breaking the original assumption of independent trajectories. By linking the advantages and logits in a group, the loss and gradients per output are intertwined with other outputs' that share the same prompt. In other words, gradients from all sequences, containing both positive and negative advantages, are backpropagated through each sequence because of `Bridge` blocks.

While our method inserts dependence between sequence and therefore their corresponding rewards, the sample permutation invariance of `Bridge` blocks means the unconditional rewards are still identically distributed. This implies

$$\mathbb{E}(\widehat{A}_i) = \mathbb{E}\left(\frac{r_i - \frac{1}{n}\sum_{j=1}^{G} r_j}{\text{std}(r_1,\ldots,r_G)}\right) = \mathbb{E}\left(\frac{r_i - \frac{1}{n}\cdot n r_i}{\text{std}(r_1,\ldots,r_G)}\right) = 0, \tag{6.4}$$

preserving unbiasedness of the advantage. To preserve some notion of independence between rollouts for GRPO, one can generate multiple groups per prompt with `Bridge` and compute advantages between groups. Though this deserves exploration in future work, we

do not do this here as it would be computationally expensive, and our single group setup is already empirically performative.

Since `Bridge` is just an architectural change, training is not just limited to SFT and RLVR for reasoning problems. For instance, `Bridge` may also be applied for reinforcement learning from human feedback (RLHF) which is an interesting future direction.

6.3 Core Experiments

We now showcase the benefit of `Bridge` across multiple models and math reasoning benchmarks. After describing our setup in Section 6.3.1, we first show that applying RLVR with `Bridge` blocks improves accuracy more than other methods. For instance, DeepSeek-R1-Distill-Qwen-7B with `Bridge` blocks observes a relative 39% further improvement with RLVR than the next best method (Section 6.3.2). Then, in Section 6.3.3, we demonstrate that `Bridge` also improves the output set quality across several metrics in terms of coverage and correctness consistency.

6.3.1 Experimental Settings

Models and Baselines

We test `Bridge` on DeepSeek-R1-Distill-Qwen-1.5B, DeepSeek-R1-Distill-Qwen-7B, and DeepSeek-R1-Distill-Llama-8B, which we abbreviate to DS-Qwen-1.5B, DS-Qwen-7B, and DS-Llama-8B, respectively (Dubey et al., 2024; Guo et al., 2025; Yang et al., 2024a). We use 4 query and key-value attention heads for `Bridge`, each with the same dimension as the original model's head dimension. This only adds 5.1%, 2.8%, and 3.4% extra parameters on top of the original DS-Qwen-1.5B, DS-Qwen-7B, and DS-Llama-8B models, respectively. Table 6.1 lists the exact parameter counts. Our parameter-matched baseline which we call "P-Match" adds 2-layer MLPs of the same size in the same positions as `Bridge` blocks which serves to show the limited effect of just adding parameters. Matched in parameter count, P-Match and `Bridge` are also trained with the same warm up and RLVR pipeline. Both methods are initialized to have zero contribution.

Table 6.1: Distribution of parameters (B) across embedding/head, attention (Attn), feed-forward (FF), and `Bridge` blocks.

Model	Embed & Head	Attn	FF	`Bridge`	Orig. Total	New Total
DS-Qwen-1.5B	0.47	0.15	1.16	0.09	1.78	1.86
DS-Qwen-7B	1.09	0.82	5.70	0.21	7.62	7.82
DS-Llama-8B	1.05	1.34	5.64	0.27	8.03	8.30

Training

For the SFT warm up stage, we first use the original LLM to generate 8 response for each GSM8K (Cobbe et al., 2021) problem and then filter out incorrect responses and problems with one or fewer correct responses. We train only the additional parameters with `Bridge` and P-Match on this custom dataset for 5 epochs and keeping the best checkpoint according to the perplexity on 500 validation problems (and their corresponding set of correct reasoning traces). This checkpoint is inserted in the model for RLVR where we train the full model on DeepScaleR-Preview-Dataset (Luo et al., 2025) for 1000 gradient steps. DS-Qwen-1.5B is trained with generation width 8 while the others were trained with 4. The only reward is correctness of the generation. Table 6.2 lists the hyperparameters used for RLVR and SFT.

Evaluation

We evaluate `Bridge` on 7 math benchmarks (MATH-500 (Hendrycks et al., 2021; Lightman et al., 2023), AIME24, AIME25 (AIME, 2025), AMC23 (AMC, 2023), BRUMO25 (BRUMO, 2025), CMIMC25 (CMIMC, 2025), and HMMT_FEB25 (HMMT, 2025)) and 5 challenging non-math benchmarks (XSum (Narayan et al., 2018), CNN/ DailyMail (Hermann et al., 2015; See et al., 2017), GPQA (Rein et al., 2024), ZebraLogic (Lin et al., 2025), and Countdown (Pan et al., 2025b)). Evaluating MATH-500 on every 100 training steps, the checkpoint with the highest validation accuracy is used to test on the remaining benchmarks. We evaluate across 4 responses per MATH-500 sample and 32 responses per sample from the other benchmarks. Sampling temperature and top-p are set to 0.6 and

Table 6.2: `Bridge` training hyperparameters.

Hyperparameter	RLVR	SFT & Warmup Data Generation
Mixed precision	BF16	BF16
Optimizer	AdamW	AdamW
KL penalty	1e-3	N/A
Learning rate	1e-6	2e-5
LR scheduler	constant	linear
LR warmup	10%	10%
Training steps	1000	varies
Batch size (rollouts × samples)	448	32
Rollouts per sample	8	[2,8]
Total samples	7000	varies
Max length	4096	2048
Steps per rollout	8	N/A
Temperature	0.6	0.6
Heads	4	4
Generation width	4 or 8	[2,4]

0.95, respectively. We set the generation width of `Bridge` to 8 for all tasks except MATH-500, which we set to 4 since we only evaluate on 4 responses per sample. We adapt our evaluations from the Lighteval framework (Habib et al., 2023).

6.3.2 Reasoning Accuracy

Beginning with standard accuracy (Pass@1), we compare the performance of the original model, original model with RLVR, P-Match with SFT and RLVR, and `Bridge` with SFT and RLVR on several math benchmarks. Results in Table 6.3 show that in nearly all cases and on average, `Bridge` obtains the highest accuracy compared to all other methods. *In particular, the average performance improvements of our method on the original model is 26%, 39%, and 34% relatively more than that of the next best method on DS-Qwen-1.5B, DS-Qwen-7B, and DS-Llama-8B models, respectively.* P-Match with parameter counts pegged to `Bridge` improves accuracy from just pure RLVR most of the time but is much more inconsistent, such as in the case of DS-Qwen-7B. This indicates that the superior performance of `Bridge` is not solely attributed to additional parameters. Furthermore,

Table 6.3: Accuracy comparison across math benchmarks. In each section, the 4 rows from top to bottom are the performance of the original model, RLVR applied on the original model, P-Match (extra MLPs) with SFT warm up and RLVR, and `Bridge` with SFT warm up and RLVR. The 2 rightmost columns show the average across all benchmarks and the average improvement over the original model. MATH-500, AMC23, BRUMO25, CMIMC25, and HMMT_FEB25 are abbreviated to MATH, AMC, BRU, CMI, and HMMT, respectively.

Model	MATH	AIME24/25	AMC	BRU	CMI	HMMT	Avg	↑ Δ
DS-Qwen-1.5B	73.65	13.75 / 13.44	50.00	18.12	4.30	8.23	25.93	0.00
RLVR only	78.75	17.40 / 18.44	60.55	18.54	3.83	7.50	29.29	3.36
P-Match	78.65	18.12 / 19.17	**60.62**	20.94	5.08	8.54	30.16	4.23
`Bridge`	**81.30**	**20.11** / **20.00**	60.55	**21.36**	**5.63**	**9.79**	**31.25**	**5.32**
DS-Qwen-7B	82.15	23.44 / 21.88	66.02	23.75	5.63	11.98	33.55	0.00
RLVR only	**88.15**	29.06 / 23.85	74.30	28.33	7.97	**12.60**	37.75	4.20
P-Match	86.80	28.85 / **25.73**	70.47	26.77	6.25	11.87	36.68	3.13
`Bridge`	**88.15**	**32.19** / 25.41	**77.65**	**30.21**	**9.77**	12.40	**39.40**	**5.85**
DS-Llama-8B	73.40	15.42 / 13.12	57.97	15.62	2.73	8.23	26.64	0.00
RLVR only	76.70	18.12 / 18.12	63.44	15.83	5.47	10.52	29.74	3.10
P-Match	78.00	22.29 / **20.21**	61.80	17.81	5.08	11.67	30.98	4.34
`Bridge`	**80.15**	**24.76** / 18.18	**66.36**	**19.91**	**6.02**	**11.93**	**32.47**	**5.83**

even though DS-Qwen-7B and DS-Llama-8B were trained with generation width 4, the evaluation results with width 8 are still stronger than the other independent sampling methods, showing the robustness of `Bridge`. In addition, the improvement by `Bridge` is greater for larger models, and scaling up to even larger ones remains of interest for future work. Although we train `Bridge` solely on math, we observe no degradation and sometimes improvement on non-math tasks (Table 6.4).

6.3.3 Set-level Evaluations

Zooming out, we show `Bridge` also improves the consistency and coverage (i.e., the percentage of questions that have at least 1 correct response in the response set) across multiple generation attempts. To evaluate the set of responses to a single input, we use the G-Pass@k_τ (Liu et al., 2025) metric, which paints a more holistic picture of model potential (coverage) and consistency. Whereas Pass@k is the probability of a correct output in k

Table 6.4: Evaluations on non-math tasks. Note that our training procedure only used math samples. Rouge-1 (Lin, 2004b) scores are reported for summarization (XSum and CNN/DailyMail). Average accuracies are reported for GPQA, ZebraLogic, and Countdown.

Model	XSum	CNN/DailyMail	GPQA	ZebraLogic	Countdown
DS-Qwen-1.5B	15.72	22.11	33.14	30.90	28.77
RLVR only	14.81	22.79	32.45	30.90	28.15
P-Match	15.90	22.78	32.51	32.55	31.36
`Bridge`	**17.17**	**24.07**	**33.90**	**33.15**	**34.84**
DS-Qwen-7B	18.03	24.19	43.94	40.00	49.55
RLVR only	17.24	23.52	43.56	41.25	49.93
P-Match	18.13	23.76	43.75	**42.95**	46.91
`Bridge`	**18.16**	**24.55**	**45.77**	42.60	**52.70**
DS-Llama-8B	18.23	**23.84**	35.80	41.25	14.04
RLVR only	2.25	1.65	39.46	43.25	29.23
P-Match	**19.67**	23.01	38.83	43.50	32.32
`Bridge`	18.04	22.64	**39.65**	**44.70**	**32.51**

responses, G-Pass@k_τ is the probability of $0 < \tau \leq 1$ fraction of k responses being correct. More formally, for n responses and c correct responses,

$$\text{Pass@}k = \mathbb{E}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right], \qquad \text{G-Pass@}k_\tau = \mathbb{E}\left[\sum_{j=\lceil \tau k \rceil}^{c} \frac{\binom{c}{j} \cdot \binom{n-c}{k-j}}{\binom{n}{k}}\right].$$

As $\tau \to 0$, G-Pass@k_τ is simply the coverage. On the other extreme, G-Pass@k_1 is the probability that all k responses are correct.

From Figure 6.4, `Bridge` achieves higher G-Pass@8_τ values for nearly all values of τ and models. This demonstrates that `Bridge` can achieve greater coverage without spreading out its responses to many incorrect answers. *In other words, not only do `Bridge` blocks increase the probability of a correct response in the response set more than the other methods, they also increase the frequency at which they occur.* Again, we note that Qwen-7B and DS-Llama-8B were trained with generation width 4 yet they generalize well to evaluation width 8.

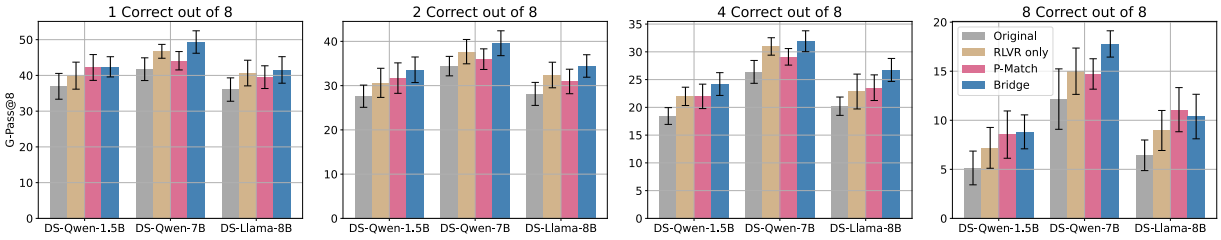

Figure 6.4: G-Pass@8_τ averaged across AIME24, AIME25, AMC23, BRUMO25, CMIMC25, and HMMT_FEB25. Each chart measures the minimum number of correct answers ($\tau \cdot k$) out of $k = 8$ simultaneous responses. `Bridge` has the greatest coverage ($\tau \cdot k = 1$) and answers correctly most consistently ($\tau \cdot k > 1$) in the vast majority of cases. Higher is better.

6.4 Ablations and Analysis

Now that we have demonstrated `Bridge`'s performance, we explore several other aspects of our method, including parallel width extrapolation (Section 6.4.1), length extrapolation (Section 6.4.2), self-consistency aggregation (Section 6.4.3), algorithmic design choices (Section 6.4.4), and additional effects on outputs beyond performance (Section 6.4.5).

6.4.1 Generation Width Extrapolation

The design of `Bridge` allows complete flexibility in the number of parallel generations, or generation width w, due to the removal of positional encoding. Here, we show its generalizability to other widths on DS-Qwen-7B which was trained on a width of 4 with RLVR. In Table 6.5, in all cases where $w > 1$, `Bridge` outperforms P-Match in terms of task-wise and global average accuracy. We also investigate the effect of w on set quality in Figure 6.5. Again, we generally see a vast improvement upon the original model and P-Match with $w > 1$ for all G-Pass@8_τ settings. *These results show not only the benefit of sharing information via* ***`Bridge`*** *but also the generalizability to widths wider and thinner than its training width.* At the extreme of $w = 1$, equivalent to independent generations, results in average accuracy that falls between RLVR only and P-Match, indicating that `Bridge` blocks do not harm independent reasoning.

Table 6.5: Accuracy across 32 samples of varying `Bridge` generation widths, w, with DS-Qwen-7B which was trained at width 4 with RLVR. A `Bridge` width of 1 is equivalent to independent generation. Tasks are abbreviated as described in Table 6.3.

Method	AIME24	AIME25	AMC	BRU	CMI	HMMT	Avg	$\uparrow \Delta$
DS-Qwen-7B	23.44	21.88	66.02	23.75	5.63	11.98	25.45	0.00
RLVR only	29.06	23.85	74.30	28.33	7.97	12.60	29.35	3.90
P-Match	28.85	**25.73**	70.47	26.77	6.25	11.87	28.32	2.87
`Bridge`								
$w = 1$	28.13	24.48	74.85	28.02	9.07	11.77	29.39	3.94
$w = 4$	31.57	25.63	76.93	28.65	**10.16**	**13.13**	31.01	5.56
$w = 8$	32.19	25.41	**77.65**	**30.21**	9.77	12.40	**31.28**	**5.82**
$w = 16$	**32.92**	25.11	75.70	30.63	8.21	12.50	30.85	5.40

Table 6.6: Accuracy comparison between cold start RLVR and RLVR with SFT warmed up `Bridge` blocks in DS-Qwen-7B. Tasks are abbreviated as described in Table 6.3. 16 responses were collected per task except MATH-500 in which 4 were collected.

	MATH	AIME24/25	AMC	BRU	CMI	HMMT	Avg
Cold Start	**88.70**	33.13 / **26.67**	77.19	29.80	7.04	12.09	39.23
Warmed up	88.15	**33.75** / 25.63	**77.97**	**30.21**	**10.00**	**13.13**	**39.83**

6.4.2 Generation Length Extrapolation

`Bridge` also shows strong generalizability along the length axis. We demonstrate its performance as we extrapolate beyond its training length of 4096, again measuring both individual and set performance. From Figure 6.6, our method scales smoothly and better than the other baselines in most cases. At the individual response level, our method achieves the highest accuracy across all generation lengths. At the set level, `Bridge` blocks increase the number of sets that *only* had correct answers by 6.0% compared to the next best at 16K generation length, illustrating our method's consistency to generate correct answers.

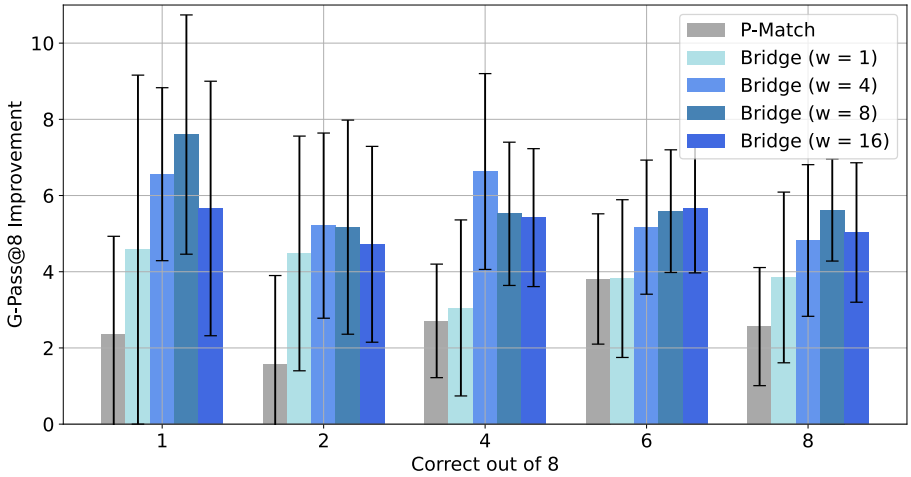

Figure 6.5: G-Pass@8_τ improvement upon the original DS-Qwen-7B model averaged across AIME24, AIME25, AMC23, BRUMO25, CMIMC25, and HMMT_FEB25 with relation to the evaluation generation width w of `Bridge`. The x-axis ($\tau \cdot k$) indicates the number of responses out of $k = 8$ that must be correct.

6.4.3 Self-consistency

Since `Bridge` solely focuses on generation, we can use any post-generation method to synthesize or choose the final output. As an example, Table 6.7 demonstrates `Bridge` coupled with self-consistency (Wang et al., 2022).

6.4.4 Design Choices

There are a couple different choices we can make to integrate `Bridge` blocks into LLMs. We briefly explore the effect of warm-up and the ordering of transformer blocks to provide clarity on our choices.

Cold Start vs. Warm up

Warming up `Bridge` blocks with SFT prior to RLVR outlined in Section 6.2.2 leads to improvements in performance, shown in Table 6.6. The slight improvement implies that although it is preferred to warm up these new layers, it is not catastrophic if RLVR is applied directly from initialization.

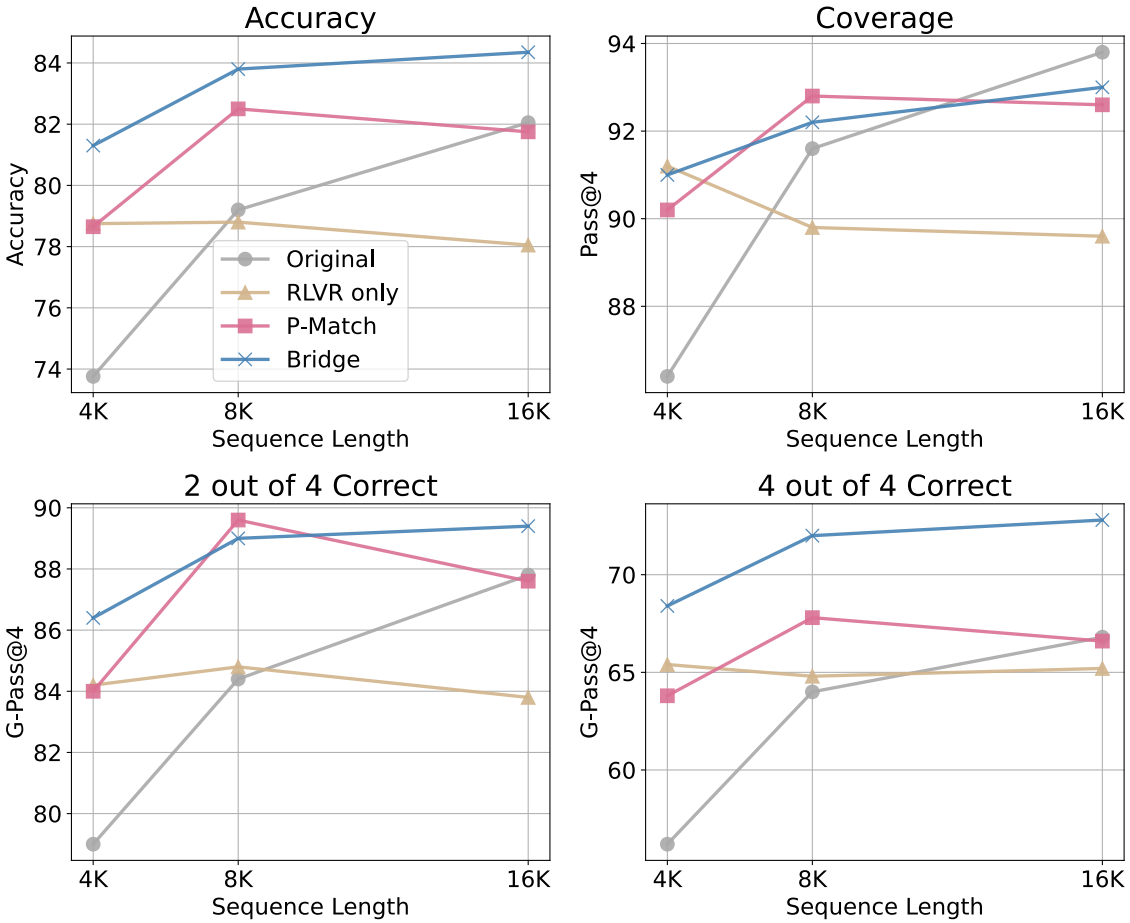

Figure 6.6: DS-Qwen-1.5B MATH-500 accuracy (top left), coverage (top right), G-Pass@$4_{0.5}$ (bottom left), and G-Pass@4_1 (bottom right) as generation length increases. We generate 4 responses per input.

Block Placement

Here, we examine in the architectural placement of `Bridge` blocks. In Table 6.8, we compare the resulting MATH500 accuracy after applying RLVR on DS-Qwen-1.5B with `Bridge` blocks added after attention blocks or after feedforward blocks, our chosen architecture for the experiments. Since there is not a significant difference, this implies flexibility in placement, though we choose to stick with the one with the higher warmed up performance for our experiments.

Table 6.7: Self-consistency accuracy across 32 samples. Tasks are abbreviated as described in Table 6.3.

Model	AIME24	AIME25	AMC	BRU	CMI	HMMT	Avg	↑ Δ
DS-Qwen-1.5B	20.83	19.17	66.87	30.83	13.75	10.83	27.05	0.00
RLVR only	28.33	25.00	71.25	24.17	7.50	13.33	28.26	1.21
P-Match	25.00	26.67	72.50	28.33	8.75	15.00	29.38	2.33
`Bridge`	30.83	24.17	75.00	28.33	11.25	15.00	30.76	**3.71**
DS-Qwen-7B	30.00	26.67	78.12	33.33	8.75	20.83	32.95	0.00
RLVR only	36.67	27.50	83.75	38.33	14.37	17.50	36.35	3.40
P-Match	37.50	29.17	81.87	36.67	13.13	15.00	35.56	2.61
`Bridge`	40.00	30.00	86.25	41.67	17.50	16.67	38.68	**5.73**
DS-Llama-8B	25.00	16.67	73.12	23.33	8.13	13.33	26.60	0.00
RLVR only	25.83	25.00	76.25	25.83	12.50	17.50	30.49	**3.89**
P-Match	25.83	25.00	70.63	25.83	14.37	19.17	30.14	3.54
`Bridge`	31.67	22.50	78.75	24.17	10.00	15.00	30.35	3.75

Table 6.8: Effect on MATH-500 accuracy when inserting `Bridge` blocks after attention blocks vs. after feedforward blocks (chosen architecture) in DS-Qwen-1.5B.

Placement	Cold Start	Warmed up
After Attention	80.20	80.20
After Feedforward	80.15	81.30

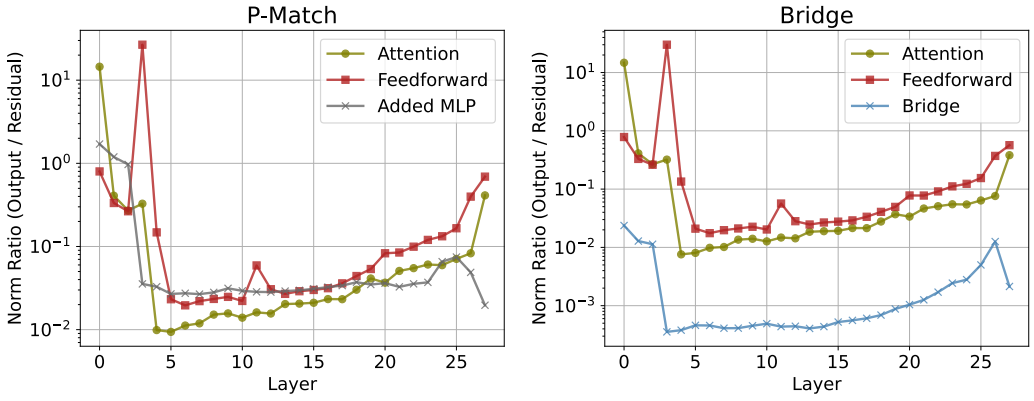

Figure 6.7: Ratio between feature norms of the block output and residual of every DS-Qwen-7B layer.

6.4.5 Auxiliary Effects of `Bridge`

In Section 6.3, we showcased the performance of `Bridge` as measured by various accuracy metrics. Here, we describe its effect (or lack thereof) on intermediate features generation quality stability, and output variance.

Feature Contribution

Having shown the improved performance brought by `Bridge`, we now briefly peer into the effect that it has on LLM hidden states. We measure this by finding the ratio between the output norm of each block with the corresponding residual norm of each token, with lower values suggesting relatively little effect on the residual features (Figure 6.7). Surprisingly, we find `Bridge` blocks contribute little compared to its counterpart in P-Match, despite having a significant impact on the performance.

Output Stability

We additionally measure the effect of `Bridge` on the output tokens in Table 6.9. First, we find the average pair-wise BERTScores (Zhang et al., 2019) between MATH-500 responses, where higher scores indicate more similar output sequences. Our method has a slightly higher BERTScore, meaning `Bridge` marginally increases output similarity but crucially does not collapse the distribution of outputs. Second, we measure the variance in the evaluation results for different responses to the same prompt. For this, we turn to summarization tasks where the evaluation metric (Rouge) of a single response is more granular than the 0-1 nature of math tasks. With the lowest variance, `Bridge` produces outputs with the most consistent quality.

6.5 Chapter Summary

To generalize and enhance parallel inference scaling for LLMs, we introduce `Bridge`, a novel and inexpensive architectural addition to LLMs that allows parallel generations for the same input to share information with each other throughout the decoding process. We

Table 6.9: DS-Qwen-7B BERTScores (F1) for MATH-500 and Rouge-1 variances of summarization tasks. Higher BERTScores indicate greater similarity of outputs.

	MATH-500 BERTScore	CNN/DailyMail Variance	XSum Variance
DS-Qwen-7B	89.93	16.14	21.21
RLVR only	90.06	27.96	33.15
P-Match	89.89	16.09	22.29
`Bridge`	90.41	14.44	20.23

demonstrate that our method improves both single sample accuracy and set-wise quality across multiple models and several reasoning tasks. We achieve this by rethinking hidden states in parallel scaling as higher order tensors rather than disjoint slices. With this interpretation, this also plants the seeds for many exciting future directions such as observing the the effect of `Bridge` blocks during pretraining or mid-training, post-training with a Pass@k (Chen et al., 2025) or another global objective, and quantifying the benefit on other modalities and tasks which can exhibit different levels of output homogeneity (Jain et al., 2025). Such directions will push parallel scaling as a much more effective axis of LLM inference scaling.

Up until now, this thesis has been focused on evaluating performance and efficiency on language-based tasks, particularly for LLMs. While LLMs have been central to much of the progress in AI recently, other atypical domains can also benefit from inference optimizations. Next, in Chapter 7, we will study physical dynamics and propose models for scalable machine learning in a materials science application.

Chapter 7

Application: Scalable AI Methods for Materials Science

Just as vanilla LLMs can be inefficient, so too are models that work with modalities and domains beyond language. This chapter briefly covers our line of work on carrying the lessons and observations in LLMs over to machine learning for materials characterization, the study of a material's properties, which comes with its unique challenges and constraints. In particular, we will be improving data collection speed and robustness of electron backscattered diffraction (EBSD) microscopy images. This chapter is based on work introduced in Dong et al. (2023b) and Dong et al. (2026c) while using results in Dong et al. (2023c) and Efimov et al. (2025). *For this chapter only, we use the term "transformers" to describe encoder transformers for brevity.*

7.1 EBSD Microscopy Primer: Utility and Challenges

Experimental methodologies for 3-D tomography of the internal microstructure of materials has been refined considerably in the past few decades (Calcagnotto et al., 2010; Chapman et al., 2021b; Jolley et al., 2021; Kotula et al., 2006; Nguyen and Rowenhorst, 2021; Polonsky et al., 2019, 2022; Uchic et al., 2016), growing to include a variety of modalities such as X-ray computed tomography, optical imaging, electron imaging, energy dispersive X-ray

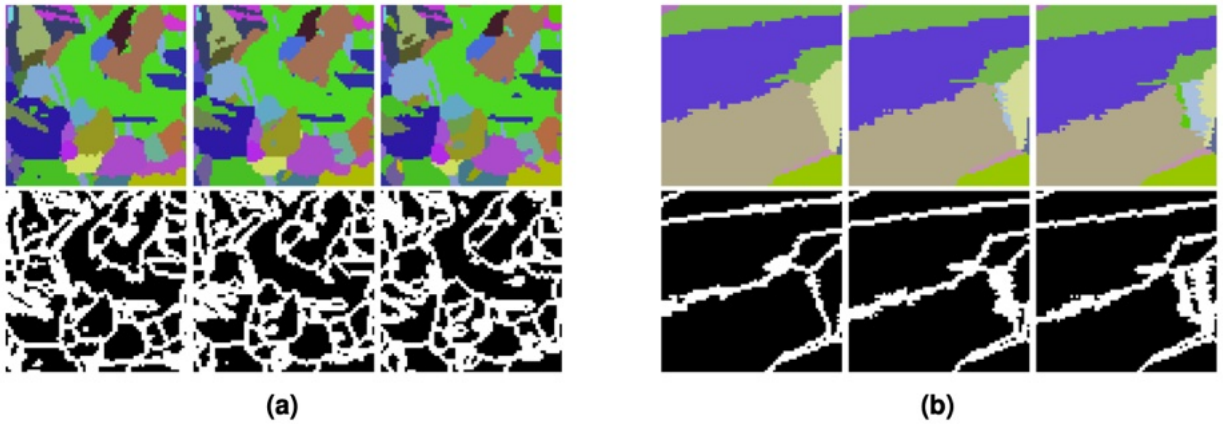

Figure 7.1: Three consecutive crops of real EBSD slices from IN625 (Chapman et al., 2021b) (left) and IN718 (Stinville et al., 2022b) (right). The top row shows the cubochoric values that are scaled and shifted for better visualization, and the bottom row shows the same region in `Boundaries`. Note that the height and width of each slice here is 64 voxels and not the original shape.

spectroscopy, and EBSD data, among others. In particular, EBSD microscopy is an important tool for many materials science applications, as they can provide unique insight into the topology of microstructural features like grains, pores, precipitates and the true shape and size distributions of such features. Properties such as fatigue and oxide transport are sensitive to the 3-D arrangement of these features (Naragani et al., 2017; Sandgren et al., 2016; Wilson et al., 2006). Novel manufacturing processes like additive manufacturing also demand three dimensional characterization of microstructure to fully link the processing to the structure (Polonsky et al., 2022; Teferra and Rowenhorst, 2021).

Despite its many use cases, EBSD can be arduous to collect. For a volume of material, EBSD data is collected with a raster scan of each slice, such that the result is a volume of data where each voxel contains a vector of orientation information in the form of Euler angles. Example EBSD slices are depicted in Figure 7.1. In addition to it being time consuming and expensive, EBSD collection can also be corrupted due to a number of reasons including removing more material (via mechanical polishing or laser/ion ablation) than planned, the electron source nearing the end of its life resulting in low signal images, the magnification on the microscope being incorrect for a slice, or the brightness/contrast settings result in an over/under saturated image. While adjustments to the control software

can be made to prevent these and other issues, it is hard to a priori imagine every reason a slice or subset of data can be corrupted. *Critically, since each slice is sanded off in between each raster scan, the material is essentially destroyed, so recollecting data is not an option. This provides a strong incentive to accelerate, recover, or even bypass EBSD data collection either with partial observations or cheaper modalities using computational methods.*

However, we face challenges that prevent direct application of modern machine learning methods. **1)** First is data scarcity and quality. EBSD volumes can be few and noisy, posing a barrier to common data-hungry machine learning methods. **2)** Second, EBSD data, expressed as Euler angles, are discontinuous and lie in non-Euclidean space, rendering many distance metrics unintuitive. **3)** Third, method outputs should ideally obey physical constraints which may not be reflected in popular quantitative evaluation metrics. While this new domain presents unique challenges, it also come ripe with unique patterns and structures that can be leveraged. Once again, we look at the various AI inference facets from Figure 1.1 and find hope. In particular, we can overcome these challenges in with tools for and observations of EBSD data:

1. **Synthetic and Augmented Data:** Synthetic data generation tools like DREAM.3D (Groeber and Jackson, 2014) and EMsoftOO (Graef, 2024) can produce vast amounts of training data that mimic the distribution of real data. EBSD structure also lends itself to data augmentation methods that may be nonsensical for natural images and videos.
2. **Geometric Awareness:** With a transformation into cubochoric coordinates (Roşca et al., 2014), we can use Euclidean distance metrics to estimate differences between quaternion representations of EBSD data.
3. **In-domain Metrics and Validation:** In conjunction with machine learning metrics, we also use in-domain metrics that translate to physically tangible notions of error like disorientation (Larsen and Schmidt, 2017; Varley, 2024). Moreover, we also rely on qualitative evaluations by materials scientists.

7.2 Organization & Contributions Overview

Adhering to the inference principles of Figure 1.1, we improve the EBSD collection process using deep learning with contributions outlined below:

1. **Efficient Transformer for Missing and Corrupted EBSD Slices** (Section 7.3): Inspired by the observation that EBSD has sparse and low-rank structure (Dong et al., 2023c), we design a tailored two-stage efficient encoder transformer for tensors by leveraging sparse slice-to-slice evolutions (Dong et al., 2023b). In turn, we can reduce the need to collect 25% of EBSD slices with a relatively small 30M parameter transformer.
2. **Multimodal Diffusion for Low-resolution EBSD** (Section 7.4): We can also use the help of a cheaper modality (e.g., polarized light images) with shared information with EBSD to reduce the need for complete EBSD collection. With a multimodal diffusion model and bringing in the notion of parallel inference scaling, we are able to accurately recover EBSD features from low-resolution EBSD observations with information from a cheaper modality (Dong et al., 2026c).

7.3 Efficient Transformers for EBSD Data Recovery

While advancements have been made in robustness and closed loop collection of EBSD data (Chapman et al., 2021b), corruptions in data can still happen. This may not be as significant of a problem in non-destructive techniques where the data can be recollected, but in destructive techniques like serial sectioning (e.g., EBSD), it may often be the case that a few slices out of a thousand have no data or are of very poor quality, lowering the accuracy of the reconstruction.

However, if most of the data was collected properly, we hypothesize it should be possible to infer a substantial amount of the missing data. The current common practice is to fill in missing slices is to just copy the layer above or below the missing slice (Chapman et al., 2021b). This is reasonable in most serial sectioning cases where the slices are being collected at a high enough frequency that most of the data stays the same from slice to slice. Still,

there is some room for improvement on nearest neighbor replacement type approaches, and the transformer model, which has been used to fill in text data (Devlin et al., 2019), is a tantalizing framework for also filling in missing data in sequential image data.

Previous chapters have shown their wide utility in language tasks, but transformers have spread across multiple domains, such as in computer vision (Dosovitskiy et al., 2021), speech processing (Latif et al., 2023), and bioinformatics (Zhang et al., 2023a). In our case, transformers are appealing to EBSD data because the readings are inherently sequential as it represents a real physical structure. Taking inspiration from how encoder transformers are trained, our training procedure involves randomly masking a slice in an EBSD volume and having the model predict the masked slice.

Our slice recovery task is closely related to that of masked image modeling, masked autoencoders, and inpainting in computer vision (Chang et al., 2019; He et al., 2022; Kong et al., 2023; Liu et al., 2021a; Pathak et al., 2016). Even though these may be applicable and there exists plenty of transformers in computer vision (Dosovitskiy et al., 2021; Khan et al., 2022), we want our model to also be computationally efficient as we scale our method and leverage the fact that EBSD volumes have sparse structures (Dong et al., 2023c), the dynamics of which operate differently from typical images or videos. Regarding efficiency, the fact that EBSD produces high dimensional data means the final model's computational footprint cannot be ignored. Fortunately, since we observe compressible structure with EBSD data, we opt for a simplified attention mechanism for high-order tensor data. In summary, our transformer for EBSD recovery exhibits the following characteristics:

1. **Domain Adaptation:** The piecewise constant nature of EBSD data motivates us to use an axial transformer (Ho et al., 2019) for efficient training and inference for tensor data. The benefit of our method is accentuated when zoning in on the recovery accuracy of grain boundaries where other methods appear to perform poorly.
2. **Transfer to Real Data:** We demonstrate that despite being trained solely on synthetic data, our transformer can generalize to real EBSD data *without* additional training while still outperforming all the baselines. This robustness to out-of-distribution EBSD data overcomes a major limitation of relatively low amount of

available real EBSD data that transformer training requires.

3. **Accelerated Collection:** Our results suggest that serial sectioning experiments using similar experimental parameters to those in our test datasets, collection time can effectively be slashed by up to 25% with very little error in predicted voxel orientation, as the collection of every fourth slice can be bypassed. With the current lengthy procedure of EBSD, this is a significant improvement in efficiency.

Next up in Section 7.3.1, we cover efficient encoder attention methods with an emphasis on axial attention, which is designed for high order tensors. Then, we introduce our method in Section 7.3.2 which shows strong results against all baselines in Section 7.3.3. Finally, we describe some open problems and limitations of current methods in Section 7.3.4.

7.3.1 Background on Encoder Attention Methods

Full attention in encoder transformers (2.11) suffer from quadratic scaling of computation with respect to the sequence length, just like attention in decoders (2.4). Consequently, a variety of subquadratic methods have been developed in an attempt to address this (Bolya et al., 2022; Chen et al., 2021; Choromanski et al., 2020; Tay et al., 2022; Wang et al., 2020b; Xiong et al., 2021). Although, these focus primarily on text (1-D inputs) and images (2-D inputs).

When it comes to high-order tensor data, such as EBSD volumes, the sequence length explodes if we flatten (or patchify in the case of many transformers in computer vision (Khan et al., 2022)) the input and apply vanilla attention, making computation a serious bottleneck. Furthermore, since we observe a lot of structure with EBSD data, it is reasonable to utilize some simplified attention mechanism. Our model uses axial attention (Ho et al., 2019; Wang et al., 2020a) which runs the self-attention mechanism along the axes of the input tensor. For instance, in a cube, each voxel only attends to voxels in the same row, column, or depth. This greatly reduces the amount of computation and memory, especially for higher order tensors. Intuitively, axial attention is appropriate because we hypothesize that since local information can be quite uniform (nearby voxels are likely to be in the same grain), long range information should also be included. With axial attention, it is

highly likely to also obtain information in other grains and its boundaries.

In terms of implementation, the formula for multi-headed attention can be reused. Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{N_1 \times \cdots \times N_K \times D}$ be a single K-th order input where $N_1 \times \cdots \times N_K$ defines the shape of the volume and D is the embedding size. As an example, suppose we are interested in finding axial attention along the k-th axis for $1 < k < K$. With proper axis permutation and flattening, we can reshape $\boldsymbol{\mathcal{X}}$ to a tensor of shape $(\prod_{i \neq k} N_i \times N_k \times D)$ and compute multi-headed attention by treating the first axis as the batch axis. In our model, we repeatedly use the outputs of axial attention to compute axial attention along the next axis.

7.3.2 Method

We propose a transformer model to learn missing slices of EBSD data, followed by a projection step to smooth out the voxel values. Described in more detail in the following sections, our methodology is summarized in Figure 7.2. Due to limited real EBSD data, our goal is to train this model on a large and diverse synthetic dataset and demonstrate that it can generalize to real EBSD data, which we evaluate on two nickel superalloy EBSD volumes, one for alloy IN625 (Menasche et al., 2021; Shade et al., 2019) and one for alloy IN718 (Stinville et al., 2022a,b).

Data Description and Preparation

Each dataset (both synthetic and experimental) includes orientation information at every voxel in a 3-D volume. For the experimental data, a substatial amount of preprocessing was done to handle the alignment of the data and clean up noise; a complete description of the preprocessing may be found in Chapman et al. (2021b) and Stinville et al. (2022b). In particular, we remove any grains smaller than 27 voxels (3^3) and average orientations per grain. While operating on grain-average orientations simplifies the orientation prediction problem, it does not represent the real complexity of orientation fields, which often exhibit subtle local variations. We choose to focus on grain-averaged orientations in this first implementation primarily to simplify our initial interpretation of the

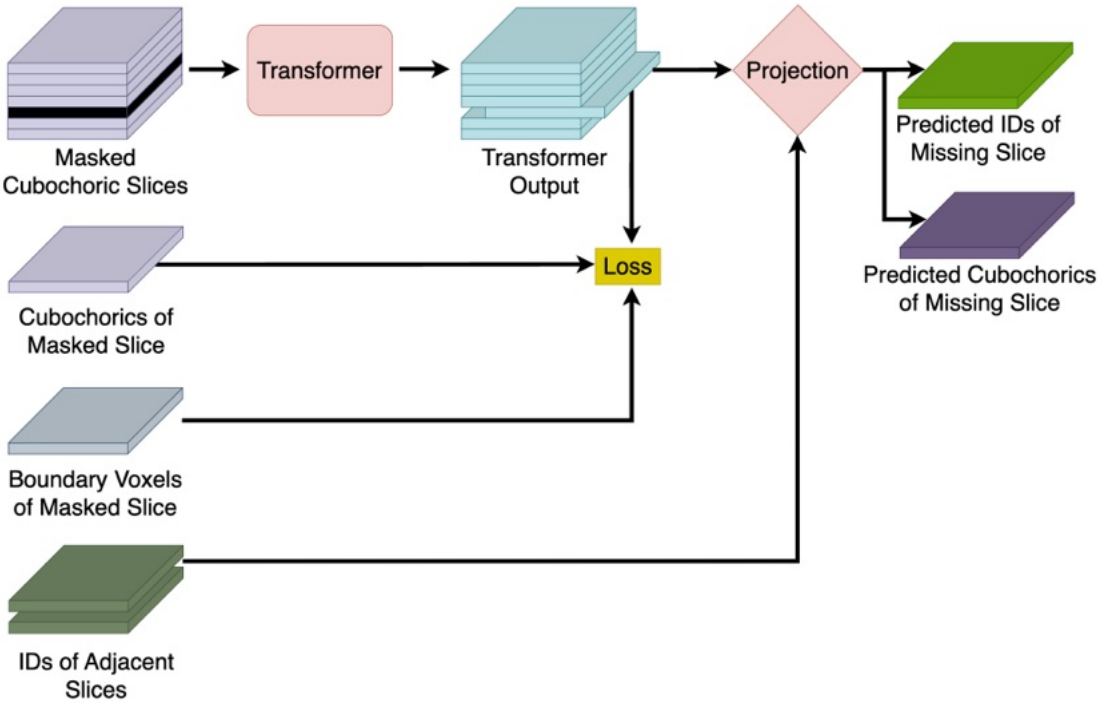

Figure 7.2: Method overview. The transformer takes in a sequence of cubochoric EBSD slices with one unobserved slice as input to produce an output of identical shape where the index of the unobserved slice contains the orientation predictions along the grain boundaries which are the sole contributors to the loss function. The output is then processed by a projection step that assigns each voxel to a grain based on its predicted orientation and its neighboring voxels.

transformer predictions. Additionally, the publicly available version of the IN625 dataset only contains grain-averaged orientations. The original volume containing Euler angles is of shape $N_1 \times N_2 \times N_3 \times 3$, where N_1, N_2, N_3 are the physical axes, and the final axis represents the 3 Euler angles needed to define an orientation. Additional artifacts are then computed from the input data:

- `Cubochorics`: A volume of shape $N_1 \times N_2 \times N_3 \times 3$ where the last axis contains the cubochoric coordinates (Roşca et al., 2014) converted from the original Euler angles at each voxel. Cubochoric coordinates are chosen since the Euclidean metric is used for regressing the transformer model. The Euclidean distance between points in Euler angle space does not necessarily relate to the angular distance between those points. While this is also true for points in cubochoric space, as the cubochoric representation is an equal-volume mapping of $SO(3)$ onto a grid as opposed to an equal-angle

mapping, the Euclidean distance in cubochoric space approximates the Euclidean distance between unit quaternions for small misorientations (Polonsky et al., 2020). Since Euclidean distance is a valid metric in $SO(3)$ (Huynh, 2009), we operate completely in cubochoric space, under the assumption that the Euclidean metric is a reasonable approximation for similarity between points in this space.

- `IDs`: A volume of shape $N_1 \times N_2 \times N_3$ which assigns identification numbers (IDs) to denote which grain each voxel belongs to. Each ID number has a unique vector of cubochoric coordinates associated with it. While not needed during training, these ID numbers will be used to smooth our model outputs and evaluate model accuracy. For experimental data, the IDs are found by segmenting the grains using a misorientation tolerance (Chapman et al., 2021b; Stinville et al., 2022b), and for synthetic data, the IDs are generated alongside the orientation data.
- `Boundaries`: A Boolean volume of shape $N_1 \times N_2 \times N_3$ indicating if a voxel is on the grain boundary. A voxel is a boundary voxel is at least one of its face-sharing neighbors is of a different ID number than itself.

To illustrate, Figure 7.1 contains example slices of (scaled and shifted) `Cubochorics` and `Boundaries` of IN625 and IN718.

Synthetic volumes are generated via DREAM.3D (Groeber and Jackson, 2014). Each synthetic training and validation volume is of physical shape $192 \times 192 \times 192$ and $64 \times 192 \times 64$, respectively. Within DREAM.3D, we generate 9 training volumes while independently varying the mean grain size and mean transformations per grain. In particular, we generate volumes with mean grain sizes 2, 2.5, and 3 with no twins; we also generate volumes with mean twins frequencies 0, 1, 2, 3, 4, and 5 while fixing mean grain size to be 2.3. Due to the nature of the software, these parameters are unitless, as we can always synthetically increase or decrease the granularity of the volume. For example slices of these synthetic volumes, see Figure 7.3. The grain size distributions for each dataset are shown in Figure 7.4 as probability plots. The natural logarithm of the normalized grain sizes are shown; ideal lognormally distributed data would lie on straight lines in such plots. In general, the grain sizes are primarily lognormal near their means, with noted deviations from lognormality

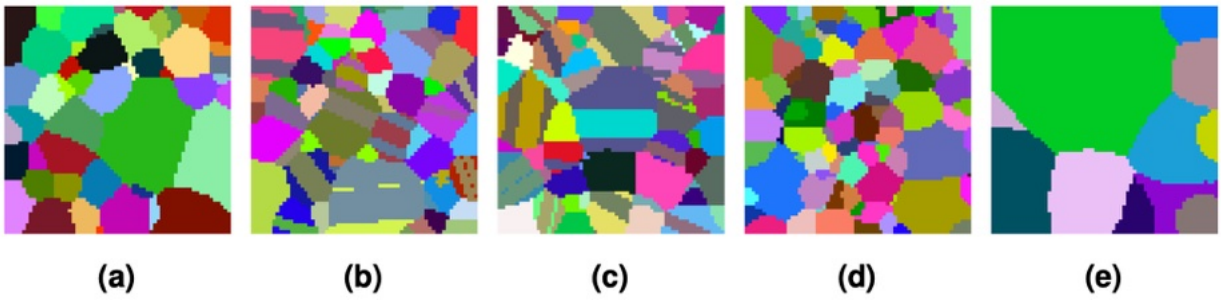

Figure 7.3: Example synthetic slices of cubochoric values that are scaled and shifted for visualization. Keeping mean grain size fixed, Figures 7.3a, 7.3b, and 7.3c show generated slices when we specify the mean frequency of twins per grain to be 0, 2, and 4, respectively. With no twins, Figures 7.3d and 7.3e show generated slices when we specify the mean grain size to be 2 and 3, respectively. All slices have a height and width of 64 voxels.

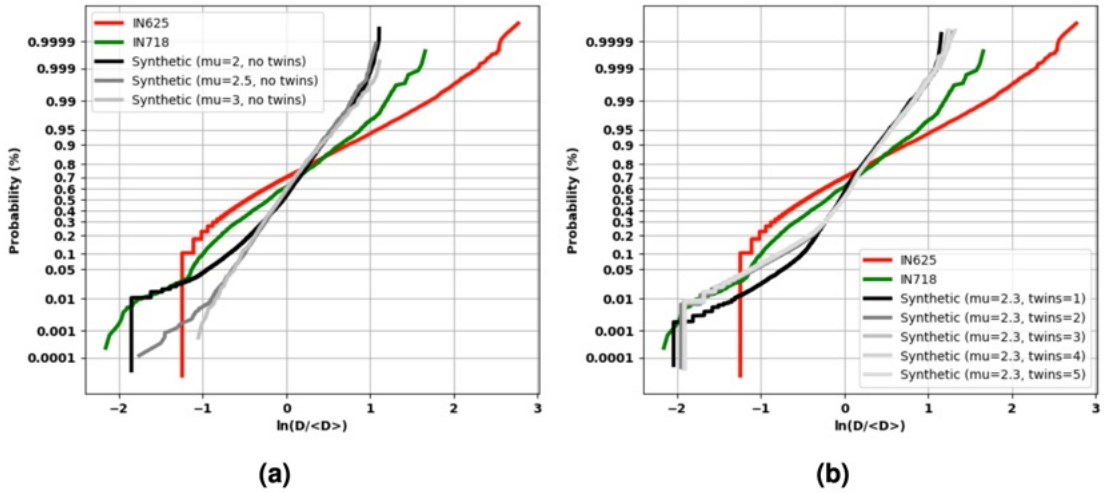

Figure 7.4: Probability plots showing the grain size distributions for each dataset. Figure 7.4a compares the real test datasets to the synthetic training datasets without twins, while Figure 7.4b compares the real test datasets to the synthetic training datasets with twins. Grain sizes are represented as sphere equivalent diameters, D, normalized by the distribution mean.

in their tails, which is a known phenomenon (Donegan et al., 2013).

Training Details

We first describe a few data augmentation steps. The number of unique cubochoric coordinates is quite limited, so color shift transformations, along with other augmentations, are critical. *Note that use of the word "color" is loose here because we treat the cubochoric co-*

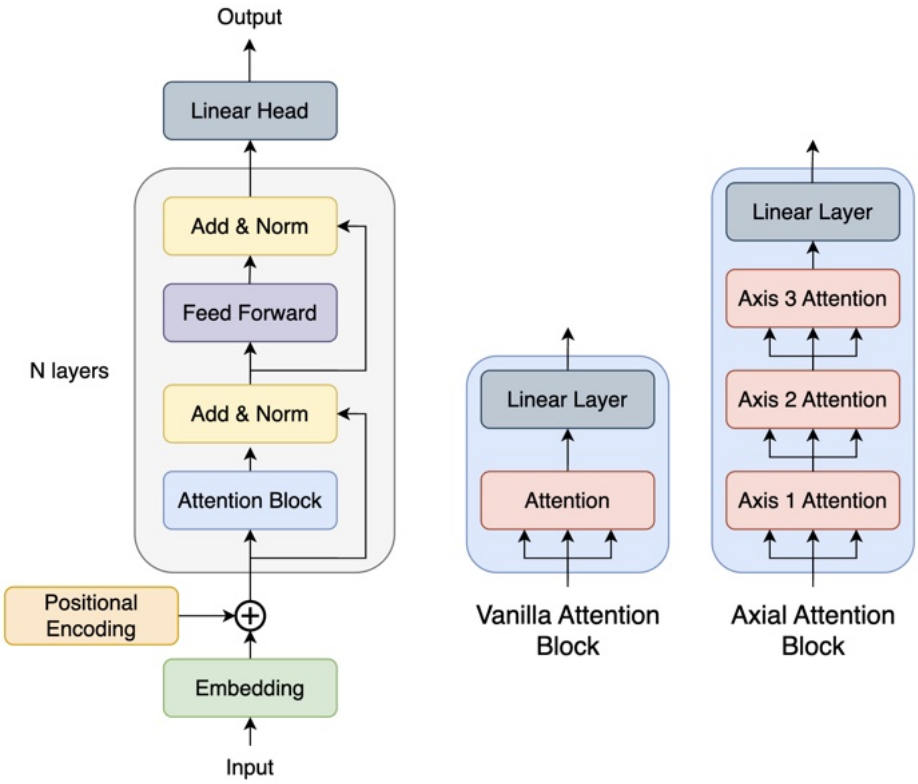

Figure 7.5: Transformer architecture (left) with either vanilla attention (center) or axial attention (right). The axial transformer is produced by simply substituting full attention with axial attention. Although axial attention has significantly more layers, it scales much more favorably for high-order tensor data. Attention layers take three inputs for the query, key, and value.

ordinates the same as color channels in computer vision. In particular, our augmentations include random linear color shifts, rotations, and flips.

Using the 9 synthetic volumes generated by DREAM.3D, we train our axial transformer model in a self-supervised fashion similar to masked language modeling tasks (Devlin et al., 2019). These volumes are first normalized along each of the three cubochoric indices. Each training input is sampled from a randomly chosen volume with physical axes randomly permuted. Not only is cropping necessary due to computational limits, it also acts as another form of augmentation. For a sample $\boldsymbol{\mathcal{X}}_\star \in \mathbb{R}^{64\times 7\times 64\times 3}$, one of the central 5 slices (along the second axis) is randomly masked. If m is the index of the masked or unobserved slice, define $\boldsymbol{\mathcal{M}} \in [0,1]^{64\times 7\times 64\times 3}$ to be a mask such that $[\boldsymbol{\mathcal{M}}]_{:,m} = 0$ and 1 elsewhere. Then, the model input and output will be $\boldsymbol{\mathcal{X}}_\star \odot \boldsymbol{\mathcal{M}}$ and $\widehat{\boldsymbol{\mathcal{X}}} \in \mathbb{R}^{64\times 7\times 64\times 3}$, respectively.

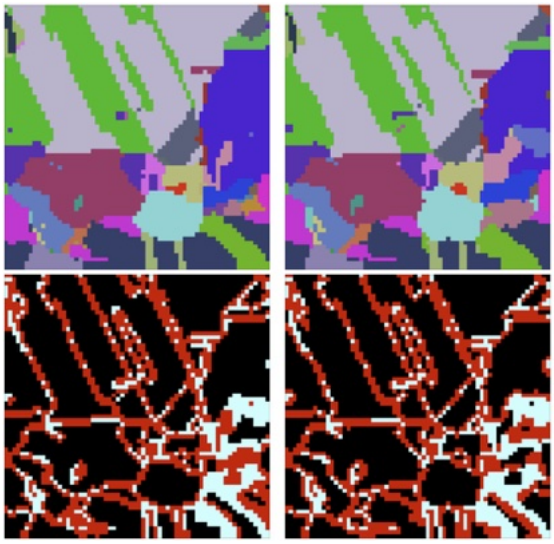

Figure 7.6: Changes between slices are captured in boundary voxels. The top row shows two consecutive slices. The bottom row shows their respective slices in `Boundaries` (red) with the ID difference between the two slices overlaid (white). Note that differences between these slices lie fully in the boundary voxels of both slices.

Since EBSD produces very structured data, we can leverage this to design a more effective loss function. A simple mean-squared-error (MSE) loss function across all voxels would not be sufficient because most voxels in the input are observed, so we would risk learning an identity function which would produce a fairly low loss value. A better approach would be to find the MSE of only the missing slice, analogous to masked language modeling objective functions (Devlin et al., 2019). However, based on Figure 7.1, we see that many of voxels remain unchanged across multiple slices, and the slice-to-slice changes all lie along the grain boundaries (see Figure 7.6). Since voxels within grains are easy to predict the values of, the source of difficulty is recovering the boundary voxels. Therefore, we opt for a MSE loss function that only considers boundary voxels in the missing slice. More formally, letting k be the index of the missing slice and $\boldsymbol{\mathcal{B}} \in [0,1]^{64\times 64\times 1}$ be the missing slice's boundaries map in `Boundaries`, the loss function we use is

$$\mathcal{L}(\widehat{\boldsymbol{\mathcal{X}}}, \boldsymbol{\mathcal{X}}_\star) = \frac{\left\| [\widehat{\boldsymbol{\mathcal{X}}} - \boldsymbol{\mathcal{X}}_\star]_{\cdot,m} \odot \boldsymbol{\mathcal{B}} \right\|_{\mathsf{F}}^2}{\|\boldsymbol{\mathcal{B}}\|_0}, \tag{7.1}$$

following broadcasting rules. By the way we defined $\boldsymbol{\mathcal{B}}$, $\|\boldsymbol{\mathcal{B}}\|_0$ is the number of unobserved

boundary voxels. In summary, this loss function essentially averages MSE error across all unobserved boundary voxels.

Using this scheme, we train our 8-headed 8-layer model using stochastic gradient descent with a momentum parameter of 0.9 and weight decay of 1e−5. Using cosine schedules, we warm-up the learning rate up to 0.01 for 8000 steps. While we decay the learning rate until 160000 total gradient steps are taken with batches of 1 sample, performance plateaus around halfway. The model uses learnable positional encoding, 4 attention heads, an embedding size of 128, and a feedforward size of 512. The feedforward block is consists of two 3-D convolutions with a window size of 3 along each axis separated by a GELU. See Figure 7.5 for a visualization of the architecture. Notably, our model is compact for a transformer, only consisting of slightly under 30 million parameters. We also apply 10% dropout. Recall that the transformer returns predicted cubochoric coordinates at each voxel of the same shape as the input, but only the masked slice contributes to the loss function (7.1).

Nearest Neighbor Projection

The final step is to use outputs of our transformer, which are continuous values, to assign each voxel to a grain and produce a smoother slice with fewer intra-grain variations. To do this, we prepare a dictionary relating observed grain IDs to cubochoric coordinates. First, we assign voxels whose neighbors (voxels that share a face) that reside in the previous and next slice have the same ID to be that ID. We observe these voxels act like anchors that provide more neighbors for voxels that are more difficult to classify, which empirically improves the recovery. Next, we begin projecting voxels, prioritizing the ones with the most neighbors that have been assigned an ID, which include observed voxels and previously projected voxels. This way, we first project voxels with the most known information in their neighborhoods first to gradually build up information in more obscure neighborhoods. Projections are determined by the minimum ℓ_2 distance to relevant cubochoric coordinates pulled from the dictionary based on neighboring IDs. Summarized in Figure 7.7, the projection algorithm essentially turns the transformer outputs of the missing slice into IDs

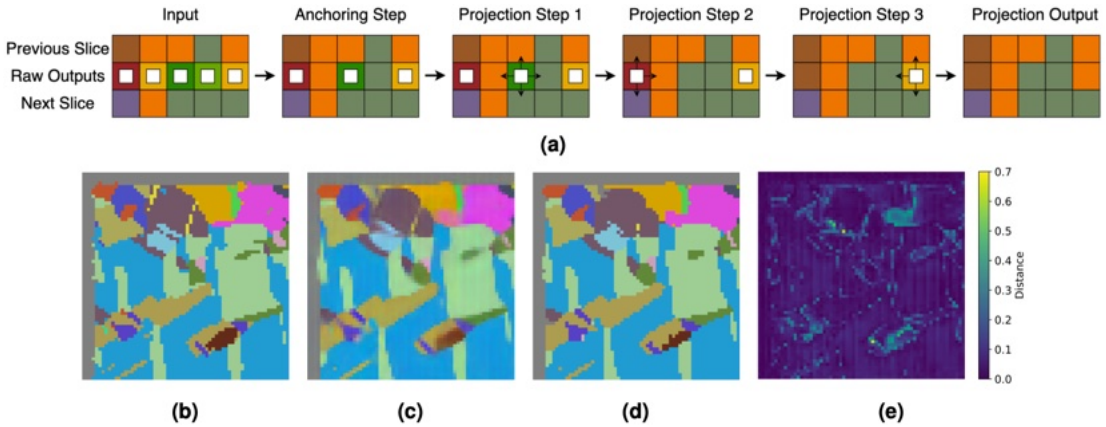

Figure 7.7: The nearest neighbors projection processes the outputs of a transformer to assign voxels to grains. A 2-D toy example is shown in Figure 7.7a. White boxes denote which voxels have not been projected yet. The input contains the transformer output sandwiched between observed adjacent slices. First, the anchoring step assigns voxels whose neighbors in adjacent slices are from the same grain. Then, voxels are sequentially projected to their neighbors, as indicated by the small arrows, starting with the voxels with the most observed and previously projected neighboring voxels. The bottom row showcases a real projection example from the IN625 dataset. Figures 7.7b, 7.7c, and 7.7d show the target slice, the transformer prediction, and the processed output via projection, respectively. Figure 7.7e captures the voxel-wise ℓ_2 difference between before and after projection.

which can then be converted into cubochoric coordinates or Euler angles. For an example, see the bottom row of Figure 7.7.

7.3.3 Core Experiments

With our method defined, we now evaluate its performance on synthetic and real EBSD data with no additional training or fine tuning. Even though our method overwhelmingly outperforms the baselines in terms of recovery, we also point out some weaknesses and avenues for improvement.

There are three baselines we compare to. One is k-nearest neighbors (KNN) where a voxel's ID is determined by vote based only on observed or previously assigned IDs among its neighboring voxels. This is the exact process of the projection step (including the anchoring procedure), except instead of using a distance metric, voxels are assigned IDs by a vote system. Ties are broken randomly. Another method, which is currently employed

as a simple solution to missing slices, is to copy an adjacent slice to replace the missing slice. This usually maintains fairly decent accuracy since the changes from slice to slice are minuscule compared to the number of voxels. As such, copying the previous and next slice are our other two baselines.

Metrics

Because the changes from slice to slice are captured by boundary voxels, we define two different accuracy metrics using the IDs. We denote the accuracy of each voxel in the recovered slice as the *overall accuracy* and the accuracy only considering boundary voxels in the recovered slice as the *boundary accuracy*. The latter poses a greater challenge for all models, as it is consistently lower than the overall accuracy where the influence of non-boundary voxels often dominates.

While we report the mean accuracies and standard deviation across validation samples, the performance of one method is heavily correlated with the performance of others. For instance, if a particular slice is difficult to recover for one method, it is likely to be difficult for all other methods. To better compare the performances between our transformer and each baseline, we obtain differences in accuracy for each sample. Namely, we find $d(m, \boldsymbol{\mathcal{X}}_\star, \widehat{\boldsymbol{\mathcal{X}}}, \widehat{\boldsymbol{\mathcal{X}}}_b) := m(\boldsymbol{\mathcal{X}}_\star, \widehat{\boldsymbol{\mathcal{X}}}) - m(\boldsymbol{\mathcal{X}}_\star, \widehat{\boldsymbol{\mathcal{X}}}_b)$ for an accuracy metric m, ground truth $\boldsymbol{\mathcal{X}}_\star$, and outputs, $\widehat{\boldsymbol{\mathcal{X}}}$ and $\widehat{\boldsymbol{\mathcal{X}}}_b$, from our transformer and baseline b, respectively. Computing this across all samples, the result is a distribution of *accuracy improvements*.

Synthetic Volumes

First, we generate 4 independent synthetic volumes of shape $(64, 192, 64)$ for each setting. Each volume is divided into 27 nonoverlapping segments of shape $(64, 7, 64)$ each with one of the five interior slices masked. This results in 108 validation samples for each setting. These validation metrics are displayed in Figure 7.8 when varying the mean twins frequency and in Figure 7.9 when varying mean grain size.

We observe that our method more accurately recovers missing slices than all other baselines for every synthetic validation sample since each slice observed a positive improvement

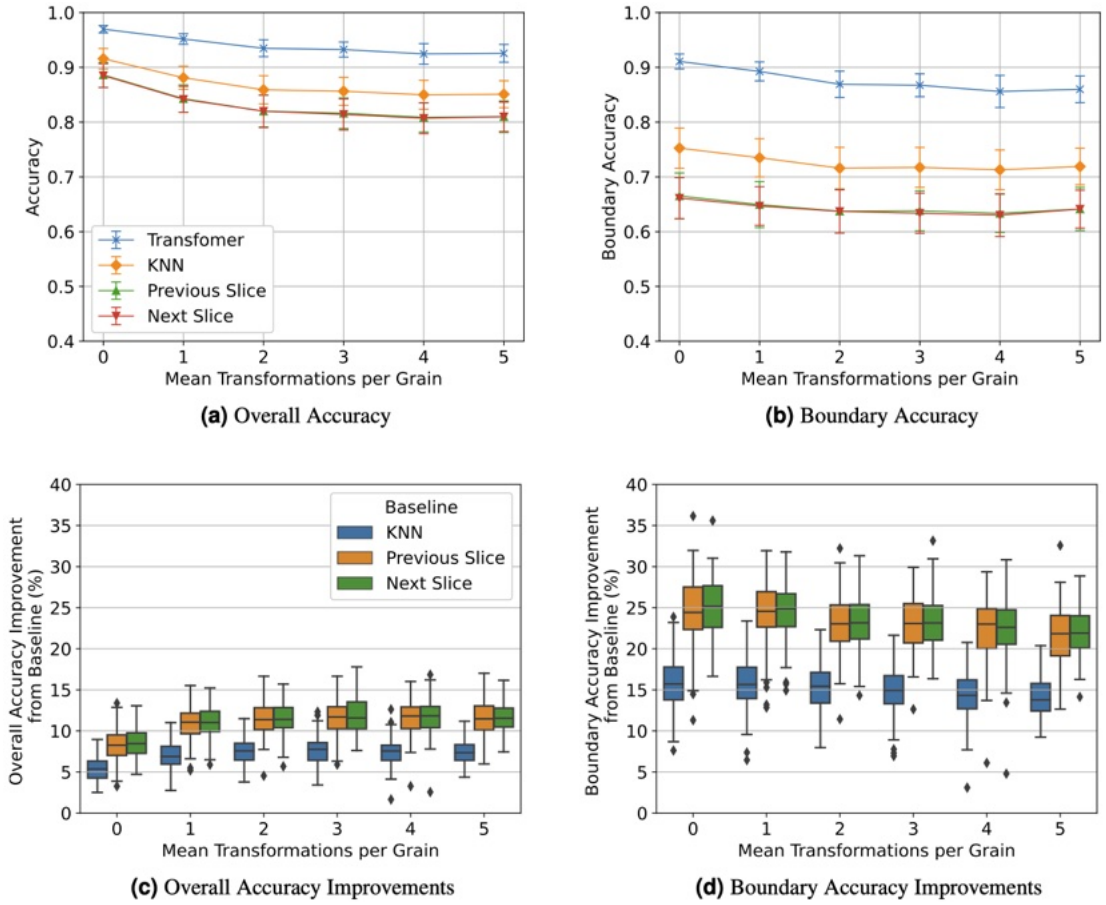

Figure 7.8: Overall accuracy (Figure 7.8a), boundary accuracy (Figure 7.8b), overall accuracy improvements (Figure 7.8c), and boundary accuracy improvements (Figure 7.8d) of synthetic volumes with varying twin frequencies. Error bars in Figures 7.8a and 7.8b represent one standard deviation.

in overall and boundary accuracy. The performance gain is much more apparent for boundary voxels. Furthermore, our transformer performance is much lower variance. Among the baselines, KNN achieves the closest accuracy to our method, but it still underperforms in comparison. As expected, the difference between our method and the baselines diminishes as the scenario get simpler (larger grain sizes and fewer twins) since the baselines are already producing very accurate results.

Real EBSD Datasets

Now, we seek to understand how well our model transfers to real data by running our trained model on IN625 (Menasche et al., 2021; Shade et al., 2019) and IN718 (Stinville

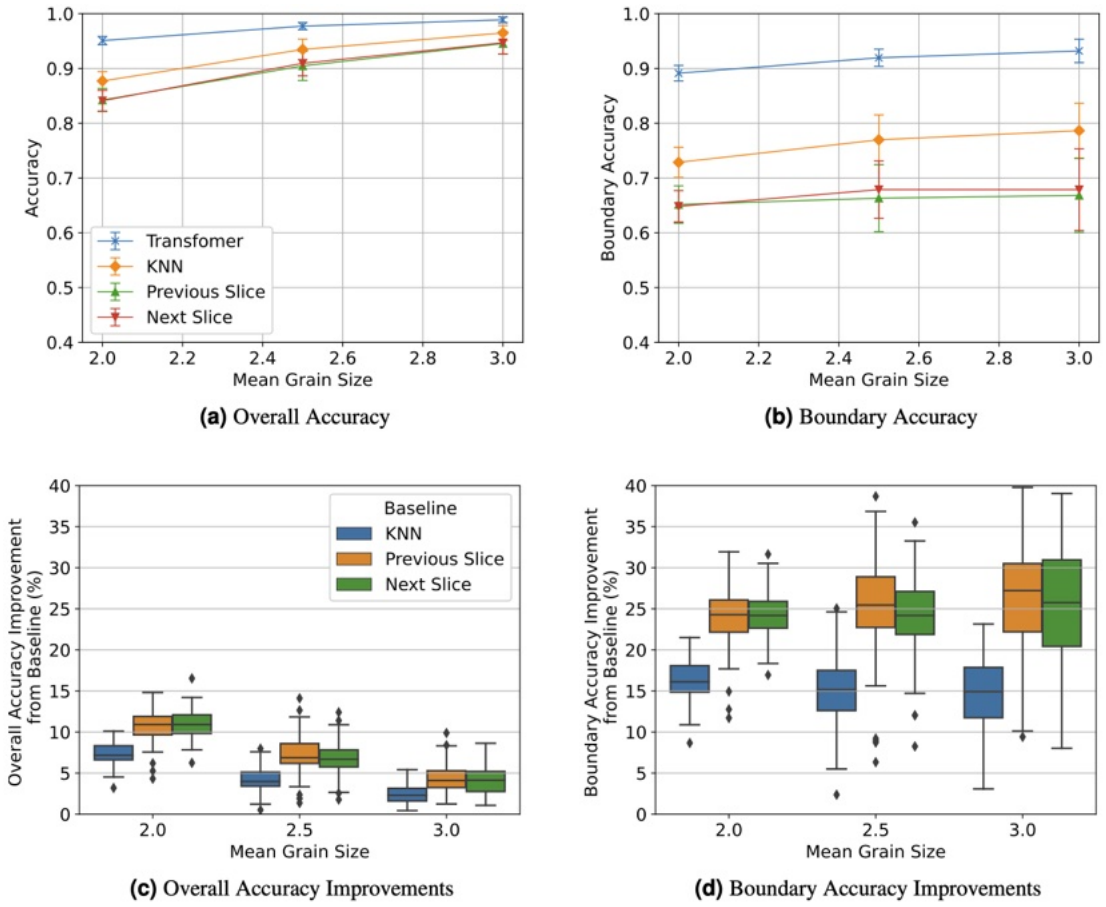

Figure 7.9: Overall accuracy (Figure 7.9a), boundary accuracy (Figure 7.9b), overall accuracy improvements (Figure 7.9c), and boundary accuracy improvements (Figure 7.9d) of synthetic volumes with varying mean grain sizes. Error bars in Figures 7.9a and 7.9b represent one standard deviation.

et al., 2022a,b). For both volumes, we subdivide each volume into nonoverlapping subvolumes such that all slices are of height and width 64 voxels. Then, each subvolume is then further partitioned into nonoverlapping segments of shape $64 \times 7 \times 64$, each representing a single test sample. Again, each sample contains one masked slice among the five central slices. In the end, there are 800 samples for IN625 and 3298 samples for IN718 after discarding a small set of samples whose missing slice did not have any boundary voxels (these samples would trivially result in exact recovery regardless of the model we use). Average accuracies and accuracy improvements for both volumes are shown in Table 7.1 and Figure 7.10, respectively. For recovery examples, see Figure 7.11.

Surprisingly, even though our model is trained exclusively on synthetic data, we observe

Table 7.1: Average overall accuracy and boundary accuracy across samples of real datasets, including their standard deviations.

Dataset	Model	Avg. Acc. (%)	Avg. Boundary Acc. (%)
IN625	Transformer	**91.83 ± 1.98**	**79.49 ± 3.20**
	KNN	89.18 ± 2.48	72.86 ± 3.77
	Previous Slice	86.44 ± 2.94	65.83 ± 4.73
	Next Slice	86.45 ± 3.15	65.91 ± 4.98
IN718	Transformer	**98.27 ± 1.03**	**85.65 ± 3.49**
	KNN	97.54 ± 1.52	79.81 ± 5.49
	Previous Slice	96.40 ± 2.23	70.43 ± 8.50
	Next Slice	96.39 ± 2.22	70.25 ± 8.47

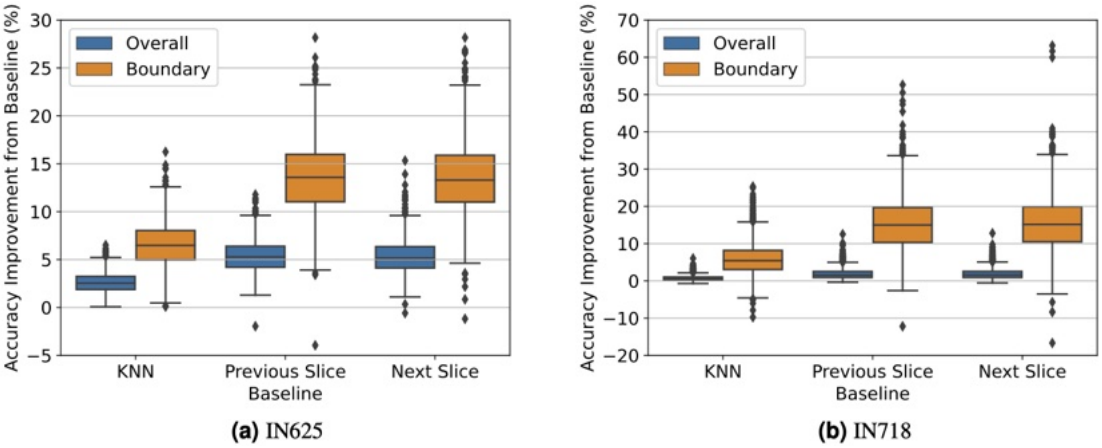

Figure 7.10: Overall and boundary accuracy improvements of IN625 and IN718 test sets. Note that the y-axes are on different scales.

it transfers well to real EBSD data, as it still outperforms every baseline on nearly every sample, and the interquartile range is positive. Again, the difference is accentuated for boundary voxels. Test results for the IN718 dataset had much higher variance, likely due to each slice having proportionally fewer boundary voxels than IN625 slices. Qualitatively, our model has a stronger capability to recovery thin features than KNN, which visually tends to ignore more subtle structures.

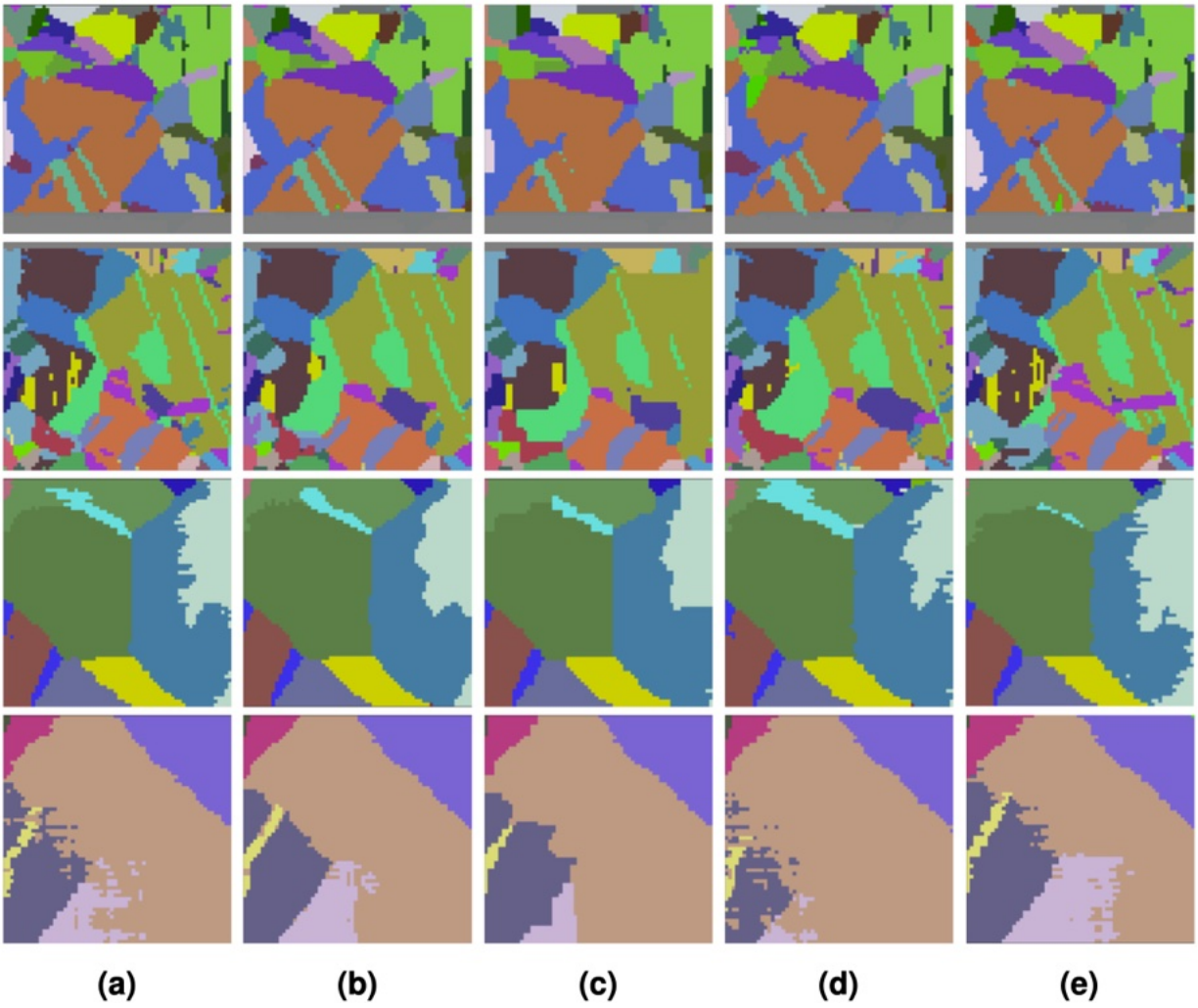

Figure 7.11: Four random example predictions of missing slices from IN625 (top two rows) and IN718 (bottom two rows) test sets . Each row is a separate example. Figure 7.11a contains the target; Figures 7.11b, 7.11c, 7.11d, and 7.11e are predictions made by our transformer, KNN, the previous slice, and the following slice, respectively.

7.3.4 Error Analysis & Limitations

While our method provides superior results compared to the baselines, we also seek to understand the circumstance in which it may underperform. One challenge is rapid changes between slices which make up a small minority of test inputs. For instance, if the k-th slice is missing and the set of grains present in the $(k-1)$-th is different from the set of grains in $(k+1)$-th slice, the model has a lot of freedom to decide on which ones are present and to what degree in the missing slice. A case in which this can arise is when the faces of grains

or twins are perpendicular to the slicing direction. However, we observe this is an issue for all methods. Using the second row of Figure 7.11 as an example, the bright green grain is a large crescent shape on the slice before the missing slice but is hardly present in the following slice. Our model and KNN strives to find some middleground, but ultimately, both misclassifies many of the voxels. A similar scene plays out in the fourth row of Figure 7.11. Thus, future work includes designing a better loss function to mitigate errors for this or emphasizing these scenarios during training.

Another limitation arises from our projection method. Recall our local projection method projects a voxel to have the ID as one of its observed or previously assigned voxels in order to encourage grain connectedness. This means long range dependencies may be ignored in favor of more smooth structures. Furthermore, connectedness is not guaranteed for very thin features at an angle. Examples of both edge cases for matrices are illustrated in Figure 7.12. Though these scenarios are fairly rare in practice, further work could be done to improve our projection algorithm. In particular, our use of grain-averaged orientations directly impacts the design of the projection step. While our transformer predicts real-valued orientations, we utilize a projection step that "snaps" these predicted values to the nearest grain-averaged value to compare with the original training data. Moving to orientation values that are not grain-averaged will require adaptation of the projection step, and potentially require modifications of the underlying transformer architecture.

Future directions can also better integrate the dynamics of orientation data into our method. For example, our use of MSE loss operating on cubochoric values is not a mathematically rigorous metric representation. We initially attempted to utilize an angular metric for loss, but encountered instability during training that we believe related to the trigonometric functions used during the loss computation. An immediate improvement to the current approach would be operating on an orientation representation for which we can compute a stable loss that represents the true angular difference in that representation space. In addition, our current data augmentation method borrowed from computer vision techniques, particularly color shifting, cannot guarantee preservation of misorientation relationships between twins and their parent grains. Taking such additional special

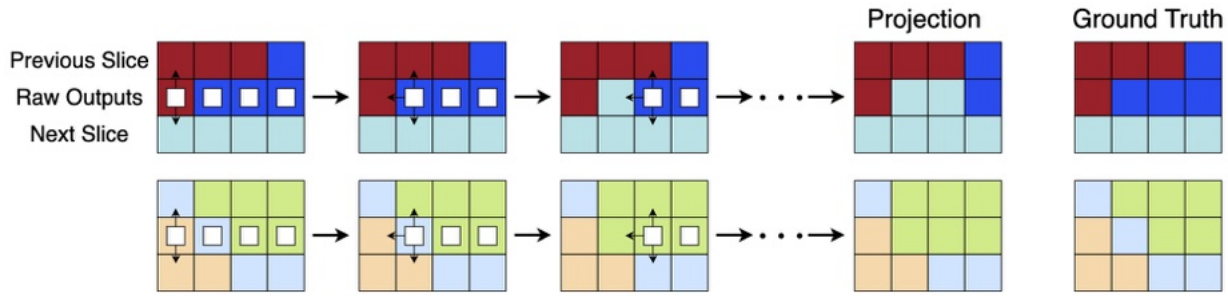

Figure 7.12: Matrix examples where projection to observed and previously projected neighbors is suboptimal, even when giving it the ground truth. Projection involves using the neighboring slices to predict the center slice. White squares indicate which pixels have not been projected yet, and the black arrows point to the possible neighbors that the pixel will choose to project to. The top row shows an example where the projection operation smooths out very thin features that lie completely in the missing slice. The bottom row shows an example where the projection operation may disconnect a grain.

considerations may enhance the performance of our method.

7.3.5 Work Summary

We have presented a novel method using transformers followed by a projection algorithm to recover missing 3-D EBSD data which vastly outperforms all baselines. Notably, even though our model is trained on synthetic data, it still recovers more accurate slices for real 3-D EBSD data than the baselines by a wide margin, making it a powerful data processing tool for faulty 3-D EBSD readings. Furthermore, our model opens the possibility to skip every fourth slice during data collection (by using every skipped slice and the three collected slices on each side as the input to our model), potentially reducing the collection time by 25%. Future work involves addressing the limitations, scaling our method, and considering more general cases, such as altering the projection algorithm to apply to consecutive missing slices within a sample. Beyond EBSD, it would be interesting to investigate our method's applicability for other material science tensor data.

7.4 Scaling Multimodal Diffusion for Sparse EBSD Data

As we saw in the previous section, since each pixel in EBSD orientation maps is a result of a machine indexed full diffraction pattern, this makes serial data acquisition with EBSD a potential bottleneck in 3-D serial sectioning experiments. Much attention has been dedicated to improving the throughput of the technique, both in the form of faster and more sensitive detectors (della Ventura et al., 2026) and better indexing algorithms (Varley et al., 2026). By contrast, polarized light (PL) microscopy is very fast, since it acquires an entire image in parallel, but the technique suffers from the fact that it can only be applied to materials with a non-cubic crystal structure; in optically uniaxial materials like hexagonal close-packed Ti, only the orientation of the crystallographic c-axis can be determined, leaving the orientation in the plane normal to this axis as a degree of freedom. For optically bi-axial materials this limitation does not apply. Nevertheless, it is tempting to consider a data fusion approach between minimal data acquisition with EBSD and the balance with PL microscopy in order to improve the overall throughput of microstructure characterization. This section presents a novel algorithm using generative AI, which applies inverse solvers with diffusion to recover features of an EBSD map, from an input PL image (Dong et al., 2026c). An EBSD map can be useful both for the explicit orientations of each grain in a polycrystalline material and for the simpler problem of just finding the boundaries of each grain. Both of these issues can be addressed from the results of this work.

Generally, *we aim to use fewer EBSD measurements, which are relatively more expensive to collect, in conjunction with a greater quantity of cheaper PL measurements to accomplish downstream objectives that can benefit more from both modalities than either alone.* As we shall see, this is deeply rooted as an inverse problem. More concretely, the aim of solving inverse problems is to recover some underlying ground truth $\boldsymbol{\mathcal{X}}^\star$ from observation $\boldsymbol{\mathcal{Y}}$ with the following relationship:

$$\boldsymbol{\mathcal{Y}} = f(\boldsymbol{\mathcal{X}}^\star) + \boldsymbol{\mathcal{E}}, \tag{7.2}$$

where f is known as the forward model and $\boldsymbol{\mathcal{E}}$ is measurement noise. Relating EBSD data as $\boldsymbol{\mathcal{X}}^\star$ and PL data as $\boldsymbol{\mathcal{Y}}$ in this manner means $f = f_{\mathrm{E}\mapsto\mathrm{P}}$ is a complicated nonlinear and discontinuous function. To make matters worse, the inverse $f^{-1}_{\mathrm{E}\mapsto\mathrm{P}}$ is ill-posed since there is information in EBSD that cannot be captured by PL (Jin and De Graef, 2018, 2020). However, *we hypothesize at the slice level, spatial and macroscopic features in PL data can contain emergent information that is missing in pointwise low quality EBSD observations.*

Diffusion models (Ho et al., 2020; Song et al., 2020) have been established as a highly expressive generative model to capture complex information in the data distribution in recent years. Extensively tested on natural image generation (Peebles and Xie, 2023; Ramesh et al., 2022; Rombach et al., 2022), diffusion models have also shown efficacy as priors to solve inverse problems for natural (Chung et al., 2022; Lugmayr et al., 2022; Xu and Chi, 2024), medical (Islam et al., 2023; Lyu and Wang, 2022; Webber and Reader, 2024), and microscopy images (Efimov et al., 2025, 2026; Saguy et al., 2025). Since diffusion models are learned from the data of interest directly, they are capable of capturing characteristic that are difficult for pre-determined, hand-crafted priors such as sparsity (Baraniuk, 2007; Candès et al., 2006) or total variation (Rudin et al., 1992). Diffusion models also generate images from randomly initialized noise, and repeated inference from different initializations with an inverse solver can lead to different output reconstructions of $\boldsymbol{\mathcal{X}}^\star$ from the same input observation $\boldsymbol{\mathcal{Y}}$. Hence, *inverse solvers with diffusion models have the advantage that instead of producing one reconstruction per observation, they can capture distributions of possible reconstructions via repeated inference.* Akin to Monte Carlo sampling from the posterior distribution, a set of output reconstructions provides measures of uncertainty, which are richer representations than a single predicted reconstruction. This is an analogous application of parallel inference scaling described in Section 2.2.4 and seen in Chapters 5 and 6.

Though very successful in the domain of natural images, several technical challenges arise when we think about diffusion in our setting. One major challenge is that the forward model $f = f_{\mathrm{E}\mapsto\mathrm{P}}$ and its gradients are complex nonlinear functions, limiting the use of many existing diffusion-based inverse problem solvers. To correctly guide the reconstruction with

multimodal information, we reformulate the original problem of reconstructing just one modality from another into joint multimodal reconstruction via a multimodal diffusion model, following Efimov et al. (2025). By jointly learning the probability distribution of all modalities with a multimodal diffusion model, the new forward model becomes just a linear masking operator, f_{EP}. Consequently, the new reconstruction or inverse problem also becomes linear, bypassing the need to explicitly have a forward model between the two modalities. To illustrate the difference between unimodal and multimodal diffusion, let us adapt (7.2) for each case. Define perfect EBSD measurements, observed EBSD measurements, and EBSD measurement noise to be $\boldsymbol{\mathcal{X}}^{\star}_{\mathrm{E}}$, $\boldsymbol{\mathcal{Y}}_{\mathrm{E}}$, and $\boldsymbol{\mathcal{E}}_{\mathrm{E}}$ respectively, and similarly for PL ($\boldsymbol{\mathcal{X}}^{\star}_{\mathrm{P}}$, $\boldsymbol{\mathcal{Y}}_{\mathrm{P}}$, and $\boldsymbol{\mathcal{E}}_{\mathrm{P}}$). The forward model in the unimodal case is

$$\boldsymbol{\mathcal{Y}}_{\mathrm{P}} = f_{\mathrm{E}\mapsto\mathrm{P}}(\boldsymbol{\mathcal{X}}^{\star}_{\mathrm{E}}) + \boldsymbol{\mathcal{E}}_{\mathrm{P}}. \tag{7.3}$$

The multimodal case concatenates the modalities into $\boldsymbol{\mathcal{X}}^{\star}_{\mathrm{EP}} = [\boldsymbol{\mathcal{X}}^{\star}_{\mathrm{E}} \quad \boldsymbol{\mathcal{X}}^{\star}_{\mathrm{P}}]$ for the target, $\boldsymbol{\mathcal{Y}}_{\mathrm{EP}} = [\boldsymbol{\mathcal{Y}}_{\mathrm{E}} \quad \boldsymbol{\mathcal{Y}}_{\mathrm{P}}]$ for the measurement, and $\boldsymbol{\mathcal{E}}_{\mathrm{EP}} = [\boldsymbol{\mathcal{E}}_{\mathrm{E}} \quad \boldsymbol{\mathcal{E}}_{\mathrm{P}}]$ for the noise. Hence, the new multimodal forward model becomes

$$\boldsymbol{\mathcal{Y}}_{\mathrm{EP}} = f_{\mathrm{EP}}(\boldsymbol{\mathcal{X}}^{\star}_{\mathrm{EP}}) + \boldsymbol{\mathcal{E}}_{\mathrm{EP}}, \tag{7.4}$$

where f_{EP} is a masking operation such that we only partially observe EBSD images but fully observe PL images. In practice, f_{EP} can involve intentional choices like subsampling EBSD data to be at a lower resolution for faster data collection. By learning an unconditional multimodal diffusion model over both modalities, we implicitly capture the relationship between EBSD and PL data. *Moreover, by learning an unconditional multimodal diffusion model as a joint prior on both modalities, the synthesis of information between EBSD and PL data can capture more holistic representations of the material than either alone.*

The second challenge is the lack of real EBSD data. This is a major issue as diffusion models are typically trained on a large corpus of data. To overcome this, we enlist the help of diverse synthetic data created by DREAM.3D (Groeber and Jackson, 2014) to train our models, like in Section 7.3. In fact, without touching any real data during training, we

are able to see significant gains in performance on real data. In summary, we propose a general method for various reconstruction tasks involving EBSD and PL data which has the following benefits:

1. **Transfer to Real Data:** Although we only use synthetic EBSD and PL data during training due to scarcity of real data, our method still has strong performance when deployed on real data without any additional training.
2. **Parallel Inference Scaling:** By sampling multiple reconstructions of $\boldsymbol{\mathcal{X}}^\star$ from a single observation $\boldsymbol{\mathcal{Y}}$, we are able to obtain both a distribution of reconstructions and after aggregating, a higher quality objective-specific prediction than a single inference call.
3. **Task Generalizability:** Although we focus primarily on *boundary prediction* in this paper, we show that the same diffusion model and inference pipeline with minor post-processing adjustments can be used for other tasks such as *EBSD super-resolution* and *PL denoising*.

Comprehensive experiments and evaluations show strong efficacy of parallel inference scaling for inverse solvers with unconditional multimodal diffusion for a variety of scenarios. For the rest of this section, we briefly cover relevant background and related works. Afterwards, we outline our training and inference pipelines in Section 7.4.2. In Section 7.4.3, we demonstrate our method's strong performance across a variety of settings where EBSD and PL data fusion is helpful such as boundary prediction, super-resolution, and denoising. Finally, we end with some concluding thoughts in Section 7.4.5.

7.4.1 Background

We provide brief overviews on diffusion models and their use in inverse problems here. While their architectures in practice are often similar to transformers, they generate outputs in a completely different fashion.

Diffusion Models

At a high level, diffusion models (Ho et al., 2020; Song et al., 2020) map pure noise to some data distribution of interest with the probability density function $p(\boldsymbol{x})$. By sampling initializations, one can obtain generated data that ideally fall in the data distribution. The diffusion process is split into the forward process (not to be confused with the forward model f) and the backward process. Here, we let data points be vectors for simplicity, but the same formulations naturally carry over to matrices and tensors.

The forward process incrementally adds Gaussian noise to a sample $\boldsymbol{x}_0 \in \mathbb{R}^D$ from the data distribution p_{data} such that

$$\boldsymbol{x}_t = \sqrt{1-\beta_t}\boldsymbol{x}_{t-1} + \sqrt{\beta_t}\boldsymbol{\epsilon}_t, \tag{7.5}$$

for $1 \leq t \leq T$, noise vectors $\boldsymbol{\epsilon}_t$ drawn i.i.d. from $\mathcal{N}(\mathbf{0}, \boldsymbol{I}_D)$, and learning rates $\beta_t \in (0,1)$ that determine the amount of noise added at each step. In a single step, this reduces to

$$\boldsymbol{x}_t = \sqrt{\overline{\alpha}_t}\,\boldsymbol{x}_0 + \sqrt{1-\overline{\alpha}_t}\,\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_D), \tag{7.6}$$

where

$$\alpha_t := 1 - \beta_t, \quad \overline{\alpha}_t := \prod_{k=1}^{t} \alpha_k. \tag{7.7}$$

Furthermore, for some $\boldsymbol{x} \in \mathbb{R}^D$, the score function is defined as

$$s_t^\star(\boldsymbol{x}) = \nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x}), \tag{7.8}$$

where p_t is the probability density function of $\boldsymbol{x}_t$. Intuitively, the score is the gradient that points in the direction of higher probability images. For the case of denoising from a Gaussian distribution, the score is the direction towards a less noisy image.

The backward process incrementally removes noise, starting with an initial Gaussian

sample $\boldsymbol{x}_T^{\text{rev}}$:

$$\boldsymbol{x}_T^{\text{rev}} \to \boldsymbol{x}_{T-1}^{\text{rev}} \to \cdots \to \boldsymbol{x}_0^{\text{rev}}. \tag{7.9}$$

In practice, we do not have access to $s_t^\star(\boldsymbol{x})$, and learn it by connecting score matching to denoising as described in Vincent (2011). In summary, a denoising network, denoted as $\mathcal{M}_\theta(\boldsymbol{x}, t)$, directly predicts the noise and thus implicitly approximates the score via

$$s_\theta(\boldsymbol{x}_t, t) = -\frac{\mathcal{M}_\theta(\boldsymbol{x}_t, t)}{\sqrt{1-\overline{\alpha}_t}}. \tag{7.10}$$

Substituting into the score matching objective and simplifying yields the *denoising loss* (Ho et al., 2020):

$$\mathcal{L}(\theta) = \mathbb{E}_{t,\,\boldsymbol{x}_0,\,\boldsymbol{\epsilon}}\left[\left\|\boldsymbol{\epsilon} - \mathcal{M}_\theta\big(\sqrt{\overline{\alpha}_t}\,\boldsymbol{x}_0 + \sqrt{1-\overline{\alpha}_t}\,\boldsymbol{\epsilon},\ t\big)\right\|^2\right], \tag{7.11}$$

where the expectation is over $t \sim \text{Uniform}\{1, \ldots, T\}$, $\boldsymbol{x}_0 \sim p_{\text{data}}$, and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_D)$.

Diffusion for Inverse Problems

Unconditional diffusion models provide a powerful prior on the distribution of the data and can be effectively utilized to solve inverse problems without any task-specific training. The inverse solver in that case is then formulated as posterior sampling, where we try to enforce both the data consistency and prior distribution constraints. Recalling the forward model formulation from (7.2), we want to sample $\widehat{\boldsymbol{\mathcal{X}}}$ from

$$p(\widehat{\boldsymbol{\mathcal{X}}} \mid \boldsymbol{\mathcal{Y}}) \propto p^\star(\widehat{\boldsymbol{\mathcal{X}}}) p(\boldsymbol{\mathcal{Y}} \mid \boldsymbol{\mathcal{X}}^\star = \widehat{\boldsymbol{\mathcal{X}}}), \tag{7.12}$$

where $p^\star$ is the prior distribution of $\boldsymbol{\mathcal{X}}^\star$ (captured by the diffusion model), and $p(\boldsymbol{\mathcal{Y}} \mid \boldsymbol{\mathcal{X}}^\star = \widehat{\boldsymbol{\mathcal{X}}})$ is the likelihood of observing $\boldsymbol{\mathcal{Y}}$ assuming the ground truth $\boldsymbol{\mathcal{X}}^\star = \widehat{\boldsymbol{\mathcal{X}}}$ (assumed to be known). In turn, we sample from both prior and likelihood distributions separately to obtain posterior samples.

Several posterior sampling algorithms have been proposed with provable consistency guarantees (Cardoso et al., 2023; Dou and Song, 2024a; Xu and Chi, 2024). In this paper, we use one of them, FPS-SMC (Dou and Song, 2024a), a particle filtering algorithm for linear inverse problems. Recall that even though the forward model f between modalities is not linear in general, the multimodal formulation recasts this problem as inpainting, which is linear (Efimov et al., 2025).

Parallel Inference Scaling for Diffusion

We showed that parallel inference scaling to be an effective way to improve performance in Chapters 5 and 6. We take the same intuition to sample and combine multiple independent reconstructions from the same inverse solver, diffusion model, and observation to improve the final reconstruction quality and stability. Concretely in our setting, parallel inference scaling entails repeated independent sampling from the posterior distribution, $p(\widehat{\boldsymbol{\mathcal{X}}} \mid \boldsymbol{\mathcal{Y}})$ (i.e., sampling multiple reconstructions with the same observation $\boldsymbol{\mathcal{Y}}$ and same inverse solver but with different diffusion noise initializations described in (7.9)).

7.4.2 Method

We train three models: **(1)** a multimodal diffusion model, $\mathcal{M}_{\mathrm{EP}}$, that generates EBSD and PL images; **(2)** a unimodal diffusion model, $\mathcal{M}_{\mathrm{E}}$, that generates EBSD images; **(3)** a unimodal diffusion model, $\mathcal{M}_{\mathrm{P}}$, that generates PL images. We omit θ as used in (7.11) for brevity.

Data Formulation and Preprocessing

The input data takes multiple forms depending on the model. Define H, W, D_{E}, and D_{P} to be the image height, image width, number of EBSD features, and number of PL features, respectively. Letting $\boldsymbol{\mathcal{X}}^{\star}_{\mathrm{E}} \in \mathbb{R}^{H\times W\times D_{\mathrm{E}}}$ and $\boldsymbol{\mathcal{X}}^{\star}_{\mathrm{P}} \in \mathbb{R}^{H\times W\times D_{\mathrm{P}}}$ be the true EBSD and PL data, respectively, we construct $\boldsymbol{\mathcal{X}}^{\star}_{\mathrm{EP}} = [\boldsymbol{\mathcal{X}}^{\star}_{\mathrm{E}} \quad \boldsymbol{\mathcal{X}}^{\star}_{\mathrm{P}}] \in \mathbb{R}^{H\times W\times (D_{\mathrm{E}}+D_{\mathrm{P}})}$ to be the pointwise concatenation of these measurements. By (7.4), we observe $\boldsymbol{\mathcal{Y}}_{\mathrm{E}} \in \mathbb{R}^{H\times W\times D_{\mathrm{E}}}$, $\boldsymbol{\mathcal{Y}}_{\mathrm{P}} \in \mathbb{R}^{H\times W\times D_{\mathrm{P}}}$, and consequently, $\boldsymbol{\mathcal{Y}}_{\mathrm{EP}} = [\boldsymbol{\mathcal{Y}}_{\mathrm{E}} \quad \boldsymbol{\mathcal{Y}}_{\mathrm{P}}] \in \mathbb{R}^{H\times W\times (D_{\mathrm{E}}+D_{\mathrm{P}})}$. Since the models are

trying to recover the true values from altered and incomplete ESBD data, we do not perform masking on PL data. Beyond white noise, other (perhaps systematic) real perturbations can include corruptions, registration error, and subsampling (e.g., low-resolution). At inference, the inverse solver with diffusion model $\mathcal{M}_{\mathrm{EP}}$, $\mathcal{M}_{\mathrm{E}}$, or $\mathcal{M}_{\mathrm{P}}$ observes $\boldsymbol{\mathcal{Y}}_{\mathrm{EP}}$, $\boldsymbol{\mathcal{Y}}_{\mathrm{E}}$, or $\boldsymbol{\mathcal{Y}}_{\mathrm{P}}$, respectively.

Training data consist of synthetic EBSD and PL images while test data consist of real images. Synthetic EBSD volumes are generated from DREAM.3D (Groeber and Jackson, 2014), and corresponding synthetic PL data are computed with EMsoftOO (Graef, 2024) across 9 equal 40° rotations, standing in as the inter-modality forward model $f_{\mathrm{E}\mapsto\mathrm{P}}$. All PL data are compressed via principal component analysis, keeping the top 3 principal components. All EBSD data are represented as cubochoric coordinates (Dong et al., 2023b; Roşca et al., 2014). All values are then normalized to fall between -1 and 1. The result is 3 channels for EBSD and 3 channels for PL for each voxel (i.e., $D_{\mathrm{E}} = D_{\mathrm{P}} = 3$). At each training batch, we sample slices of shape $H \times W \times D$, where D depends on the modality of the model, from the volumes applied with random geometric data augmentation techniques like rotations and flips. The models are trained to unconditionally recover these samples.

To evaluate, we test on real data from a Ti7 specimen. The sample was part of a serial sectioning experiment, but only a one section was used here. The sample was polished using the second-generation RoboMet.3D$^{\mathrm{TM}}$ starting with 3μ diamond slurry and Struers Largo pad, followed by a 3μ diamond slurry on an Allied Gold Label pad, followed by water on a cleaning pad, followed by 40nm colloidal silica on an Allied Final A pad and ending with water on a cleaning pad. EBSD on the sample was collected with a Tescan Vega SEM at 20 keV and a Bruker Quantax e-Flash 1000 EBSD detector at 3.03 micron pixel size. PL imaging was done using a Zeis Axiovert 200M Optical Microscope and the pixel resolution was 1.04 microns, with the stage rotated every 40°. Since EBSD maps are being used as a ground truth, they need to be corrected for distortions and registered to the PL image. The images were registered using the Simple ITK framework (Beare et al., 2018; Lowekamp et al., 2013; Yaniv et al., 2018) and a methodology previously described (Chapman et al., 2021a). The fixed image was one PL image, downsampled to the same resolution as the

EBSD image. The moving image was a grayscale of the inverse pole figure-X map. A bspline transform warped the collected EBSD data to match the PL data, using mutual information as a metric. While the registration significantly improves the pixel-to-pixel matchups between modalities, some registration error may still exist (Figure 7.15).

Note that EBSD and PL images in this paper have been shifted to aid visual intuition and should not be taken as actual values. Our quantitative evaluation metrics are based on the actual values.

Architecture and Training

All diffusion models are a 6-layer 143M parameter transformer-based U-Nets (Efimov et al., 2025; Ronneberger et al., 2015). We use a hidden dimension of 92, which doubles with every downsampling operation and halves with every upsampling operation. The only architectural difference between multimodal and unimodal diffusion models is the channel dimension of the input and output layers. The height (H) and width (W) are set to 64. We train the models on two textured $200 \times 200 \times 320$ volumes, taking random 64×64 slices for each sample. We use random slices from two $200 \times 200 \times 70$ volumes as validation. In total, we train for 30K gradient steps with a learning rate of 5e$-$4 and batches of 128 samples, evaluating at every 100 steps to select the best checkpoint. The training loss is the empirical denoising loss described by (7.11).

Inference Pipeline

We depict the general inference pipeline in Figure 7.13. We use FPS-SMC (Dou and Song, 2024b), a particle filtering generation algorithm for linear inverse problems with provable consistency guarantees. *In all experiments, we observe the full resolution but noisy PL data and sparsely subsampled EBSD data.* To scale inference, we generate a set of N reconstructions $\{\widehat{\boldsymbol{\mathcal{X}}}_i\}_{i=1}^N$ from a single observation $\boldsymbol{\mathcal{Y}}$ where their exact shapes depend on the number of modalities. Note that each $\widehat{\boldsymbol{\mathcal{X}}}_i$ can be generated in parallel. Although the $\widehat{\boldsymbol{\mathcal{X}}}_i$'s are Monte Carlo samples that can provide information of the reconstruction distribution, it may also be necessary to aggregate them into one final output. This procedure varies

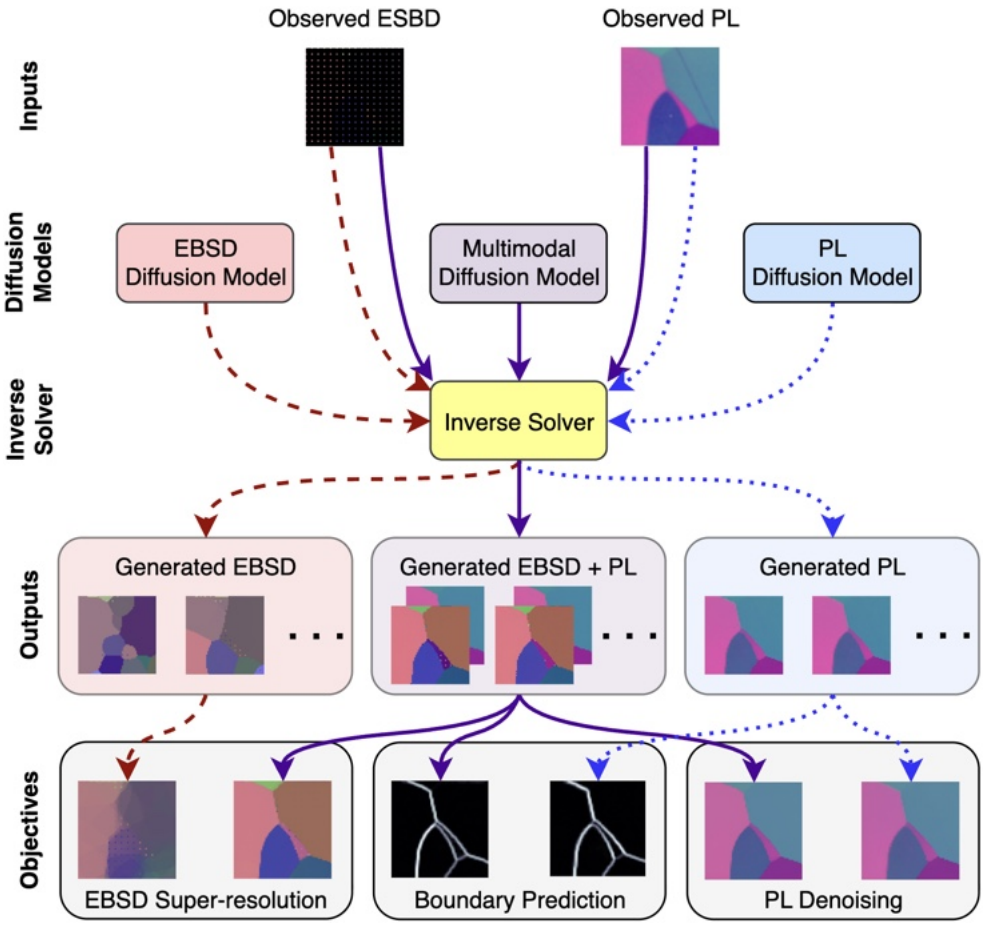

Figure 7.13: With two data modalities (EBSD and PL), one can train an EBSD-only, PL-only, or multimodal diffusion model. Regardless of the model, given observations of EBSD and/or PL data, we sample multiple outputs per input. These outputs are then used in downstream objectives like super-resolution, boundary prediction, and denoising. Unimodal models are restrictive in that they only be applied to a narrow set of tasks that rely on the modality they were trained on. The multimodal diffusion model mixes information from and jointly generates EBSD and PL data to be applicable to the union of tasks that these unimodal models are used for, often even surpassing unimodal performance. The resolution of the observed EBSD in this example is 1/16.

depending on the objective. For the rest of this section, we provide three example settings that we explore in this paper and their aggregation methods, all of which perform post-processing on the same reconstruction set $\{\widehat{\boldsymbol{\mathcal{X}}}_i\}_{i=1}^N$.

Grain Boundary Prediction

In this task, we want to make binary pixel-wise predictions on whether a pixel is on the boundary of a grain. Due to the sparse structure of EBSD and PL data (Dong et al., 2023c), grain boundaries capture uniquely critical information. A boundary pixel is defined to be any pixel that has a face-sharing neighbor from another grain. This information is present in real and synthetic EBSD data where each pixel contains an ID that indicates the grain it belongs in. Pixels with null IDs are ignored.

We use Sobel filtering (Sobel et al., 1968), a common edge detection method, on each reconstructed PL image followed by 0-1 normalization to produce gradient magnitude maps $\{\boldsymbol{S}_i \in [0,1]^{H\times W}\}_{i=1}^{N}$. Taking the average gradient map, we get $\bar{\boldsymbol{S}} = \frac{1}{N}\sum_i \boldsymbol{S}_i$, which roughly measures the confidence that a pixel is a boundary. From here, we can obtain discrete predictions of boundary locations $\boldsymbol{B} \in \{0,1\}^{H\times W}$ where boundary pixels are set to 1. While setting a fixed cutoff for all images is possible, we choose a more adaptive method to automatically identify a custom cutoff per observation. First, we sort the pixel values of $\bar{\boldsymbol{S}}$ in descending order and smooth the resulting curve with a Gaussian kernel. Then, we use the kneed algorithm (Satopaa et al., 2011) which uses estimated second-order information to identify the elbow of the curve. All pixels at or above this cutoff are predicted as boundaries. Since we use PL channels for Sobel filtering (generated boundaries derived from EBSD have greater variance), we compare the performance among the multimodal diffusion model $\mathcal{M}_{\text{EP}}$, PL diffusion model $\mathcal{M}_{\text{P}}$, and a Sobel filter directly applied to observed PL data.

EBSD Super-resolution

The goal of super-resolution is to enhance an observed image into a higher resolution image. This is well suited for EBSD where a lower resolution dramatically speeds up the time-intensive process of data collection. We formulate super-resolution as an inpainting problem. Observed pixels are scattered to the full image size in a uniform grid which is used as $\boldsymbol{\mathcal{Y}}$. The model recovers the remaining unobserved EBSD pixels. Then, a neural network learns to align the reconstructed EBSD values at the observed positions with

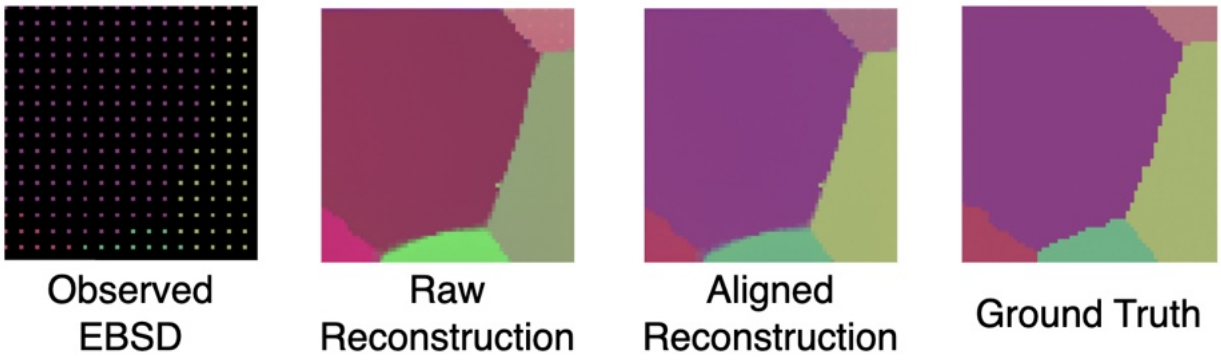

Figure 7.14: Effect of post-reconstruction alignment for super-resolution. By training an alignment network, $\mathcal{A}_\theta$, to learn a mapping to observed EBSD values from corresponding pixels of reconstructed EBSD values, we can reduce systematic error which better recovers the ground truth. This super-resolution example shows an observed EBSD image with a resolution of 1/16.

observed EBSD values and applies the same transformation to all pixels, a simplified post-processing method of Arefeen et al. (2024).

In more detail, we again start with the reconstruction set $\{\widehat{\boldsymbol{\mathcal{X}}}_i\}_{i=1}^N$. First, we take the average to obtain $\bar{\boldsymbol{\mathcal{X}}} = \frac{1}{N}\sum_i \widehat{\boldsymbol{\mathcal{X}}}_i$. Since EBSD data are comparatively more uniform within a grain than PL data (e.g., see Figure 7.15), we use observed EBSD values to align reconstructed EBSD values at the same positions to reduce systematic error like minor shifts in values, as generated EBSD images are also fairly uniform within a grain. More concretely, let Ω be the set of observed coordinates in EBSD (i.e., nonempty coordinates in $\boldsymbol{\mathcal{Y}}_{\mathrm{E}}$). For example, if we observe a resolution of 1/16, Ω would contain the grid coordinates of every 4th row and column. For each observation, we train small neural network, $\mathcal{A}_\theta$, that minimizes the following pixel-wise mean squared error loss:

$$\mathcal{L}_{\mathrm{MSE}}(\theta) = \frac{1}{|\Omega|}\sum_{(i,j)\in\Omega} \|\mathcal{A}_\theta([\bar{\boldsymbol{\mathcal{X}}}]_{i,j}) - [\boldsymbol{\mathcal{Y}}_{\mathrm{E}}]_{i,j}\|_2^2. \tag{7.13}$$

Finally, we apply this same neural network to the all pixels in $\bar{\boldsymbol{\mathcal{X}}}$ to align every EBSD value for the final output $\mathcal{A}_\theta(\bar{\boldsymbol{\mathcal{X}}}) \in \mathbb{R}^{H\times W\times D_{\mathrm{E}}}$. The effect of alignment is visualized in Figure 7.14. We compare performance between the multimodal diffusion model $\mathcal{M}_{\mathrm{EP}}$ and EBSD diffusion model $\mathcal{M}_{\mathrm{E}}$. The PL model $\mathcal{M}_{\mathrm{P}}$ is not applicable here.

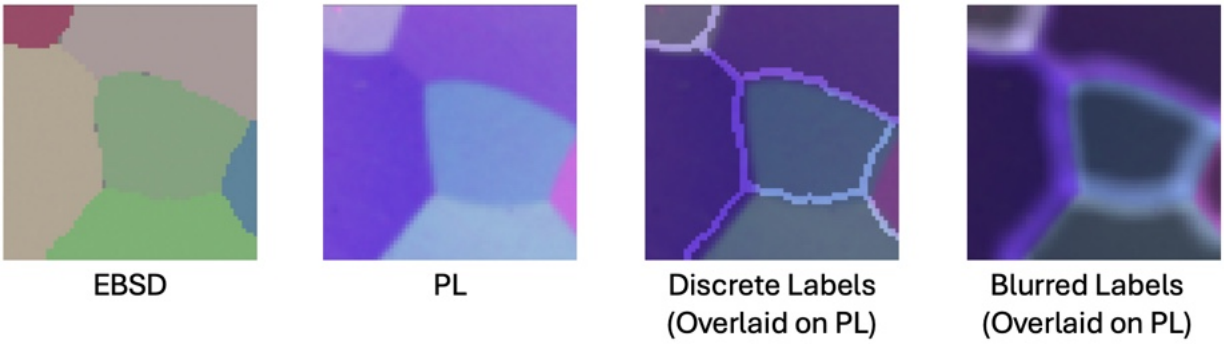

Figure 7.15: Due to image registration errors, grains in EBSD data and PL data may have slight positional mismatches. We highlight this difference in the third image from the left where ground truth boundary labels $\boldsymbol{B}^\star$ calculated from EBSD do line up with boundary pixels in PL data. Gaussian blurred ground truth boundaries $g(\boldsymbol{B}^\star)$ overlaps with boundaries present in PL data, helping BCE loss be robust to slight spatial shifts in predictions. While this directly impacts boundary prediction, this can impact the generation process and evaluation of other objectives as well.

The alignment network $\mathcal{A}_\theta$ is a 2-layer neural network with a hidden dimension of 32 separated with a GELU activation (Hendrycks and Gimpel, 2016). Setting aside 20% of Ω for validation, $\mathcal{A}_\theta$ is trained with a learning rate of 0.02 with Adam (Kingma and Ba, 2014) in batches of 16 for 50 epochs. Due to the minuscule network and data size, this takes less than 5 seconds to train, a relatively insignificant cost.

PL Denoising

Denoising cleans an image of unwanted artifacts like noise and corruptions. Here, we adopt the straightforward process of averaging the PL values in the reconstruction set: $\bar{\boldsymbol{x}}_{\mathrm{P}} = \frac{1}{N}\sum_i \widehat{\boldsymbol{x}}_{\mathrm{P},i}$. This is because the generation process will naturally produce a denoised image.

7.4.3 Core Experiments

Multimodal diffusion as a joint prior between EBSD and PL data demonstrates strong performance on multiple objectives including grain boundary prediction, super-resolution, and denoising. As described in Section 7.4.2, all objectives use the same reconstruction sets from all diffusion models and differ primarily in their post-processing procedures. Unless

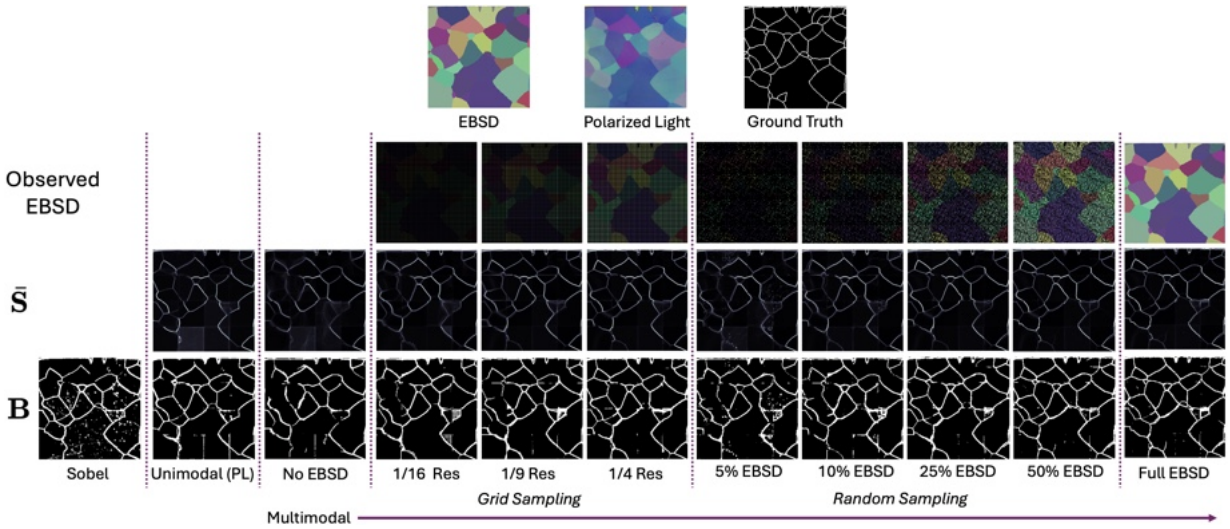

Figure 7.16: Results of 16 64×64 images stitched together for a continuous 256×256 image. From left to right, the top row shows the EBSD measurements, PL measurements, and ground truth boundaries $\boldsymbol{B}^\star$ derived from EBSD. The second row from the top shows the observed EBSD, if relevant, in each setting. The third row shows the aggregated Sobel maps $\bar{\boldsymbol{S}}$ of diffusion-based methods with brighter colors indicating higher values. The bottom row shows the predicted boundaries $\boldsymbol{B}$ of all methods. Best viewed zoomed in.

stated otherwise, we set the reconstruction set size $N = 10$ and repeat 10 times to obtain error bars.

Grain Boundary Prediction

Derived from EBSD grain IDs, let $\boldsymbol{B}^\star \in \{0,1\}^{H \times W}$ be the ground truth binary images with boundary pixels are set to 1. Since image registration is imperfect (see Figure 7.15), point-wise metrics like accuracy and binary cross entropy (BCE) can accentuate errors where predictions and labels are slightly shifted from each other. Thus, we evaluate boundary prediction with two more positionally robust metrics: Chamfer distance and Gaussian blurred BCE.

The Chamfer loss (Ravi et al., 2020) measures distance between point clouds $\mathcal{P} = \{(\frac{i}{H}, \frac{j}{W}) | [\boldsymbol{B}]_{i,j} = 1\}$ and $\mathcal{P}^\star = \{(\frac{i}{H}, \frac{j}{W}) | [\boldsymbol{B}^\star]_{i,j} = 1\}$, where $\boldsymbol{B}$ is the predicted binary

boundary image, defined as

$$C(\mathcal{P}, \mathcal{P}^\star) \coloneqq \underbrace{\frac{1}{|\mathcal{P}|}\sum_{\boldsymbol{p}\in\mathcal{P}} \min_{\boldsymbol{q}\in\mathcal{P}^\star} \|\boldsymbol{p}-\boldsymbol{q}\|_2^2}_{\overrightarrow{C}(\mathcal{P},\mathcal{P}^\star)} + \underbrace{\frac{1}{|\mathcal{P}^\star|}\sum_{\boldsymbol{q}\in\mathcal{P}^\star} \min_{\boldsymbol{p}\in\mathcal{P}} \|\boldsymbol{q}-\boldsymbol{p}\|_2^2}_{\overrightarrow{C}(\mathcal{P}^\star,\mathcal{P})}. \tag{7.14}$$

Here, $\overrightarrow{C}(\mathcal{P}, \mathcal{P}^\star)$ and $\overrightarrow{C}(\mathcal{P}^\star, \mathcal{P})$ are asymmetric forward and backward Chamfer losses, respectively. The forward Chamfer loss intuitively penalizes overpredicting the number of boundary points (since $|\mathcal{P}|$ depends on the predictions) and misclassifying distant points as a boundary, analogous to Type I error. Conversely, the backward Chamfer loss penalizes missing a boundaries, analogous to Type II error.

As a metric to measure continuous predictions, we use $\mathcal{L}_{\text{G-BCE}}$, BCE between aggregated normalized Sobel outputs, $\bar{\boldsymbol{S}}$, and Gaussian blurred ground truth boundaries, $g(\boldsymbol{B}^\star)$. We opt to blur ground truth labels because of slight label mismatch between EBSD and PL images as depicted in Figure 7.15. Blurring allows greater robustness to tolerate minor spatial discrepancies between predicted and target boundary pixels. More formally,

$$\mathcal{L}_{\text{G-BCE}}(\bar{\boldsymbol{S}}, \boldsymbol{B}^\star) = \frac{1}{HW}\sum_{i,j} [g(\boldsymbol{B}^\star)]_{i,j} \log[\bar{\boldsymbol{S}}]_{i,j} + (1 - [g(\boldsymbol{B}^\star)]_{i,j}) \log(1 - [\bar{\boldsymbol{S}}]_{i,j}).$$

We set the standard deviation and radius of the Gaussian filter to be 3 and 5, respectively.

In Figure 7.17, we show comparisons between applying Sobel filtering on the PL data, the PL unimodal model $\mathcal{M}_{\text{P}}$, and the multimodal model $\mathcal{M}_{\text{EP}}$. We observe that multimodal diffusion naturally reduces all loss metrics as we observe more EBSD. Both diffusion models are overall much more accurate than simply applying a Sobel filter on PL observations, but the lone exception is the backward Chamfer loss. Even so, the forward Chamfer loss is significantly larger in magnitude and thus makes up most of the bidirectional Chamfer loss. This implies that the Sobel filtering baseline tends to make more Type I errors, which can be visually validated in Figure 7.16 where many intra-grain pixels are predicted as boundaries. When we look at the percentage reduction in Chamfer loss in Figure 7.18, we see that the differences between the diffusion models and the Sobel filtering baseline are

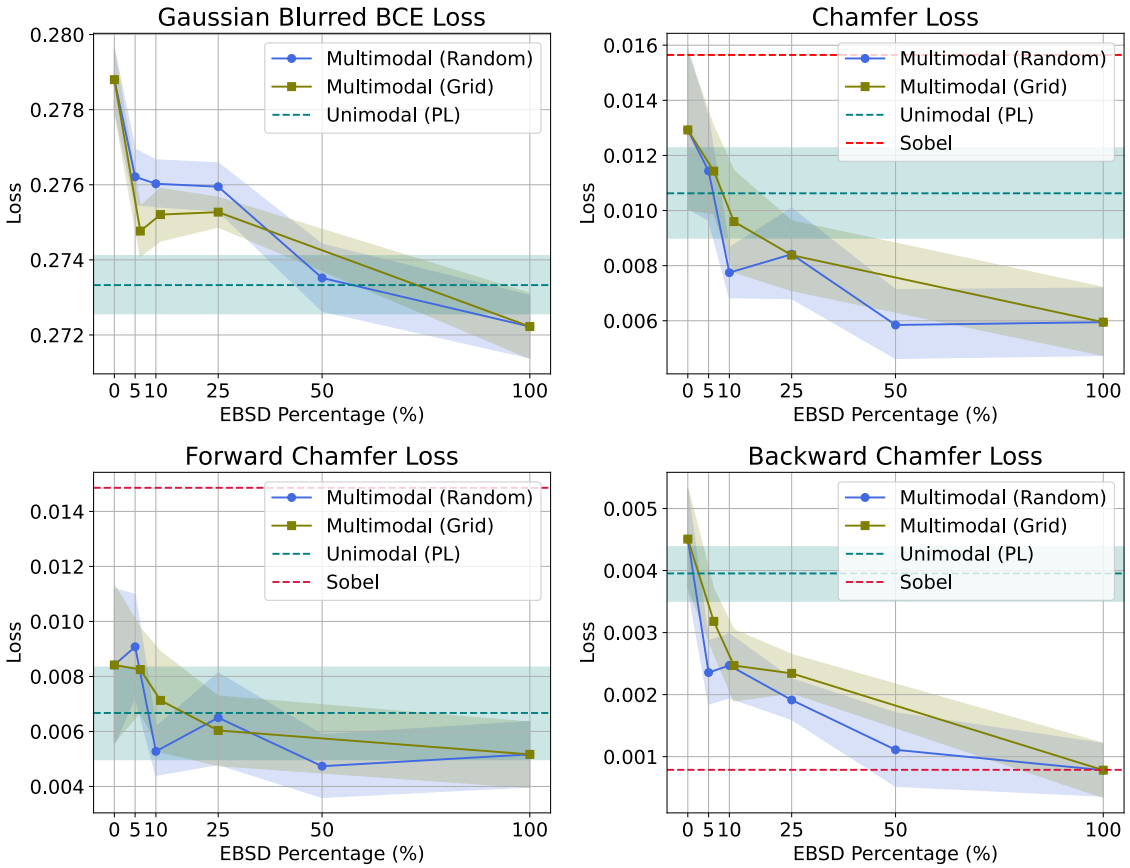

Figure 7.17: Going clockwise starting from the top left, the Gaussian blurred BCE loss, Chamfer loss, backward Chamfer loss, and forward Chamfer loss of each model as more EBSD samples are observed. Multimodal results of randomly and uniformly subsampled (low-resolution) EBSD observations are shown in blue circles and dark green squares, respectively. The red dashed lines show the Sobel baseline performance. The Sobel baseline cannot be evaluated with Gaussian blurred BCE since it does not produce a probability distribution. Shaded regions illustrate two standard errors. Lower is better.

more pronounced. In general, the performance difference between random and grid (lower resolution) sampling observed EBSD pixels is not statistically significant. The slightly lower performance of multimodal diffusion in the complete absence of EBSD data is discussed in Section 7.4.4.

We also investigate the relationship between the amount of EBSD observed and N, the number of repeated generations per input. From Figure 7.19, scaling N generally helps both the unimodal and multimodal diffusion models. Hence, there is a tangible benefit to parallel inference scaling.

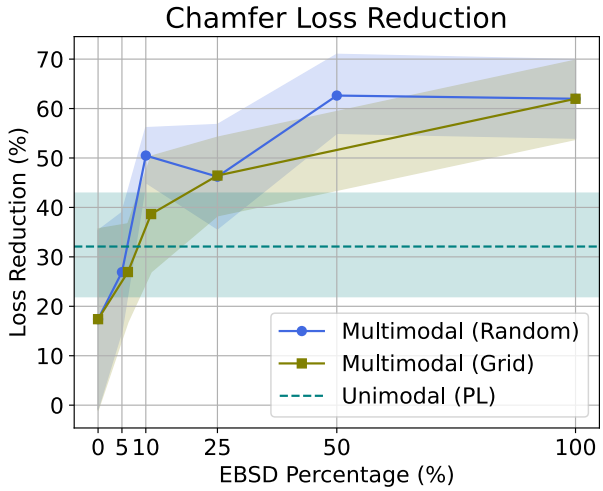

Figure 7.18: Average percent reduction in Chamfer loss per sample compared to pure Sobel filtering. Multimodal results of randomly and uniformly subsampled (low-resolution) EBSD observations are shown in blue circles and dark green squares, respectively. The diffusion models all show greatly improved predictions. Shaded regions illustrate two standard errors. Higher is better.

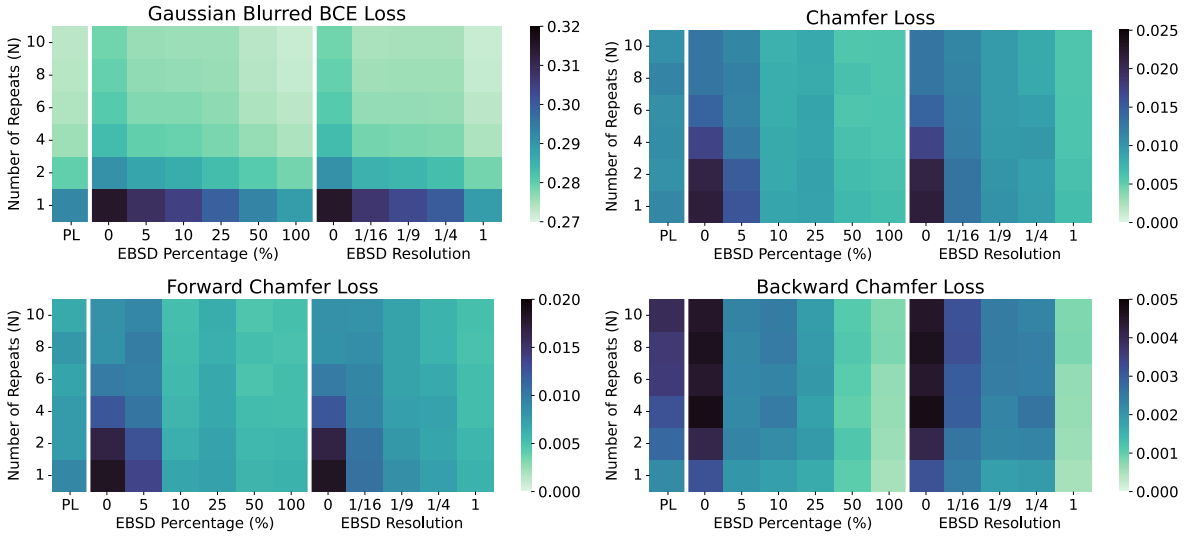

Figure 7.19: Clockwise from the top left: the Gaussian blurred BCE loss, Chamfer loss, backward Chamfer loss, and forward Chamfer loss as we vary the number of repeats N and observed EBSD percentage. From left to right in each quadrant, the blocks in each row show metrics from the unimodal model, multimodal model with randomly sampled EBSD observations, and multimodal model with low-resolution EBSD. We generally see improved performance as N and the amount of observed EBSD increases. Note the difference in numerical ranges for each row. Lighter colors are better.

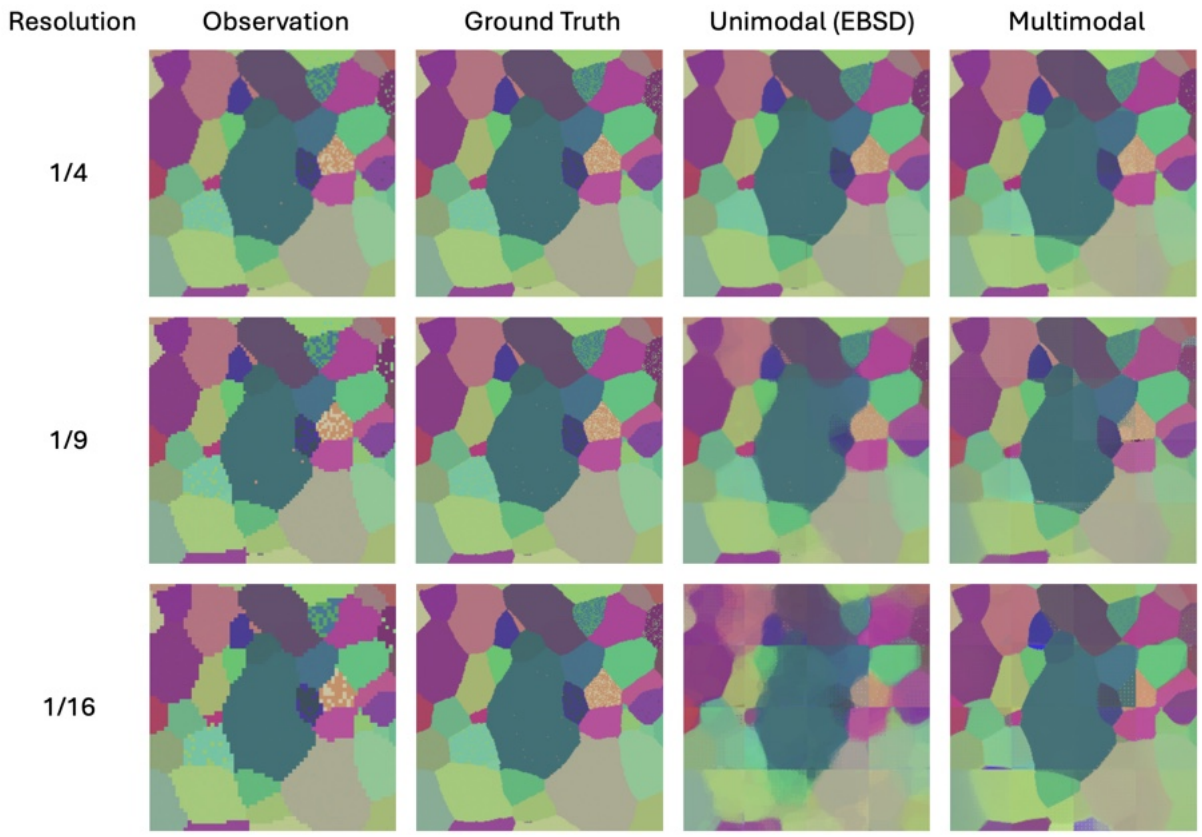

Figure 7.20: Super-resolution using diffusion models. The multimodal (EBSD and PL) model is more robust as the observed resolution diminishes. Best viewed zoomed in.

Super-resolution Enhancement

Our model also possesses strong performance on super-resolution. By scattering an observed resolution into the desired resolution and using this as the observed EBSD data, the models inpaint the remaining pixels. Shown in Figure 7.20, both the EBSD unimodal model $\mathcal{M}_{\text{E}}$ and multimodal model $\mathcal{M}_{\text{EP}}$ accurately perform EBSD super-resolution from 25% of the pixels. As the observed resolution decreases, the recovery with the multimodal model produces crisper images than with the unimodal model. Quantitatively in Figure 7.21, we see that the multimodal model is able to produce more accurate EBSD images with lower resolution observations, as measured by disorientation (Larsen and Schmidt, 2017; Varley, 2024), especially along the grain boundaries. This likely due to greater reliance on PL data for lower resolution images, making interpolation much more ill-posed for the unimodal diffusion model. The slight difference in quality between the diffusion

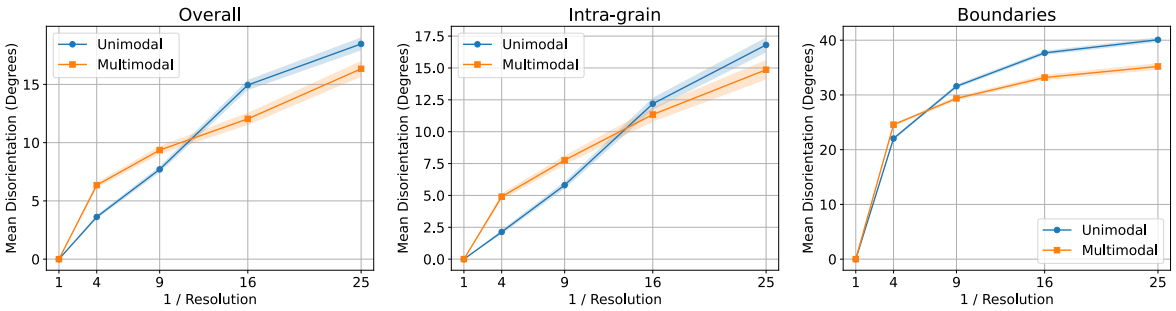

Figure 7.21: Mean disorientation error across all unobserved EBSD pixels (top), unobserved intra-grain pixels (bottom left), and unobserved boundary pixels (bottom right) from varying super-resolution factors. Shaded regions indicate two standard errors. With lower resolution EBSD data, the multimodal diffusion model is more performant with improvements more concentrated along the grain boundaries.

models when we observe 1/4 resolution EBSD images may be related to model capacity issue discussed in Section 7.4.4.

Diffusion as a Denoiser

Finally, we take a qualitative look at an implicit perk of diffusion: denoising. By simply taking the average across all N outputs from the inverse solver, we are able to significantly clean up white noise and scratches in PL data, as depicted in Figure 7.22. With more EBSD observations, the boundaries between grains with very similar PL measurements become more apparent.

7.4.4 Error Analysis & Limitations

Looking closer at the outputs of these methods, we identify cases where some methods may be preferred over others.

Noisy PL Data

Direct Sobel filtering on PL data suffers in the presence of perturbations. In turn, this can affect boundary predictions. To highlight this issue, we inject zero-centered Gaussian noise with standard deviation 0.05 into (already slightly noisy) observed PL data. Structured

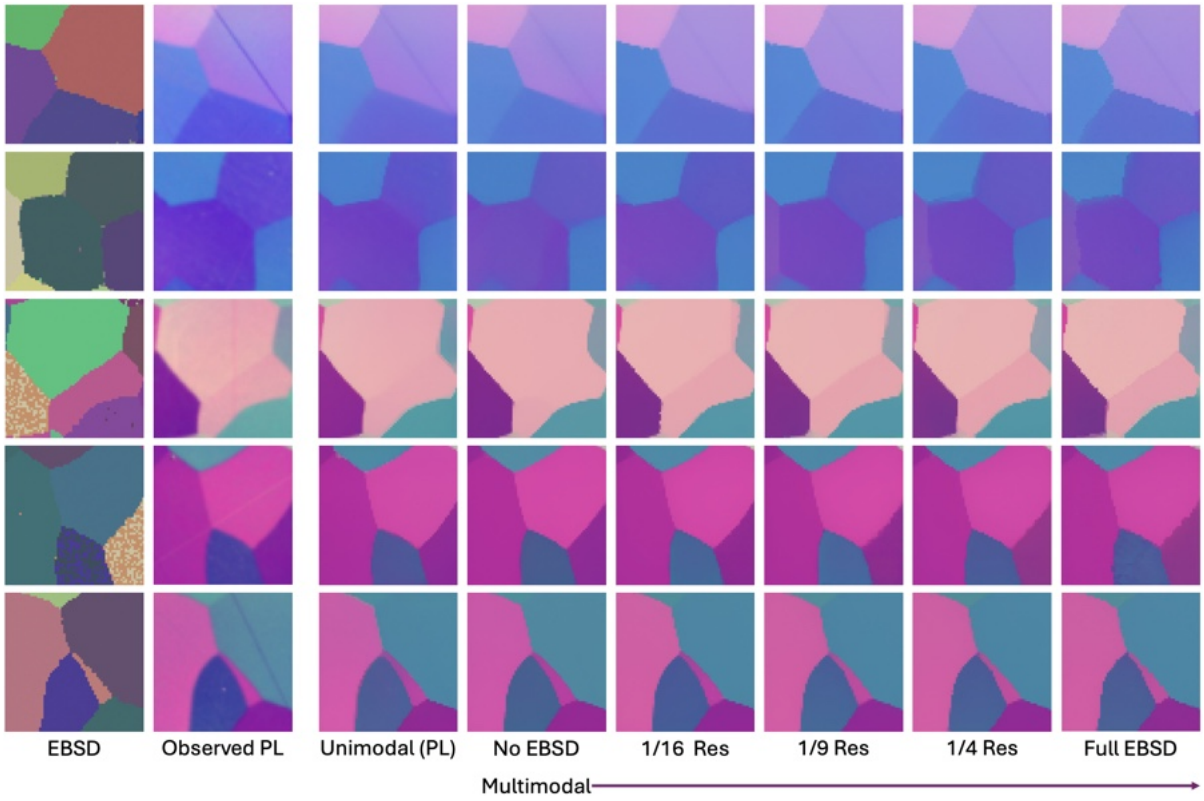

Figure 7.22: Examples of PL denoising. The left two columns are the EBSD and PL data. Continuing to the right, the columns show the outputs from the unimodal model $\mathcal{M}_{\text{P}}$ and multimodal model $\mathcal{M}_{\text{EP}}$ (ranging from no EBSD to higher resolutions of observed EBSD).

noise can also occur in real data, such as in the case of scratches, long thin streaks of discoloration. The effects of both of these types of noise are shown in Figure 7.23. In the case of Gaussian noise, the resulting Sobel output contains speckled predictions. For scratches, Sobel filtering incorrectly also labels the scratch as a boundary. In contrast, the diffusion models naturally perform denoising, dampening this issue, as we explored earlier in Section 7.4.3 and Figure 7.22.

Registration Error

Registering PL and EBSD data can result in slight errors (Figure 7.15). While most pixels would be relatively unaffected due to the data's piecewise constant structure, disruptions to the boundaries can be significant. This is the reason behind using Gaussian blurred BCE loss and Chamfer loss as evaluation metrics. In addition to evaluation, registration error

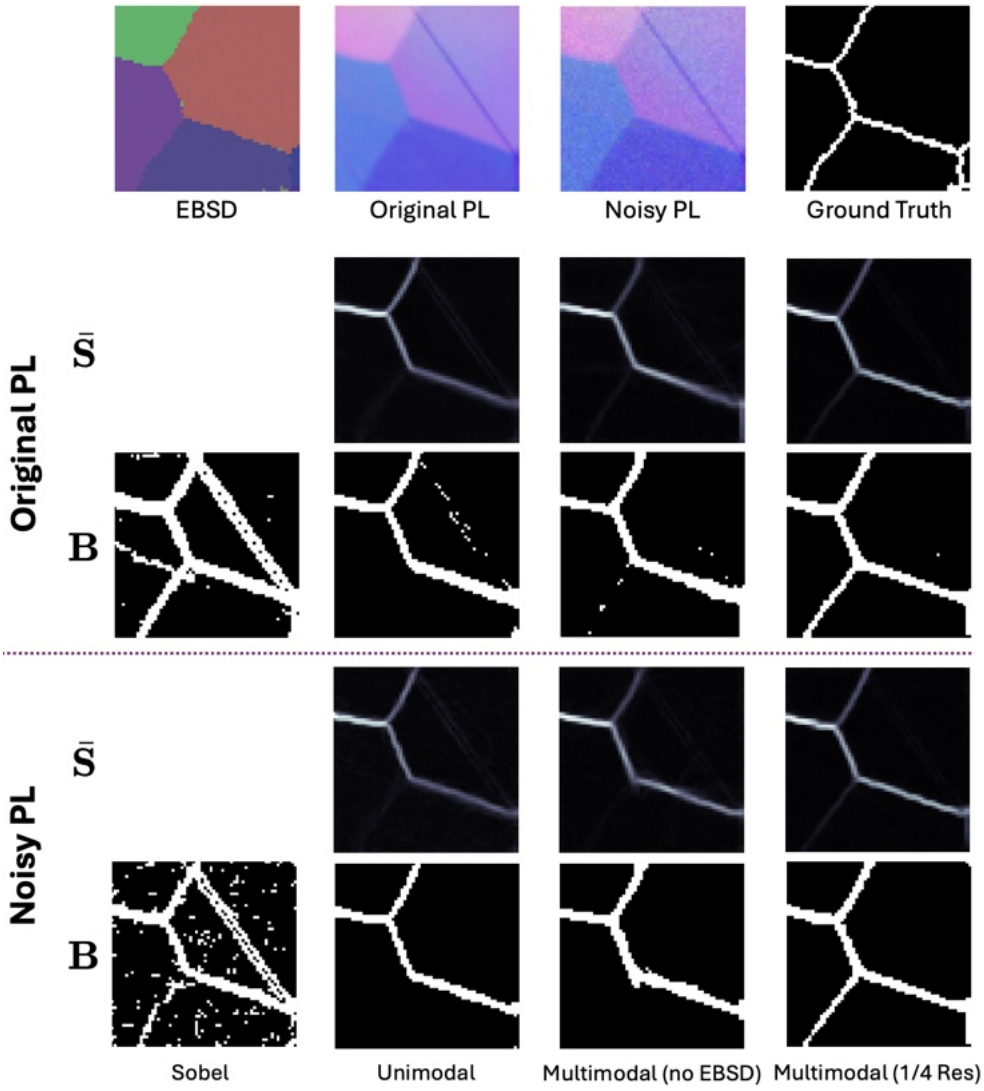

Figure 7.23: The effect of noise in PL measurements. The diffusion models are more robust to noise and scratches compared to direct Sobel filtering. Noise is sampled from a centered Gaussian distribution with variance 0.05. $\bar{\boldsymbol{S}}$ and $\boldsymbol{B}$ are the aggregated Sobel maps and predicted boundaries, respectively.

can also harm the generated reconstructions, too. Perhaps counterintuitively, observing more EBSD data can confuse the inverse solver if the two modalities have significant registration error. Shown in Figure 7.24, when observing full EBSD data, grain boundary predictions are duplicated where one set corresponds to PL observations and the other shifted set corresponds to EBSD observations. With sparser EBSD observations, the grain boundaries in EBSD become less obvious, and therefore, duplicate boundaries become less

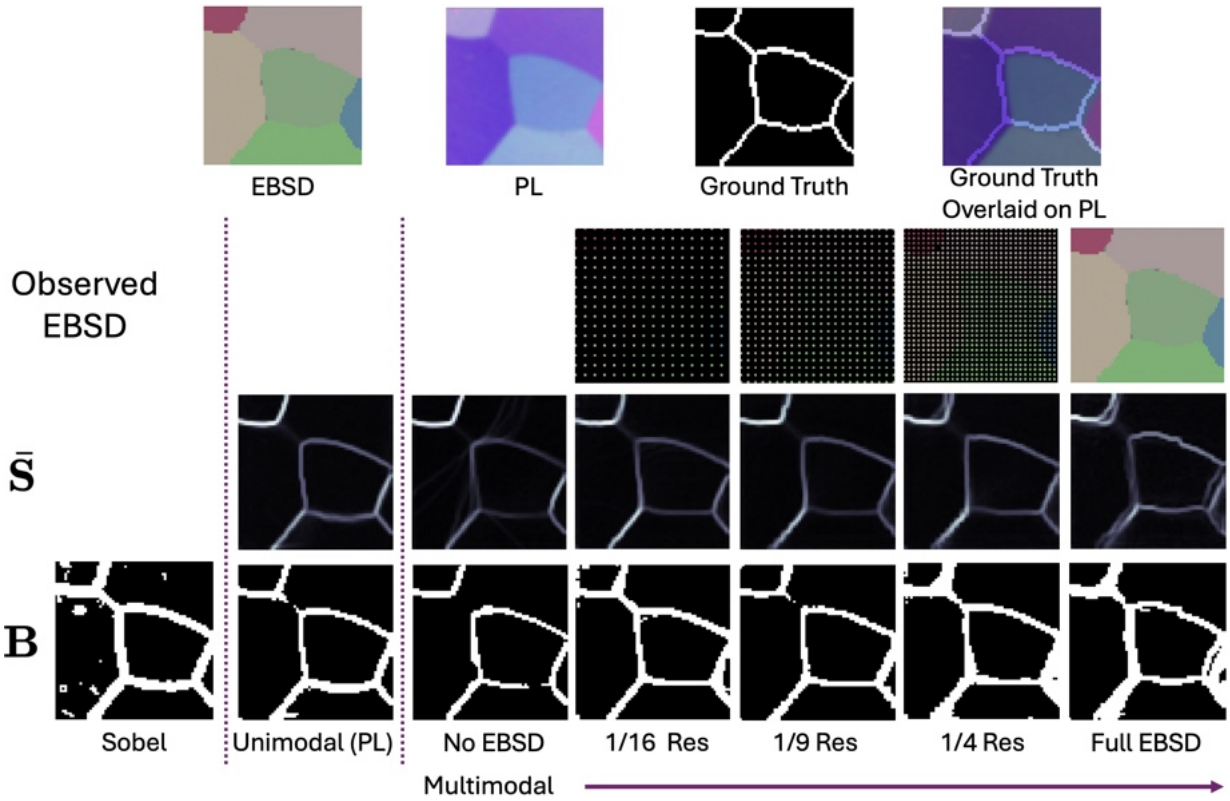

Figure 7.24: Example where observing more EBSD harms the performance due to misregistration. From left to right, the top row shows the EBSD measurements, PL measurements, ground truth boundaries $\boldsymbol{B}^\star$ derived from EBSD, and $\boldsymbol{B}^\star$ overlaid on the PL image. The middle row shows the aggregated Sobel maps $\bar{\boldsymbol{S}}$ of diffusion-based methods with brighter colors indicating higher values. The bottom row shows the predicted boundaries $\boldsymbol{B}$ of all methods. Misregistration between EBSD and PL causes noisy predictions in the form of thicker and dual boundaries.

of an issue. Interestingly, this raises another possible application of inverse solvers with multimodal diffusion for automatic image registration, which is a valuable future direction to explore.

Subtle Grain Differences

Subtle changes between grains pose a challenge for all methods, though to varying extents. Depicted in Figure 7.25 and third row of Figure 7.22, all methods have trouble separating neighboring grains with similar values. Fortunately, with the assistance of another modality, the inverse solvers with multimodal diffusion can more easily differentiate between

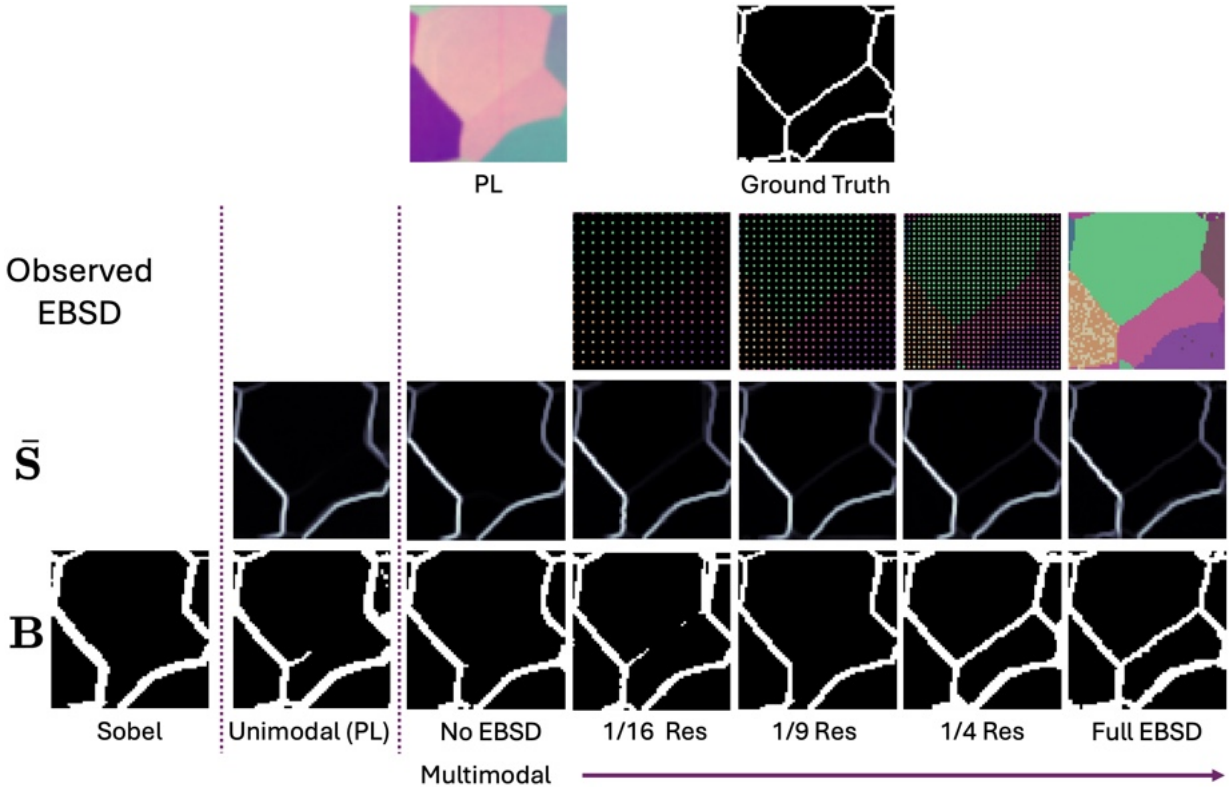

Figure 7.25: Example where observing more EBSD provides clarity on similar PL values between grains. From left to right, the top row shows the PL measurements and ground truth boundaries $\boldsymbol{B}^\star$ derived from EBSD. The second row shows the observed EBSD with increasing resolution from left to right. The third row shows the aggregated Sobel maps $\bar{\boldsymbol{S}}$ of diffusion-based methods with brighter colors indicating higher values (some pixels can be very faint). The bottom row shows the predicted boundaries $\boldsymbol{B}$ of all methods. Best viewed zoomed in.

individual grains.

Missing Modality

With no observed EBSD, multimodal diffusion performs slightly worse than the unimodal diffusion model for boundary prediction (Figure 7.17). In fact, slight degradation when an entire modality is missing at inference for multimodal AI models is a phenomenon that has been observed beyond our setting and remains an active area of research (Deng et al., 2025; Hagström and Johansson, 2022; Wang et al., 2020c). There are a couple hypotheses worth exploring in future work to investigate why this is happening. First is

model capacity. While the unimodal diffusion model just needs to learn the structure of PL data, the multimodal diffusion model also needs to learn the structure of EBSD data and the interaction between EBSD and PL data within the same number training samples and parameters, making this a question of scale. Second is over-reliance on one modality (Deng et al., 2025; Wu et al., 2024a). Our multimodal diffusion model may be placing too much emphasis on EBSD data, posing a challenge for inverse solvers to properly guide the generation process when only PL observations are accessible.

7.4.5 Work Summary

Joint EBSD and PL multimodal diffusion shows immense promise as a base generative model applicable to numerous inverse problems where information from one or more modalities is needed. Trained solely on synthetic data, our method's efficacy transfers to real EBSD and PL data without additional training. Furthermore, by sampling multiple reconstructions with the same observation, we can obtain distributions of reconstruction which provide richer and higher quality content than a single deterministic prediction. Focusing particularly on grain boundary prediction, we show that low-resolution or sparse EBSD observations can greatly improve prediction quality. Along with demonstrations on denoising and super-resolution, multimodal diffusion is effective at integrating information from PL data to supplement low-resolution EBSD to accomplish these objectives. With particularly strong results at 1/4 resolution, this allows us to accelerate EBSD data collection by 4×. There are still many exciting directions to explore to further push the capability of our models. For example, we would like to explore search-based algorithms for generation instead of just Monte Carlo sampling as we have done in this paper. Additionally, it would be interesting to examine the benefit of scaling the model and data which has seen success in traditional machine learning domains.

7.5 Chapter Summary

EBSD microscopy is a powerful tool useful in many materials science engineering applications, yet its cost and sensitivity limits its potential. Modern machine learning methods like transformer and diffusion models can help but must overcome several domain-specific obstacles such as real data scarcity, non-Euclidean dynamics, and high-order tensor data. On the flip side, there are also domain-specific properties we can leverage to design physics-informed and scalable algorithms.

This chapter presents two methods to accelerate and recover EBSD data collection. First in Section 7.3, we introduce a small and efficient transformer-based algorithm to recover missing EBSD data slices and accelerate collection by up to 25%. Our method shows greatly improved performance compared to current methods in practice. Then in Section 7.4, we can accelerate collection further with the help of PL data, a much cheaper imaging modality, using multimodal diffusion and parallel inference scaling. The same inverse solver and diffusion model demonstrate strong generalization to various tasks (e.g., super-resolution, boundary prediction, and denoising) with only 1/4 the original resolution of EBSD data. Trained on synthetic data, both methods transfer nearly seamlessly to real data.

Taking a broader view, opportunities for machine learning exist beyond accelerating EBSD collection and materials science. Intricacies and idiosyncrasies of domains outside of natural images and text can prevent direct application of state-of-the-art deep learning techniques due to lack of scalability or interpretability. Thankfully, with some careful domain adaptation, the benefits of AI can be extended to aid the sciences.

Chapter 8

Conclusion & Final Remarks

This thesis explored several methods to improve the efficiency-performance tradeoff of LLM generation by studying the model architectures, data, features, and hardware. Different settings (e.g., server and local inference) and different model types (e.g., base, instruct, and reasoning models) create different computational bottlenecks. Thankfully, characteristics like sparsity, low-rank, and parallelism permeate in many of these settings. In Chapter 3, we showed by synthesizing a low-rank RNN-like component to any sparse KV cache eviction method, `LESS` accelerates inference. Tested with various model architectures and tasks, `LESS` achieves higher accuracy than any sparse method alone and better captures the semantics of text. In Chapter 4, we uncovered the surprising phenomenon of flocking, contextually structured sparse FF features after a quick transformation, and proposed `GRIFFIN` which exploits these observations. `GRIFFIN` is a purely test-time method which adaptively prunes FF neurons to achieve significant reductions in generation latency without quality degradation on language tasks. In Chapter 5, seeing many efficient methods decimate performance on math and reasoning tasks in part due to error accumulation across long generations, we introduced `Caprese`, a low-rank component that seamlessly integrates into sparse FF blocks for fast parallel inference. Moreover, `Caprese` recovers most if not all of the lost performance on reasoning tasks caused by sparse FF methods in instruct and reasoning LLMs. In Chapter 6, we generalized parallel scaling of LLM inference with `Bridge` by allowing information to be transferred between multiple responses to a

single prompt throughout the generation process. We demonstrated `Bridge` better utilizes equivalent compute and parameters to produce higher quality responses across reasoning domains. Finally, in Chapter 7, we revealed that similar intuitions can transfer to materials science microscopy. By adapting modern deep learning architectures and algorithms, we are able to accelerate and improve imaging for an expensive data modality.

In addition to those mentioned in the summaries of each chapter, there are still many exciting general future directions to explore. To name a few:

- *Compute adaptive methods:* Model inference can be made more flexible. For example, explicit resource allocation methods can determine how much compute should be used for each input depending on difficulty (Singh et al., 2025). Additionally, as hardware becomes more diverse, they may have different behaviors that inference algorithms can take advantage of.
- *Alternative architectures:* To address inference efficiency closer to the root cause, exploring alternative architectures that are designed with efficiency in mind may be more effective than enforcing algorithms during post-training or inference. Methods like Mamba (Gu and Dao, 2023) and Gated DeltaNet (Yang et al., 2024b) show great promise but matching the performance of transformers remains a challenge.
- *Training efficiency:* With the growing popularity of synthetic data and reinforcement learning for AI models, there is a greater need for efficient training methods. Possible areas of improvement can include parameter efficiency, optimizer algorithms, data quality/filtering, and compute scheduling.
- *Scientific domains:* Leveraging state-of-the-art AI methods for the problems in scientific fields has enormous potential, but they can pose unique domain-specific challenges like physical constraints and data scarcity. Having shown their AI's potential in materials science applications, we hope to carry the insight to other scientific problems and domains.

Throughout this current era of AI development, there has always been a tug of war between performance and efficiency. In times of sufficient resources, we think about scaling our models to achieve the greatest performance. Once efficiency becomes an urgent prob-

lem, we think about slashing wasteful operations to bring us back within budget without impacting performance. Then, the pendulum swings back again. Hence, innovations improving efficiency provide temporary relief on strained resources, yet growing expectations of AI have repeatedly pushed demand to catch up and exceed excess supply. This does not mean that work in efficiency is fruitless. In fact, it is a necessity for AI development. For example, early transformers used context lengths on the order of hundreds or low thousands of tokens (Radford et al., 2019; Zhang et al., 2022), limited by computational constraints. In response, efficiency advancements in attention blocks led to possible context lengths on the order of hundreds of thousands or millions (Dubey et al., 2024; Team et al., 2024a; Yang et al., 2024a), beyond what was needed for many use cases at the time. However, this unlocked new possibilities that were once impractical before, such as complex reasoning and agents (Guo et al., 2025; Team et al., 2025). And once again, these tasks pose an efficiency problem. Although this example is a bit of an oversimplification, the takeaway is that efficiency and performance drive steady progress of each other and in turn, of AI capabilities.

Bibliography

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. (2023). Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*. 1

AIME (2025). AIME problems and solutions. 5.4.1, 6.3.1

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., and Sanghai, S. (2023). Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*. 2.1.1

Almazrouei, E., Alobeidli, H., Alshamsi, A., Cappelli, A., Cojocaru, R., Debbah, M., Goffinet, E., Heslow, D., Launay, J., Malartic, Q., Noune, B., Pannier, B., and Penedo, G. (2023). Falcon-40B: an open large language model with state-of-the-art performance. 3.1, 3.1, 3.2.1

AMC (2023). American mathematics competition. 5.4.1, 6.3.1

Anthropic (2024). The claude 3 model family: Opus, sonnet, haiku. 1

Arefeen, Y., Levac, B., Stoebner, Z., and Tamir, J. I. (2024). Infusion: Diffusion regularized implicit neural representations for 2d and 3d accelerated mri reconstruction. In *2024 58th Asilomar Conference on Signals, Systems, and Computers*, pages 1886–1890. IEEE. 7.4.2

Azad, R., Aghdam, E. K., Rauland, A., Jia, Y., Avval, A. H., Bozorgpour, A., Karimijafarbigloo, S., Cohen, J. P., Adeli, E., and Merhof, D. (2024). Medical image segmentation review: The success of u-net. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 6.2.1

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*. 6.1

Bahdanau, D., Cho, K. H., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015*. 1.4, 2.1.1

Bai, Y., Jones, A., Ndousse, K., Askell, A., Chen, A., DasSarma, N., Drain, D., Fort, S., Ganguli, D., Henighan, T., et al. (2022). Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*. 2.1.3

Baraniuk, R. G. (2007). Compressive sensing [lecture notes]. *IEEE signal processing magazine*, 24(4):118–121. 7.4

BBC (2015). Fracking still opposed in wales, ministers tell councils. *The British Broadcasting Corporation*. 3.1, 3.13

BBC (2016). Three us hospitals hit by ransomware. *The British Broadcasting Corporation*. 4.6

Beare, R., Lowekamp, B., and Yaniv, Z. (2018). Image segmentation, registration and characterization in r with simpleitk. *Journal of statistical software*, 86:1–35. 7.4.2

Bisk, Y., Zellers, R., Bras, R. L., Gao, J., and Choi, Y. (2020). Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*. 4.4.1

Blalock, D., Gonzalez Ortiz, J. J., Frankle, J., and Guttag, J. (2020). What is the state of neural network pruning? *Proceedings of machine learning and systems*, 2:129–146. 2.2.2

Bolya, D., Fu, C.-Y., Dai, X., Zhang, P., Feichtenhofer, C., and Hoffman, J. (2022). Token merging: Your vit but faster. *arXiv preprint arXiv:2210.09461*. 2.2.1, 7.3.1

Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2):123–140. 2.2.4

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32. 2.2.4

Brown, B., Juravsky, J., Ehrlich, R., Clark, R., Le, Q. V., Ré, C., and Mirhoseini, A. (2024). Large language monkeys: Scaling inference compute with repeated sampling.

arXiv preprint arXiv:2407.21787. 2.2.4, 5.1

Brumfield, B. (2015). Death toll rises quickly as conflict rages in yemen. *The Cable News Network*. 3.12

BRUMO (2025). Brown university math olympiad 2025. 5.4.1, 6.3.1

Calcagnotto, M., Ponge, D., Demir, E., and Raabe, D. (2010). Orientation gradients and geometrically necessary dislocations in ultrafine grained dual-phase steels studied by 2d and 3d ebsd. *Materials Science and Engineering: A*, 527(10-11):2738–2746. 7.1

Candès, E. J., Li, X., Ma, Y., and Wright, J. (2011). Robust principal component analysis? *Journal of the ACM (JACM)*, 58(3):1–37. 2.2.1

Candès, E. J., Romberg, J., and Tao, T. (2006). Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on information theory*, 52(2):489–509. 7.4

Cardoso, G., Le Corff, S., Moulines, E., et al. (2023). Monte carlo guided denoising diffusion models for bayesian linear inverse problems. In *The Twelfth International Conference on Learning Representations*. 7.4.1

Chandrasekaran, V., Sanghavi, S., Parrilo, P. A., and Willsky, A. S. (2011). Rank-sparsity incoherence for matrix decomposition. *SIAM Journal on Optimization*, 21(2):572–596. 2.2.1

Chang, Y.-L., Liu, Z. Y., Lee, K.-Y., and Hsu, W. (2019). Free-form video inpainting with 3d gated convolution and temporal patchgan. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9066–9075. 7.3

Chapman, M., Shah, M., Donegan, S., Scott, J. M., Shade, P., Menasche, D., and Uchic, M. (2021a). Afrl additive manufacturing modeling series: Challenge 4, 3d reconstruction of an in625 high-energy diffraction microscopy sample using multi-modal serial sectioning, june 2021. *Integrated Materials and Manufacturing Innovation*. 7.4.2

Chapman, M. G., Shah, M. N., Donegan, S. P., Scott, J. M., Shade, P. A., Menasche, D., and Uchic, M. D. (2021b). Afrl additive manufacturing modeling series: challenge 4, 3d

reconstruction of an in625 high-energy diffraction microscopy sample using multi-modal serial sectioning. *Integrating Materials and Manufacturing Innovation*, 10(2):129–141. 7.1, 7.1, 7.3, 7.3.2

Chen, B., Dao, T., Winsor, E., Song, Z., Rudra, A., and Ré, C. (2021). Scatterbrain: Unifying sparse and low-rank attention. *Advances in Neural Information Processing Systems*, 34:17413–17426. 2.2.1, 3.1, 7.3.1

Chen, C., Borgeaud, S., Irving, G., Lespiau, J.-B., Sifre, L., and Jumper, J. (2023a). Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*. 5.4.1

Chen, X., Aksitov, R., Alon, U., Ren, J., Xiao, K., Yin, P., Prakash, S., Sutton, C., Wang, X., and Zhou, D. (2023b). Universal self-consistency for large language model generation. *arXiv preprint arXiv:2311.17311*. 2.2.4

Chen, Y., Tao, Q., Tonin, F., and Suykens, J. A. (2023c). Primal-attention: Self-attention through asymmetric kernel svd in primal representation. *arXiv preprint arXiv:2305.19798*. 2.2.1

Chen, Z., Liu, H., Zhou, Y., Zheng, H., and Chen, B. (2026). Jackpot: Optimal budgeted rejection sampling for extreme actor-policy mismatch reinforcement learning. *arXiv preprint arXiv:2602.06107*. 2.2.4

Chen, Z., Qin, X., Wu, Y., Ling, Y., Ye, Q., Zhao, W. X., and Shi, G. (2025). Pass@ k training for adaptively balancing exploration and exploitation of large reasoning models. *arXiv preprint arXiv:2508.10751*. 6.5

Cheng, J., Dong, L., and Lapata, M. (2016). Long short-term memory-networks for machine reading. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 551–561. Association for Computational Linguistics. 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016 ; Conference date: 01-11-2016 Through 05-11-2016. 1.4, 2.1.1

Child, R., Gray, S., Radford, A., and Sutskever, I. (2019). Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*. 2.2.1

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., et al. (2020). Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*. 2.2.1, 2, 3.3.1, 7.3.1

Chung, H., Kim, J., Mccann, M. T., Klasky, M. L., and Ye, J. C. (2022). Diffusion posterior sampling for general noisy inverse problems. *arXiv preprint arXiv:2209.14687*. 7.4

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. (2019). Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*. 3.3.1, 4.4.1

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. (2018). Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*. 4.4.1

CMIMC (2025). Carnegie mellon informatics and mathematics competition. 6.3.1

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. (2021). Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*. 5.1, 5.4.3, 2, 6.3.1

Csordás, R., Irie, K., and Schmidhuber, J. (2023). Approximating two-layer feedforward networks for efficient transformers. *arXiv preprint arXiv:2310.10837*. 2.2.2, 4.1

Cui, L., Wu, Y., Liu, S., Zhang, Y., and Zhou, M. (2020). Mutual: A dataset for multi-turn dialogue reasoning. *arXiv preprint arXiv:2004.04494*. 3.3.1

Dai, D., Deng, C., Zhao, C., Xu, R., Gao, H., Chen, D., Li, J., Zeng, W., Yu, X., Wu, Y., et al. (2024). Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066*. 2.2.2

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. (2022a). Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359. 2.2.3

Dao, T., Fu, D. Y., Saab, K. K., Thomas, A. W., Rudra, A., and Ré, C. (2022b). Hungry hungry hippos: Towards language modeling with state space models. *arXiv preprint*

arXiv:2212.14052. 2.2.1

Dasigi, P., Lo, K., Beltagy, I., Cohan, A., Smith, N. A., and Gardner, M. (2021). A dataset of information-seeking questions and answers anchored in research papers. *arXiv preprint arXiv:2105.03011*. 4.4.1, 5.4.3

della Ventura, N., Moore, K., Echlin, M., Begley, M., Pollock, T., De Graef, M., and Gianola, D. (2026). Energy-resolved ebsd using a monolithic direct electron detector. *Ultramicroscopy*, 281:114301. 7.4

Deng, A., Cao, T., Chen, Z., and Hooi, B. (2025). Words or vision: Do vision-language models have blind faith in text? In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 3867–3876. 7.4.4

Dery, L., Kolawole, S., Kagey, J.-F., Smith, V., Neubig, G., and Talwalkar, A. (2024). Everybody prune now: Structured pruning of llms with only forward passes. *arXiv preprint arXiv:2402.05406*. 2.2.2

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. (2022). Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*. 2.2.2, 2.2.3, 4.1

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. (2023). Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115. 5.4.3

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186. 2.1.4, 7.3, 7.3.2, 7.3.2

Donegan, S., Tucker, J., Rollett, A., Barmak, K., and M, G. (2013). Extreme value analysis of tail departure from log-normality in experimental and simulated grain size distributions. *Acta Materialia*, 61(15):5595–5604. 7.3.2

Dong, H., Acun, B., Chen, B., and Chi, Y. (2026a). Scalable llm reasoning acceleration

with low-rank distillation. In *Conference on Parsimony and Learning (CPAL)*. 1.2, 1.5, 5, 5.1

Dong, H., Brandfonbrener, D., Helenowski, E., He, Y., Kumar, M., Fang, H., Chi, Y., and Sankararaman, K. A. (2026b). Generalized parallel scaling with interdependent generations. In *International Conference on Learning Representations (ICLR)*. 1.2, 1.5, 6, 6.1

Dong, H., Chen, B., and Chi, Y. (2023a). Towards structured sparsity in transformers for efficient inference. In *Workshop on Efficient Systems for Foundation Models@ ICML2023*. 1.5, 2.2.2, 4

Dong, H., Chen, B., and Chi, Y. (2024a). Prompt-prompted adaptive structured pruning for efficient llm generation. In *First Conference on Language Modeling*. 1.2, 1.5, 4, 5.1, 5.2, 5.3.1

Dong, H., Donegan, S., Shah, M., and Chi, Y. (2023b). A lightweight transformer for faster and robust ebsd data collection. *Scientific Reports*, 13(1):21253. 1.2, 1.5, 6.2.1, 7, 1, 7.4.2

Dong, H., Efimov, T., Shah, M., Simmons, J., Donegan, S., Graef, M. D., and Chi, Y. (2026c). Multimodal diffusion to mutually enhance polarized light and low resolution ebsd data. *arXiv preprint arXiv:2604.22212*. 1.5, 7, 2, 7.4

Dong, H., Johnson, T., Cho, M., and Soroush, E. (2024b). Towards low-bit communication for tensor parallel llm inference. *arXiv preprint arXiv:2411.07942*. 2.2.2, 2.2.3

Dong, H., Shah, M., Donegan, S., and Chi, Y. (2023c). Deep unfolded tensor robust pca with self-supervised learning. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE. 1.5, 7, 1, 7.3, 7.4.2

Dong, H., Tong, T., Ma, C., and Chi, Y. (2023d). Fast and provable tensor robust principal component analysis via scaled gradient descent. *Information and Inference: A Journal of the IMA*, 12(3):1716–1758. 1.5, 2.2.1

Dong, H., Yang, X., Zhang, Z., Wang, Z., Chi, Y., and Chen, B. (2024c). Get more with less: Synthesizing recurrence with kv cache compression for efficient llm inference. In

Forty-first International Conference on Machine Learning. 1.2, 1.5, 3, 5.3.1

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*. 2.1.4, 7.3

Dou, Z. and Song, Y. (2024a). Diffusion posterior sampling for linear inverse problem solving: A filtering perspective. In *The Twelfth International Conference on Learning Representations*. 7.4.1

Dou, Z. and Song, Y. (2024b). Diffusion posterior sampling for linear inverse problem solving: A filtering perspective. In *The Twelfth International Conference on Learning Representations*. 7.4.2

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. (2024). The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*. 1, 2.1.1, 2.1.3, 5.4.3, 6.3.1, 8

Efimov, T., Dong, H., Shah, M., Simmons, J., Donegan, S., and Chi, Y. (2025). Leveraging multimodal diffusion models to accelerate imaging with side information. In *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE. 1.5, 7, 7.4, 7.4.1, 7.4.2

Efimov, T., Venkatakrishnan, S., Hossain, M., Duba-Sullivan, H., and Ziabari, A. (2026). Cross-modal guidance for fast diffusion-based computed tomography. *arXiv preprint arXiv:2603.01253*. 7.4

Fabbri, A. R., Li, I., She, T., Li, S., and Radev, D. R. (2019). Multi-news: a large-scale multi-document summarization dataset and abstractive hierarchical model. 3.3.2

Face, H. (2025). Open r1: A fully open reproduction of deepseek-r1. 5.4.1

Faria, G. and Smith, N. A. (2025). Sample, don't search: Rethinking test-time alignment for language models. *arXiv preprint arXiv:2504.03790*. 2.2.4

Fatahalian, K., Sugerman, J., and Hanrahan, P. (2004). Understanding the efficiency

of gpu algorithms for matrix-matrix multiplication. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 133–137. 4.2

Fawzi, A., Balog, M., Huang, A., Hubert, T., Romera-Paredes, B., Barekatain, M., Novikov, A., R. Ruiz, F. J., Schrittwieser, J., Swirszcz, G., et al. (2022). Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53. 6.2.1

Fedus, W., Zoph, B., and Shazeer, N. (2022). Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *The Journal of Machine Learning Research*, 23(1):5232–5270. 2.2.2, 4.1

Frankle, J. and Carbin, M. (2018). The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*. 2.2.2

Frantar, E. and Alistarh, D. (2023). Massive language models can be accurately pruned in one-shot. *arXiv preprint arXiv:2301.00774*. 2.2.2

Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., et al. (2020). The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*. 3.3.1, 4.1

Gao, L., Tow, J., Abbasi, B., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., Le Noac'h, A., Li, H., McDonell, K., Muennighoff, N., Ociepa, C., Phang, J., Reynolds, L., Schoelkopf, H., Skowron, A., Sutawika, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. (2023). A framework for few-shot language model evaluation. 3.3.1, 4.4.1

Ge, S., Zhang, Y., Liu, L., Zhang, M., Han, J., and Gao, J. (2023). Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*. 2.2.1

Geva, M., Schuster, R., Berant, J., and Levy, O. (2020). Transformer feed-forward layers are key-value memories. *arXiv preprint arXiv:2012.14913*. 2.2.2, 2.2.2, 4.1

Graef, M. D. (2024). Emsoftoo. https://github.com/EMsoft-org/EMsoftOO. 1, 7.4.2

Groeber, M. A. and Jackson, M. A. (2014). Dream. 3d: a digital representation environment for the analysis of microstructure in 3d. *Integrating materials and manufacturing innovation*, 3(1):56–72. 1, 7.3.2, 7.4, 7.4.2

Gu, A. and Dao, T. (2023). Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*. 2.2.1, 8

Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al. (2025). Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*. 1, 2.1.3, 2.2.3, 2.2.4, 5.1, 5.4.1, 6.3.1, 8

Habib, N., Fourrier, C., Kydlíček, H., Wolf, T., and Tunstall, L. (2023). Lighteval: A lightweight framework for llm evaluation. 6.3.1

Hagström, L. and Johansson, R. (2022). How to adapt pre-trained vision-and-language models to a text-only input? In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 5582–5596. 7.4.4

Han, C., Wang, Q., Xiong, W., Chen, Y., Ji, H., and Wang, S. (2023). Lm-infinite: Simple on-the-fly length generalization for large language models. *arXiv preprint arXiv:2308.16137*. 2.2.1, 3.1, 3.1, 3.3

He, H. and Lab, T. M. (2025). Defeating nondeterminism in llm inference. *Thinking Machines Lab: Connectionism*. https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/. 3

He, K., Chen, X., Xie, S., Li, Y., Dollár, P., and Girshick, R. (2022). Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16000–16009. 7.3

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. (2021). Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*. 5.4.1, 5.4.3, 6.3.1

Hendrycks, D. and Gimpel, K. (2016). Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*. 7.4.2

Hermann, K. M., KociskÃœ, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. (2015). Teaching machines to read and comprehend. In *NIPS*, pages 1693–1701. 3.1, 3.3.2, 5.4.3, 6.3.1

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*. 2.2.3

HMMT (2025). Hmmt. 6.3.1

Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851. 7.4, 7.4.1, 7.4.1

Ho, J., Kalchbrenner, N., Weissenborn, D., and Salimans, T. (2019). Axial attention in multidimensional transformers. *arXiv preprint arXiv:1912.12180*. 6.2.1, 1, 7.3.1

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., et al. (2022). Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*. 1, 2.2.4

Hsu, C.-J., Buffelli, D., McGowan, J., Liao, F.-T., Chen, Y.-C., Vakili, S., and Shiu, D.-s. (2025). Group think: Multiple concurrent reasoning agents collaborating at token level granularity. *arXiv preprint arXiv:2505.11107*. 2.2.4, 6.1

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. (2022). Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3. 5.3.5

Huynh, D. Q. (2009). Metrics for 3d rotations: Comparisons and analysis. *Journal of Mathematical Imaging and Vision*, 35:155–164. 7.3.2

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr. 6.1

Islam, K. T., Zhong, S., Zakavi, P., Chen, Z., Kavnoudias, H., Farquharson, S., Durbridge, G., Barth, M., McMahon, K. L., Parizel, P. M., et al. (2023). Improving portable low-field mri image quality through image-to-image translation using paired low-and high-field images. *Scientific Reports*, 13(1):21183. 7.4

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87. 2.2.2

Jain, S., Lanchantin, J., Nickel, M., Ullrich, K., Wilson, A., and Watson-Daniels, J. (2025). Llm output homogenization is task dependent. *arXiv preprint arXiv:2509.21267*. 6.5

Jaiswal, A. K., Liu, S., Chen, T., Ding, Y., and Wang, Z. (2023). Instant soup: Cheap pruning ensembles in a single pass can draw lottery tickets from large models. In *International Conference on Machine Learning*, pages 14691–14701. PMLR. 2.2.2

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. d. l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al. (2023). Mistral 7b. *arXiv preprint arXiv:2310.06825*. 3

Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., Chaplot, D. S., de las Casas, D., Hanna, E. B., Bressand, F., Lengyel, G., Bour, G., Lample, G., Lavaud, L. R., Saulnier, L., Lachaux, M.-A., Stock, P., Subramanian, S., Yang, S., Antoniak, S., Scao, T. L., Gervet, T., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E. (2024). Mixtral of experts. 2.2.2, 4.1

Jin, K.-W. and De Graef, M. (2018). Correlation of c-axis orientation of a-titanium grains with polarized light optical microscopy intensity profiles. *Microscopy and Microanalysis*, 24(S1):548–549. 7.4

Jin, K.-W. and De Graef, M. (2020). c-axis orientation determination of α-titanium using computational polarized light microscopy. *Materials Characterization*, 167:110503. 7.4

Jin, T., Cheng, E. Y., Ankner, Z., Saunshi, N., Elias, B. M., Yazdanbakhsh, A., Ragan-Kelley, J., Subramanian, S., and Carbin, M. (2025). Learning to keep a promise: Scaling language model decoding parallelism with learned asynchronous decoding. *arXiv preprint arXiv:2502.11517*. 2.2.4, 6.1

Jolley, B. R., Uchic, M. D., Sparkman, D., Chapman, M., and Schwalbach, E. J. (2021). Application of Serial Sectioning to Evaluate the Performance of x-ray Computed Tomography for Quantitative Porosity Measurements in Additively Manufactured Metals. *JOM*. 7.1

Kahneman, D. (2011). *Thinking, fast and slow*. macmillan. 1.4

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. (2020). Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*. 1, 2.2.4

Kasai, J., Peng, H., Zhang, Y., Yogatama, D., Ilharco, G., Pappas, N., Mao, Y., Chen, W., and Smith, N. A. (2021). Finetuning pretrained transformers into rnns. *arXiv preprint arXiv:2103.13076*. 2.2.1

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. (2020). Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR. 2.2.1, 2

Khan, S., Naseer, M., Hayat, M., Zamir, S. W., Khan, F. S., and Shah, M. (2022). Transformers in vision: A survey. *ACM computing surveys (CSUR)*, 54(10s):1–41. 7.3, 7.3.1

Khandelwal, U., He, H., Qi, P., and Jurafsky, D. (2018). Sharp nearby, fuzzy far away: How neural language models use context. *arXiv preprint arXiv:1805.04623*. 2.2.1

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. 3.2.1, 7.4.2

Kong, L., Ma, M. Q., Chen, G., Xing, E. P., Chi, Y., Morency, L.-P., and Zhang, K. (2023). Understanding masked autoencoders via hierarchical latent variable models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7918–7928. 7.3

Kotula, P. G., Keenan, M. R., and Michael, J. R. (2006). Tomographic spectral imaging with multivariate statistical analysis: comprehensive 3d microanalysis. *Microscopy and Microanalysis*, 12(1):36–48. 7.1

Lambert, N., Morrison, J., Pyatkin, V., Huang, S., Ivison, H., Brahman, F., Miranda, L. J. V., Liu, A., Dziri, N., Lyu, S., et al. (2024). Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*. 2.1.3

Larsen, P. M. and Schmidt, S. (2017). Improved orientation sampling for indexing

diffraction patterns of polycrystalline materials. *Journal of Applied Crystallography*, 50(6):1571–1582. 3, 7.4.3

Latif, S., Zaidi, A., Cuayahuitl, H., Shamshad, F., Shoukat, M., and Qadir, J. (2023). Transformers in speech processing: A survey. *arXiv preprint arXiv:2303.11607*. 7.3

LeCun, Y., Denker, J., and Solla, S. (1989). Optimal brain damage. *Advances in neural information processing systems*, 2. 2.2.2

Lee, D., Lee, J.-Y., Zhang, G., Tiwari, M., and Mirhoseini, A. (2024). Cats: Contextually-aware thresholding for sparsity in large language models. *arXiv preprint arXiv:2404.08763*. 5.1, 5.2, 5.3.1

Leviathan, Y., Kalman, M., and Matias, Y. (2023). Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR. 2.2.3, 2.2.4, 5.4.1

Li, Y., Yu, Y., Zhang, Q., Liang, C., He, P., Chen, W., and Zhao, T. (2023). Losparse: Structured compression of large language models based on low-rank and sparse approximation. *arXiv preprint arXiv:2306.11222*. 2.2.1, 2.2.2, 4.1

Li, Z., You, C., Bhojanapalli, S., Li, D., Rawat, A. S., Reddi, S. J., Ye, K., Chern, F., Yu, F., Guo, R., et al. (2022). The lazy neuron phenomenon: On emergence of activation sparsity in transformers. In *The Eleventh International Conference on Learning Representations*. 2.2.2, 4.1

Liang, T., Glossner, J., Wang, L., Shi, S., and Zhang, X. (2021). Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403. 2.2.2

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. (2023). Let's verify step by step. *arXiv preprint arXiv:2305.20050*. 5.1, 5.4.1, 6.3.1

Lin, B. Y., Bras, R. L., Richardson, K., Sabharwal, A., Poovendran, R., Clark, P., and Choi, Y. (2025). Zebralogic: On the scaling limits of llms for logical reasoning. *arXiv preprint arXiv:2502.01100*. 6.3.1

Lin, C.-Y. (2004a). ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics. 3.3.2

Lin, C.-Y. (2004b). Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81. 6.4

Lin, J., Tang, J., Tang, H., Yang, S., Chen, W.-M., Wang, W.-C., Xiao, G., Dang, X., Gan, C., and Han, S. (2024). Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6:87–100. 2.2.3

Liu, B., Zhang, Z., He, P., Wang, Z., Xiao, Y., Ye, R., Zhou, Y., Ku, W.-S., and Hui, B. (2024). A survey of lottery ticket hypothesis. *arXiv preprint arXiv:2403.04861*. 2.2.2

Liu, J., Liu, H., Xiao, L., Wang, Z., Liu, K., Gao, S., Zhang, W., Zhang, S., and Chen, K. (2025). Are your llms capable of stable reasoning? 6.1, 6.3.3

Liu, R., Deng, H., Huang, Y., Shi, X., Lu, L., Sun, W., Wang, X., Dai, J., and Li, H. (2021a). Decoupled spatial-temporal transformer for video inpainting. *arXiv preprint arXiv:2104.06637*. 7.3

Liu, Y., Li, H., Du, K., Yao, J., Cheng, Y., Huang, Y., Lu, S., Maire, M., Hoffmann, H., Holtzman, A., et al. (2023a). Cachegen: Fast context loading for language model applications. *arXiv preprint arXiv:2310.07240*. 2.2.1

Liu, Z., Desai, A., Liao, F., Wang, W., Xie, V., Xu, Z., Kyrillidis, A., and Shrivastava, A. (2023b). Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *arXiv preprint arXiv:2305.17118*. 2.2.1, 3.1

Liu, Z., Li, F., Li, G., and Cheng, J. (2021b). Ebert: Efficient bert inference with dynamic structured pruning. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 4814–4823. 2.2.2

Liu, Z., Wang, J., Dao, T., Zhou, T., Yuan, B., Song, Z., Shrivastava, A., Zhang, C., Tian, Y., Re, C., et al. (2023c). Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pages 22137–22176. PMLR.

2.2.2, 2.2.2, 4.1

Lowekamp, B. C., Chen, D. T., Ibáñez, L., and Blezek, D. (2013). The design of simpleitk. *Frontiers in neuroinformatics*, 7:45. 7.4.2

Lugmayr, A., Danelljan, M., Romero, A., Yu, F., Timofte, R., and Van Gool, L. (2022). Repaint: Inpainting using denoising diffusion probabilistic models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11461–11471. 7.4

Luo, M., Tan, S., Wong, J., Shi, X., Tang, W., Roongta, M., Cai, C., Luo, J., Zhang, T., Li, E., Popa, R. A., and Stoica, I. (2025). Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl. Notion Blog. 6.3.1

Lyu, Q. and Wang, G. (2022). Conversion between ct and mri images using diffusion and score-matching models. *arXiv preprint arXiv:2209.12104*. 7.4

Ma, X., Fang, G., and Wang, X. (2023). Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720. 2.2.2, 4.1

Menasche, D. B., Musinski, W. D., Obstalecki, M., Shah, M. N., Donegan, S. P., Bernier, J. V., Kenesei, P., Park, J.-S., and Shade, P. A. (2021). Afrl additive manufacturing modeling series: challenge 4, in situ mechanical test of an in625 sample with concurrent high-energy diffraction microscopy characterization. *Integrating Materials and Manufacturing Innovation*, 10(3):338–347. 7.3.2, 7.3.3

Merity, S., Xiong, C., Bradbury, J., and Socher, R. (2016). Pointer sentinel mixture models. 3.3, 3.3.1, 4.3.1

Mirzadeh, I., Alizadeh, K., Mehta, S., Del Mundo, C. C., Tuzel, O., Samei, G., Rastegari, M., and Farajtabar, M. (2023). Relu strikes back: Exploiting activation sparsity in large language models. *arXiv preprint arXiv:2310.04564*. 2.2.2

Muennighoff, N., Yang, Z., Shi, W., Li, X. L., Fei-Fei, L., Hajishirzi, H., Zettlemoyer, L., Liang, P., Candès, E., and Hashimoto, T. B. (2025). s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*,

pages 20286–20332. 2.2.4

Nallapati, R., Zhou, B., Gulcehre, C., Xiang, B., et al. (2016). Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*. 4.4.1

Naragani, D., Sangid, M. D., Shade, P. A., Schuren, J. C., Sharma, H., Park, J.-S., Kenesei, P., Bernier, J. V., Turner, T. J., and Parr, I. (2017). Investigation of fatigue crack initiation from a non-metallic inclusion via high energy x-ray diffraction microscopy. *Acta Materialia*, 137:71–84. 7.1

Narayan, S., Cohen, S. B., and Lapata, M. (2018). Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *arXiv preprint arXiv:1808.08745*. 3.1, 3.3.2, 4.4.1, 5.4.3, 6.3.1

Narayanan, D., Shoeybi, M., Casper, J., LeGresley, P., Patwary, M., Korthikanti, V., Vainbrand, D., Kashinkunti, P., Bernauer, J., Catanzaro, B., et al. (2021). Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pages 1–15. 2.2.3

Nawrot, P., Łańcucki, A., Chochowski, M., Tarjan, D., and Ponti, E. M. (2024). Dynamic memory compression: Retrofitting llms for accelerated inference. *arXiv preprint arXiv:2403.09636*. 2.2.1

Nguyen, L. T. and Rowenhorst, D. J. (2021). The Alignment and Fusion of Multimodal 3D Serial Sectioning Datasets. *JOM*, 73(11):3272–3284. 7.1

Nikdan, M., Tabesh, S., and Alistarh, D. (2024). Rosa: Accurate parameter-efficient fine-tuning via robust adaptation. *arXiv preprint arXiv:2401.04679*. 2.2.1

Oren, M., Hassid, M., Adi, Y., and Schwartz, R. (2024). Transformers are multi-state rnns. *arXiv preprint arXiv:2401.06104*. 2.2.1, 3.3

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. (2022). Training language models to follow instructions with human feedback. *Advances in neural information processing systems*,

35:27730–27744. 2.1.3

Pan, J., Li, X., Lian, L., Snell, C., Zhou, Y., Yala, A., Darrell, T., Keutzer, K., and Suhr, A. (2025a). Learning adaptive parallel reasoning with language models. *arXiv preprint arXiv:2504.15466*. 6.1

Pan, J., Zhang, J., Wang, X., Yuan, L., Peng, H., and Suhr, A. (2025b). Tinyzero. https://github.com/Jiayi-Pan/TinyZero. Accessed: 2025-01-24. 6.3.1

Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, L., Shazeer, N., Ku, A., and Tran, D. (2018). Image transformer. In *International conference on machine learning*, pages 4055–4064. PMLR. 2.2.1

Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., and Efros, A. A. (2016). Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2536–2544. 7.3

Peebles, W. and Xie, S. (2023). Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4195–4205. 7.4

Peng, B., Alcaide, E., Anthony, Q., Albalak, A., Arcadinho, S., Cao, H., Cheng, X., Chung, M., Grella, M., GV, K. K., et al. (2023). Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*. 2.2.1

Peng, H., Pappas, N., Yogatama, D., Schwartz, R., Smith, N. A., and Kong, L. (2021). Random feature attention. *arXiv preprint arXiv:2103.02143*. 2.2.1

Piórczyński, M., Szatkowski, F., Bałazy, K., and Wójcik, B. (2023). Exploiting transformer activation sparsity with dynamic inference. *arXiv preprint arXiv:2310.04361*. 2.2.2

Poli, M., Massaroli, S., Nguyen, E., Fu, D. Y., Dao, T., Baccus, S., Bengio, Y., Ermon, S., and Ré, C. (2023). Hyena hierarchy: Towards larger convolutional language models. *arXiv preprint arXiv:2302.10866*. 2.2.1

Polonsky, A. T., Lang, C. A., Kvilekval, K. G., Latypov, M. I., Echlin, M. P., Manjunath, B. S., and Pollock, T. M. (2019). Three-dimensional Analysis and Reconstruction of Additively Manufactured Materials in the Cloud-Based BisQue Infrastructure. *Integrating*

Materials and Manufacturing Innovation, 8(1):37–51. 7.1

Polonsky, A. T., Lenthe, W. C., Echlin, M. P., Livescu, V., Gray, G. T. I., and Pollock, T. M. (2020). Solidification-driven orientation gradients in additively manufactured stainless steel. *Acta Materialia*, 183:249–260. 7.3.2

Polonsky, A. T., Raghavan, N., Echlin, M. P., Kirka, M. M., Dehoff, R. R., and Pollock, T. M. (2022). Scan strategies in EBM-printed IN718 and the physics of bulk 3D microstructure development. *Materials Characterization*, 190:112043. 7.1

Qi, J., Ye, X., Tang, H., Zhu, Z., and Choi, E. (2025). Learning to reason across parallel samples for llm reasoning. *arXiv preprint arXiv:2506.09014*. 2.2.4

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. (2018). Improving language understanding by generative pre-training. 2.1.1

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9. 2.1.1, 8

Rae, J. W., Potapenko, A., Jayakumar, S. M., Hillier, C., and Lillicrap, T. P. (2019). Compressive transformers for long-range sequence modelling. *arXiv preprint*. 3.3.1, 4.1

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv e-prints*. 3.2.1

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67. 2.1.4

Rahimi, A. and Recht, B. (2007). Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20. 3.3.1

Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., and Chen, M. (2022). Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 1(2):3. 7.4

Rastogi, A., Jiang, A. Q., Lo, A., Berrada, G., Lample, G., Rute, J., Barmentlo, J.,

Yadav, K., Khandelwal, K., Chandu, K. R., et al. (2025). Magistral. *arXiv preprint arXiv:2506.10910*. 2.2.4

Ravi, N., Reizenstein, J., Novotny, D., Gordon, T., Lo, W.-Y., Johnson, J., and Gkioxari, G. (2020). Accelerating 3d deep learning with pytorch3d. *arXiv:2007.08501*. 7.4.3

Reddy, S., Chen, D., and Manning, C. D. (2019). Coqa: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266. 4.4.1, 5.4.3

Rein, D., Hou, B. L., Stickland, A. C., Petty, J., Pang, R. Y., Dirani, J., Michael, J., and Bowman, S. R. (2024). Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*. 5.4.1, 6.3.1

Renggli, C., Pinto, A. S., Houlsby, N., Mustafa, B., Puigcerver, J., and Riquelme, C. (2022). Learning to merge tokens in vision transformers. *arXiv preprint arXiv:2202.12015*. 2.2.1

Rodionov, G., Garipov, R., Shutova, A., Yakushev, G., Schultheis, E., Egiazarian, V., Sinitsin, A., Kuznedelev, D., and Alistarh, D. (2025). Hogwild! inference: Parallel llm generation via concurrent attention. *arXiv preprint arXiv:2504.06261*. 2.2.4, 6.1

Roemmele, M., Bejan, C. A., and Gordon, A. S. (2011). Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *2011 AAAI Spring Symposium Series*. 4.4.1

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695. 7.4

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer. 7.4.2

Roşca, D., Morawiec, A., and De Graef, M. (2014). A new method of constructing a grid in the space of 3d rotations and its applications to texture analysis. *Modelling and Simulation in Materials Science and Engineering*, 22(7):075013. 2, 7.3.2, 7.4.2

Rudin, L. I., Osher, S., and Fatemi, E. (1992). Nonlinear total variation based noise removal algorithms. *Physica D: nonlinear phenomena*, 60(1-4):259–268. 7.4

Sadhukhan, R., Cao, S., Dong, H., Zhao, C., Purpura-Pontoniere, A., Tian, Y., Liu, Z., and Chen, B. (2026). Stem: Scaling transformers with embedding modules. *arXiv preprint arXiv:2601.10639*. 2.2.2, 4.1

Saguy, A., Nahimov, T., Lehrman, M., Gómez-de Mariscal, E., Hidalgo-Cenalmor, I., Alalouf, O., Balakrishnan, A., Heilemann, M., Henriques, R., and Shechtman, Y. (2025). This microtubule does not exist: Super-resolution microscopy image generation by a diffusion model. *Small Methods*, 9(3):2400672. 7.4

Samsi, S., Zhao, D., McDonald, J., Li, B., Michaleas, A., Jones, M., Bergeron, W., Kepner, J., Tiwari, D., and Gadepally, V. (2023). From words to watts: Benchmarking the energy costs of large language model inference. In *2023 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–9. IEEE. 1

Sandgren, H. R., Zhai, Y., Lados, D. A., Shade, P. A., Schuren, J. C., Groeber, M. A., Kenesei, P., and Gavras, A. G. (2016). Characterization of fatigue crack growth behavior in lens fabricated ti-6al-4v using high-energy synchrotron x-ray microtomography. *Additive Manufacturing*, 12:132–141. 7.1

Santacroce, M., Wen, Z., Shen, Y., and Li, Y. (2023). What matters in the structured pruning of generative language models? *arXiv preprint arXiv:2302.03773*. 2.2.2, 4.1

Satopaa, V., Albrecht, J., Irwin, D., and Raghavan, B. (2011). Finding a" kneedle" in a haystack: Detecting knee points in system behavior. In *2011 31st international conference on distributed computing systems workshops*, pages 166–171. IEEE. 7.4.2

See, A., Liu, P. J., and Manning, C. D. (2017). Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics. 3.1, 3.3.2, 5.4.3, 6.3.1

Shade, P. A., Musinski, W. D., Shah, M. N., Uchic, M. D., Donegan, S. P., Chapman, M. G., Park, J.-S., Bernier, J. V., Kenesei, P., Menasche, D. B., Obstalecki, M., Schwalbach,

E. J., Miller, J. D., Groeber, M. A., and Cox, M. E. (2019). Afrl am modeling challenge series: Challenge 4 data package. 7.3.2, 7.3.3

Shaham, U., Segal, E., Ivgi, M., Efrat, A., Yoran, O., Haviv, A., Gupta, A., Xiong, W., Geva, M., Berant, J., et al. (2022). Scrolls: Standardized comparison over long language sequences. *arXiv preprint arXiv:2201.03533*. 4.4.1

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y., Wu, Y., et al. (2024). Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*. 6.2.3

Shazeer, N. (2019). Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*. 2.1.1, 3.2.1

Shazeer, N. (2020). Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*. 2.1.1, 2.2.2, 3

Sheng, Y., Zheng, L., Yuan, B., Li, Z., Ryabinin, M., Fu, D. Y., Xie, Z., Chen, B., Barrett, C. W., Gonzalez, J., Liang, P., Ré, C., Stoica, I., and Zhang, C. (2023). High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*. 3.3.3

Singh, A., Fry, A., Perelman, A., Tart, A., Ganesh, A., El-Kishky, A., McLaughlin, A., Low, A., Ostrow, A., Ananthram, A., et al. (2025). Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*. 8

Snell, C., Lee, J., Xu, K., and Kumar, A. (2024). Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*. 2.2.4, 5.1

Sobel, I., Feldman, G., et al. (1968). A 3x3 isotropic gradient operator for image processing. *a talk at the Stanford Artificial Project in*, 1968:271–272. 7.4.2

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. (2020). Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*. 7.4, 7.4.1

Stinville, J., Hestroffer, J., Charpagne, M., Polonsky, A., Echlin, M., Torbet, C., Valle, V., Loghin, A., Klaas, O., Miller, M., Nygren, K., Beyerlein, I., and Pollock, T. (2022a). Multi-modal dataset of a polycrystalline metallic material: 3d microstructure and deformation fields. 7.3.2, 7.3.3

Stinville, J., Hestroffer, J., Charpagne, M., Polonsky, A., Echlin, M., Torbet, C., Valle, V., Nygren, K., Miller, M., Klaas, O., et al. (2022b). Multi-modal dataset of a polycrystalline metallic material: 3d microstructure and deformation fields. *Scientific Data*, 9(1):460. 7.1, 7.3.2, 7.3.2, 7.3.3

Strubell, E., Ganesh, A., and McCallum, A. (2020). Energy and policy considerations for modern deep learning research. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 13693–13696. 1

Sun, H., Chang, L.-W., Bao, W., Zheng, S., Zheng, N., Liu, X., Dong, H., Chi, Y., and Chen, B. (2025). Shadowkv: Kv cache in shadows for high-throughput long-context llm inference. In *International Conference on Machine Learning*, pages 57355–57373. PMLR. 2.2.1

Sun, M., Liu, Z., Bair, A., and Kolter, J. Z. (2023a). A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*. 2.2.2, 4.4.1

Sun, Y., Dong, L., Huang, S., Ma, S., Xia, Y., Xue, J., Wang, J., and Wei, F. (2023b). Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*. 2.2.1

Tay, Y., Dehghani, M., Bahri, D., and Metzler, D. (2022). Efficient transformers: A survey. 2.2.1, 7.3.1

Team, G., Georgiev, P., Lei, V. I., Burnell, R., Bai, L., Gulati, A., Tanzer, G., Vincent, D., Pan, Z., Wang, S., et al. (2024a). Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*. 8

Team, G., Mesnard, T., Hardin, C., Dadashi, R., Bhupatiraju, S., Pathak, S., Sifre, L., Rivière, M., Kale, M. S., Love, J., et al. (2024b). Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*. 3

Team, G., Riviere, M., Pathak, S., Sessa, P. G., Hardin, C., Bhupatiraju, S., Hussenot, L., Mesnard, T., Shahriari, B., Ramé, A., et al. (2024c). Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*. 1, 2.1.1, 5.1, 5.4.3

Team, K., Bai, Y., Bao, Y., Chen, G., Chen, J., Chen, N., Chen, R., Chen, Y., Chen, Y., Chen, Y., et al. (2025). Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*. 2.2.4, 8

Team, S. (2023). Sparse large language models with relu activation. 3

Teferra, K. and Rowenhorst, D. J. (2021). Optimizing the cellular automata finite element model for additive manufacturing to simulate large microstructures. *Acta Materialia*, page 116930. 7.1

Tong, T., Ma, C., and Chi, Y. (2021). Accelerating ill-conditioned low-rank matrix estimation via scaled gradient descent. *Journal of Machine Learning Research*, 22(150):1–63. 2.2.1

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. (2023). Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*. 3.1, 3.2.1, 3

Tsai, Y.-H. H., Bai, S., Yamada, M., Morency, L.-P., and Salakhutdinov, R. (2019). Transformer dissection: a unified understanding of transformer's attention via the lens of kernel. *arXiv preprint arXiv:1908.11775*. 2.2.1, 2

Uchic, M., Groeber, M., Shah, M., Callahan, P., Shiveley, A., Scott, M., Chapman, M., and Spowart, J. (2016). An Automated Multi-Modal Serial Sectioning System for Characterization of Grain-Scale Microstructures in Engineering Materials. In De Graef, M., Poulsen, H. F., Lewis, A., Simmons, J., and Spanos, G., editors, *Proceedings of the 1st International Conference on 3D Materials Science*, pages 195–202, Cham. Springer International Publishing. 7.1

Varley, Z. (2024). ebsdtorch. https://github.com/ZacharyVarley/ebsdtorch. 3, 7.4.3

Varley, Z., Rohrer, G., and M., D. G. (2026). Accelerating dictionary indexing of electron backscatter diffraction patterns with pca and quantization. *Scientific Reports*, 16:4382.

7.4

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30. 1, 2.1.1, 2.1.4

Verine, A., Pydi, M. S., Negrevergne, B., and Chevaleyre, Y. (2024). Optimal budgeted rejection sampling for generative models. In *International Conference on Artificial Intelligence and Statistics*, pages 3367–3375. PMLR. 2.2.4

Vincent, P. (2011). A connection between score matching and denoising autoencoders. *Neural computation*, 23(7):1661–1674. 7.4.1

Wang, H., Zhu, Y., Green, B., Adam, H., Yuille, A., and Chen, L.-C. (2020a). Axial-deeplab: Stand-alone axial-attention for panoptic segmentation. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IV*, pages 108–126. Springer. 7.3.1

Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. (2020b). Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*. 7.3.1

Wang, W., Tran, D., and Feiszli, M. (2020c). What makes training multi-modal classification networks hard? In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12695–12705. 7.4.4

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. (2022). Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*. 2.2.4, 6.4.3

Wang, Y. E., Wei, G.-Y., and Brooks, D. (2019). Benchmarking tpu, gpu, and cpu platforms for deep learning. *arXiv preprint arXiv:1907.10701*. 4.2

Webber, G. and Reader, A. J. (2024). Diffusion models for medical image reconstruction. *BJR| Artificial Intelligence*, 1(1):ubae013. 7.4

Wei, J., Bosma, M., Zhao, V. Y., Guu, K., Yu, A. W., Lester, B., Du, N., Dai, A. M., and Le, Q. V. (2021). Finetuned language models are zero-shot learners. *arXiv preprint*

arXiv:2109.01652. 2.1.3

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837. 2.2.4, 5.1, 6.1

Wilson, J. R., Kobsiriphat, W., Mendoza, R., Chen, H.-Y., Hiller, J. M., Miller, D. J., Thornton, K., Voorhees, P. W., Adler, S. B., and Barnett, S. A. (2006). Three-dimensional reconstruction of a solid-oxide fuel-cell anode. *Nature materials*, 5(7):541–544. 7.1

Wu, R., Wang, H., Chen, H.-T., and Carneiro, G. (2024a). Deep multimodal learning with missing modality: A survey. *arXiv preprint arXiv:2409.07825*. 7.4.4

Wu, Y., Sun, Z., Li, S., Welleck, S., and Yang, Y. (2024b). An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*. 5.1

Xia, M., Gao, T., Zeng, Z., and Chen, D. (2023). Sheared llama: Accelerating language model pre-training via structured pruning. *arXiv preprint arXiv:2310.06694*. 2.2.2, 4.1

Xia, M., Zhong, Z., and Chen, D. (2022). Structured pruning learns compact and accurate models. *arXiv preprint arXiv:2204.00408*. 2.2.2, 4.1

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. (2023). Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*. 2.2.1, 3.1, 3.1, 3.3

Xiong, Y., Zeng, Z., Chakraborty, R., Tan, M., Fung, G., Li, Y., and Singh, V. (2021). Nyströmformer: A nyström-based algorithm for approximating self-attention. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 14138–14148. 7.3.1

Xu, X. and Chi, Y. (2024). Provably robust score-based diffusion posterior sampling for plug-and-play image reconstruction. *arXiv preprint arXiv:2403.17042*. 7.4, 7.4.1

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., et al. (2025a). Qwen3 technical report. *arXiv preprint arXiv:2505.09388*. 2.1.1, 2.2.4

Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., et al. (2024a). Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*. 5.4.1, 6.3.1, 8

Yang, S., Kautz, J., and Hatamizadeh, A. (2024b). Gated delta networks: Improving mamba2 with delta rule. *arXiv preprint arXiv:2412.06464*. 8

Yang, X., An, Y., Liu, H., Chen, T., and Chen, B. (2025b). Multiverse: Your language models secretly decide how to parallelize and merge generation. *arXiv preprint arXiv:2506.09991*. 2.2.4, 6.1

Yaniv, Z., Lowekamp, B. C., Johnson, H. J., and Beare, R. (2018). Simpleitk image-analysis notebooks: a collaborative environment for education and reproducible research. *Journal of digital imaging*, 31(3):290–303. 7.4.2

Yao, F., Liu, L., Zhang, D., Dong, C., Shang, J., and Gao, J. (2025). Your efficient rl framework secretly brings you off-policy rl training. *Feng Yao's Notion*. 3, 2.2.4

Yerram, V., You, C., Bhojanapalli, S., Kumar, S., Jain, P., Netrapalli, P., et al. (2024). Hire: High recall approximate top-k estimation for efficient llm inference. *arXiv preprint arXiv:2402.09360*. 2.2.2

Yin, L., Jaiswal, A., Liu, S., Kundu, S., and Wang, Z. (2024). Pruning small pre-trained weights irreversibly and monotonically impairs "difficult" downstream tasks in llms. 4.4.1

Yu, Q., Zhang, Z., Zhu, R., Yuan, Y., Zuo, X., Yue, Y., Dai, W., Fan, T., Liu, G., Liu, L., et al. (2025). Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*. 6.2.3

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. (2019). Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*. 4.4.1

Zhang, S., Fan, R., Liu, Y., Chen, S., Liu, Q., and Zeng, W. (2023a). Applications of transformer-based language models in bioinformatics: A survey. *Bioinformatics Advances*. 7.3

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li,

X., Lin, X. V., Mihaylov, T., Ott, M., Shleifer, S., Shuster, K., Simig, D., Koura, P. S., Sridhar, A., Wang, T., and Zettlemoyer, L. (2022). Opt: Open pre-trained transformer language models. 2.1.1, 2.2.2, 4.1, 3, 5.2, 8

Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., and Artzi, Y. (2019). Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*. 6.4.5

Zhang, Z., Lin, Y., Liu, Z., Li, P., Sun, M., and Zhou, J. (2021). Moefication: Transformer feed-forward layers are mixtures of experts. *arXiv preprint arXiv:2110.01786*. 2.2.2, 2.2.2, 4.1

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., et al. (2023b). H _2 o: Heavy-hitter oracle for efficient generative inference of large language models. *arXiv preprint arXiv:2306.14048*. 2.2.1, 3.1, 3.1, 3.2.1, 3.3

Zhao, W., Aggarwal, P., Saha, S., Celikyilmaz, A., Weston, J., and Kulikov, I. (2025). The majority is not always right: Rl training for solution aggregation. *arXiv preprint arXiv:2509.06870*. 2.2.4

Zheng, H., Bai, X., Chen, B., Lai, F., and Prakash, A. (2024). Learn to be efficient: Build structured sparsity in large language models. *arXiv preprint arXiv:2402.06126*. 2.2.2, 2.2.2

Zhou, Y., Chen, Z., Xu, Z., Lin, V., and Chen, B. (2024). Sirius: Contextual sparsity with correction for efficient llms. *arXiv preprint arXiv:2409.03856*. 5.1