Dissecting LLMs: From Component Attribution to Actionable Insights

by

Ting-Yun Chang

A Dissertation Presented to the
FACULTY OF THE USC GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

May 2026

Copyright 2026 Ting-Yun Chang

Acknowledgments

I would like to thank my co-advisors, Robin Jia and Jesse Thomason, for always being supportive of my research. I also thank my thesis and qualification committee members: Xiang Ren, Samuel Nastase, Jieyu Zhao, Morteza Dehghani, and Swabha Swayamdipta. I thank my coauthors, Harvey Fu, Deqing Fu, Tejas Srinivasan, Muru Zhang, Chenghao Yang, and Leticia Pinto-Alva, for the late nights we spent together meeting paper deadlines. I thank my amazing labmates and friends: Abrar Anwar, Gustavo Lucas Carvalho, Ta-Chung Chi, Ting-Rui Chiang, Tsu-Jui Fu, Ameya Godbole, Brihi Joshi, Lee Kezar, Chia-Hsuan Lee, Yenting Lin, Alexander Liu, Ollie Liu, Ziyi Liu, Yejia Liu, Ishika Singh, Yi-Lin Tuan, Ryan Wang, Yau-Shian Wang, Johnny Wei, Lorena Yan, Yuqing Yang, Qinyuan Ye, Grace Zhang, Jesse Zhang, Tianyi Zhou, and Wang Zhu. I thank my internship hosts at Nvidia, Google, and Amazon. I thank USC ping pong posse, Song Jeho table tennis club, and Swan ping pong club for all the smashing time we spent together. Finally, I want to thank my extended family in the US. I would not have had the courage to keep going on my PhD journey without the love and care of my great-uncle.

Table of Contents

List of Tables

List of Figures

Abstract

Large language models (LLMs) have achieved remarkable performance across a wide range of tasks; however, their behaviors emerge from billions of parameters whose individual roles are unclear. When an LLM exhibits undesirable behaviors, such as high sensitivity to minor prompt variations, it is difficult for researchers to diagnose or correct the underlying issues. This thesis addresses these challenges through *component attribution*, a mechanistic interpretability approach that traces the causal effect of individual model units on a target behavior. By attributing model outputs to specific neurons, attention heads, or layers, component attribution opens the LLM black box by decomposing the model into analyzable units, enabling more precise and effective model interventions.

This thesis presents four studies, approaching different research problems through component analysis. First, we introduce two benchmarks for evaluating whether existing component attribution methods can truly identify the LLM components responsible for verbatim memorization. Our benchmarks reveal that network pruning methods achieve the strongest localization, yet even the best methods identify neurons that are shared across memorized sequences, making precise unlearning of a single sequence an open challenge. Second, we decompose LLM outputs into individual component contributions to understand in-context learning, discovering good-performing components that individually outperform the full model and bad-performing components that hurt accuracy. This finding motivates component reweighting, a lightweight method that re-scales component activations from a handful of labeled examples, improving over standard 24-shot in-context learning by up to 6% accuracy points. Third, we analyze why certain inputs suffer disproportionately large errors under 3–4 bit quantization, showing that small residual stream magnitudes cause progressive error amplification through RMS LayerNorm, and that MLP gate outputs are the most critical components for preserving model quality. Finally, we work on KV cache com-

pression methods for reasoning models to improve inference-time efficiency in both memory and throughput, treating each KV pair as a component. We identify that a small fraction of value states have abnormally large magnitudes, and evicting them causes catastrophic failure where models enter repetitive reasoning loops. Based on these findings, we propose **V**alue-**a**ware **S**tochastic KV Cache **E**viction, a training-free recipe that protects large-magnitude value states and promotes diverse eviction decisions. Across six reasoning tasks, our method on Qwen3 models improves over SOTA eviction methods by 5% on average.

In summary, this work establishes component attribution as a fundamental technique to trace the causal effect of the internal model units on a model behavior, enabling actionable insights that improve model reliability and performance.

Chapter 1

Introduction

1.1 Recent Successes in Large Language Models

The past decade has witnessed a remarkable transformation in natural language processing, driven by the scaling of language models [1, 2, 3, 4, 5]. The introduction of the Transformer architecture [6] established self-attention as a powerful mechanism for capturing long-range dependencies in text. Building on this foundation, GPT-3 [5] demonstrated that a large language model (LLM) trained on a diverse web corpus could achieve strong accuracy across a wide range of language tasks simply by framing them as text completion problems, without any task-specific fine-tuning. The model can generalize to new tasks at test time by providing a handful of examples as demonstrations in the input prompt, a paradigm known as in-context learning (ICL) [5]. The open-source release of the model weights [7, 8, 9] democratized access to strong LLMs, enabling widespread research and deployment.

As LLMs grow in size and capability, deploying them efficiently has become an important practical concern. Weight quantization reduces the effective model size by lowering the precision of model weights, making it possible to run large models on consumer hardware [10, 11, 12]. KV cache compression [13, 14, 15] condenses the stored keys and values of previous tokens, allowing faster inference and longer conversation contexts without exhausting GPU memory. Taken together, these advances have made it practical to deploy powerful models

that were unimaginable just a few years ago, and they have unlocked applications ranging from question answering to code generation and scientific discovery.

1.2 Open Problems in LLM Behavior

Despite their impressive capabilities, LLMs are still largely a black box to the community. Their behaviors are determined by billions of floating-point parameters updated during training on web-scale corpora, and there is no systematic account of how these parameters collectively produce a given output. This lack of understanding gives rise to several undesirable behaviors that are difficult to predict, diagnose, or correct. Our thesis addresses four research problems that demand model interpretability:

Verbatim memorization of training data [16]. LLMs can memorize specific sequences from their pretraining corpora verbatim [17, 18, 19]. Carlini et al. [20] show that an adversary can extract verbatim training examples from GPT-2, including private contact information. Because training data often contains copyrighted text, personally identifiable information, and sensitive content, this memorization behavior poses real privacy and legal risks. Ideally, one could surgically remove a specific piece of memorized information from a deployed model, a goal known as machine unlearning [21, 22]. Achieving this requires first knowing *which* components of the model are responsible for the memorized content. Unfortunately, prior localization methods that identify the weights storing factual knowledge [23, 24] have not been evaluated in a controlled setting where the ground truth components are known, making their reliability for unlearning unclear.

Fragility of in-context learning [25]. ICL empowers LLMs to perform classification and reasoning tasks with only a few labeled demonstrations in the prompt. However, its behavior is sometimes unintuitive and brittle. Min et al. [26] show that swapping in random labels for the demonstrations barely hurts accuracy, suggesting the model may not actually learn from the label semantics. More strikingly, prior work [27, 28] finds that adding a single

space or newline to a prompt can cause large swings in accuracy, even for instruction-tuned or very large models. Most prior work addresses this problem by changing the input prompts [29, 30, 31], while the fundamental internal mechanism driving these sensitivity phenomena is poorly understood.

Input-dependent quantization failures [32]. Quantization reduces model size by representing weights in lower bit-width formats. While 4-bit weight-only quantization generally preserves model quality on standard benchmarks [33], at 3 bits LLMs begin to exhibit noticeable accuracy degradation [34]. Critically, this degradation is not uniform: certain inputs suffer from far larger quantization errors than others, even when state-of-the-art methods that address weight outliers are applied [35, 36]. Which properties of an input make it vulnerable to quantization errors? Which components of the model give rise to the largest errors? The answers to these questions are understudied, limiting our ability to design better quantization methods or to identify inputs for which quantized models should not be trusted.

Long reasoning traces [37]. Reasoning models [38, 39, 40] leverage test-time compute [41] to improve accuracy by generating extended chains of thought before producing a final answer. While this approach yields high accuracy on complex tasks, it introduces an efficiency bottleneck at test time because the reasoning models tend to overthink and generate unnecessarily long outputs. KV cache *eviction* methods [13, 42] solve this problem by permanently discarding low-importance KV pairs once the cache reaches a fixed budget. However, without careful designs, *eviction* methods can yield catastrophic degradation to the model, causing it to generate nonsensical outputs or be stuck in a repetitive loop. We treat key-value pairs as natural components in KV cache eviction, aiming to improve method performance by deeply analyzing the attributes of these components.

These four phenomena share a common root: without understanding the internal computations of an LLM, we cannot explain why the model behaves as it does, predict when it will fail, or intervene to improve it.

1.3 From Interpretability to Action

Mechanistic interpretability [43, 44, 45] aims to reverse-engineer the algorithms implemented by the model, moving beyond correlational observations about inputs and outputs to causal accounts of how specific model components produce specific behaviors.

Component attribution characterizes each model unit (such as an individual neuron, an attention head, or a layer), reflecting its contribution to a target behavior. For example, Olsson et al. [44] identify attention heads that implement a simple copy-and-complete algorithm underlying ICL. FFN layers are characterized as key-value memories that store factual associations [46, 47]. Activation patching and causal tracing [24] are proposed to identify the layers and token positions that are responsible for factual recall.

We believe that a mechanistic understanding of model behavior is most valuable when it leads to actionable consequences. If the neurons responsible for memorizing a specific training sequence can be identified, they can be selectively pruned to make the model forget that sequence without affecting unrelated behavior, enabling targeted machine unlearning (Chapter 3). If the attention heads that are responsible for solving a classification task can be separated from those that introduce noise, they can be up-weighted to improve accuracy (Chapter 4). If the quantized weights that yield large quantization errors can be pinpointed, quantization methods can be redesigned to target those components (Chapter 5). If the value states that drive a model's reasoning progress can be distinguished from those that are safe to discard, we can greatly improve the accuracy of KV cache eviction methods (Chapter 6).

This thesis advances mechanistic interpretability along diverse axes. By developing rigorous benchmarks to evaluate component attribution, by conducting fine-grained causal analyses of four undesirable LLM behaviors, and by translating each analysis into a concrete method or recommendation, this work demonstrates that component attribution is a useful tool for understanding and improving LLMs.

Chapter 2

Background

This chapter provides the conceptual and technical foundations for the studies in this dissertation. We begin with the Transformer architecture that underlies modern LLMs, then survey mechanistic interpretability techniques that can attribute model behaviors to specific Transformer components. We then introduce the three application domains this dissertation addresses, i.e., machine unlearning, in-context learning, and model quantization, and describe how we leverage interpretability techniques to yield actionable insights in each.

2.1 Transformer Architecture

The Transformer [6] is the universal building block of modern LLMs. A Transformer of L layers processes an input sequence of token embeddings through L repeated blocks, each consisting of a multi-head attention (MHA) sublayer and a feed-forward network (FFN) sublayer, connected by residual connections and layer normalization.

2.1.1 Self-Attention

The self-attention mechanism allows each token position to gather information from all other positions in the sequence. Multi-head attention runs n attention heads in parallel,

each attending to different subspaces of the input. The output of MHA at layer l is:

$$A^{(l)} = W_O \cdot \text{Concat}\Big([\text{head}_1^{(l)}, \ldots, \text{head}_n^{(l)}]\Big), \tag{2.1}$$

where $W_O \in \mathbb{R}^{d \times n d_{\text{head}}}$ is the output projection matrix. Each head computes attention-weighted value vectors from the input. A key insight from Elhage et al. [43] is that the output projection W_O can be factored into n per-head matrices $W_{o_i} \in \mathbb{R}^{d \times d_{\text{head}}}$, so the MHA output decomposes into a sum of independent head outputs:

$$A^{(l)} = \sum_{i=1}^{n} W_{o_i}^{(l)} \cdot \text{head}_i^{(l)}. \tag{2.2}$$

Each head thus contributes an additive term directly to the residual stream. This decomposition makes individual attention heads first-class model components and is the foundation of the output decomposition used in Chapter 4.

2.1.2 Feed-Forward Networks

Each Transformer layer also contains a feed-forward network (FFN), sometimes called the MLP sublayer. A two-layer FFN applies a pointwise nonlinearity σ between two linear maps:

$$h^l = \sigma(W^l \mathbf{x}^l), \tag{2.3}$$

$$o^l = V^l h^l, \tag{2.4}$$

where $\mathbf{x}^l \in \mathbb{R}^{d_1}$ is the FFN input, $W^l \in \mathbb{R}^{d_2 \times d_1}$ and $V^l \in \mathbb{R}^{d_1 \times d_2}$ are weight matrices, $h^l \in \mathbb{R}^{d_2}$ is the intermediate hidden state, and $o^l \in \mathbb{R}^{d_1}$ is the output. The individual entries of h^l are called *neurons*. Rewriting Eq. 2.4 as a linear combination of the columns of V^l:

$$o^l = \sum_{i=1}^{d_2} h_i^l \cdot \mathbf{v}_i^l, \tag{2.5}$$

reveals that each neuron activation h_i^l scales the contribution of the corresponding value vector $\mathbf{v}_i^l$ to the output. This formulation connects to the key-value memory view of FFNs discussed in Section 2.3, and motivates searching for which neurons are responsible for specific model behaviors in Chapter 3.

Many recent LLMs, including the LLaMA family [7], use a gated FFN variant (SwiGLU) that multiplies the up projection element-wise with a gate projection before passing the result to the down projection. The gate modulates how much each dimension of the intermediate representation flows through, and its outputs play a critical role in information preservation under quantization, as analyzed in Chapter 5.

2.1.3 Residual Stream

Transformer layers are connected by residual connections. The hidden state evolves as:

$$H^{(l)} = H^{(l-1)} + A^{(l)} + M^{(l)}, \tag{2.6}$$

where $A^{(l)}$ and $M^{(l)}$ are the MHA and FFN outputs at layer l. Unrolling this recurrence, the final hidden state is:

$$H^{(L)} = H^{(0)} + \sum_{l=1}^{L} \left(A^{(l)} + M^{(l)} \right). \tag{2.7}$$

The residual stream therefore accumulates a sum of contributions from all components at all layers, and each component adds its term directly without going through subsequent layers. This additive structure is the basis for the exact logit decomposition in Chapter 4: the output logits can be written as a sum of per-component terms, allowing us to measure the direct contribution of individual components to the outputs.

2.2 Large Language Models

Large language models (LLMs) are trained to model the probability distribution over sequences of tokens drawn from large text corpora. Training proceeds via a next-token prediction objective: given a sequence $[x_1, x_2, \ldots, x_T]$, the model minimizes the negative log-likelihood of the data:

$$-\sum_{t=1}^{T} \log p_\theta(x_t \mid x_1, \ldots, x_{t-1}), \tag{2.8}$$

where θ denotes the model parameters. Applying this loss term at scale (training on hundreds of billions of tokens and with billions of model parameters) yields models with broad linguistic and factual knowledge [5].

Modern LLMs are built on the Transformer architecture [6]. Notable model families include GPT [5], Pythia [48], and LLaMA [7]. These models can be adapted to downstream tasks through natural language instructions or demonstrations at inference time, without any gradient updates to the model parameters.

2.3 Mechanistic Interpretability

Mechanistic interpretability is the study of the internal computations of neural networks, with the goal of attributing observed model behaviors to specific components [43]. For LLMs, the primary components of interest are neurons (individual entries of FFN hidden states), attention heads, and MLP blocks as a whole. All three studies in this dissertation are grounded in mechanistic interpretability: each uses internal model analysis to explain or improve LLM behavior.

2.3.1 FFNs as Parametric Memories

Geva et al. [46] propose viewing the FFN sublayer as a key-value memory store. In this view, the rows of the first weight matrix W^l serve as *keys* that pattern-match against the

input, and the columns of the second weight matrix V^l serve as *values* that encode human-interpretable concepts and world knowledge. When a key matches an input pattern (i.e., the corresponding neuron activation h_i^l is large), the associated value vector $\mathbf{v}_i^l$ is promoted in the output (Eq. 2.5). Geva et al. [47] extend this view by decoding individual value vectors through the output embedding matrix, showing that each vector corresponds to a coherent cluster of related tokens in the vocabulary.

Dai et al. [23] identify *knowledge neurons*, i.e., FFN neurons whose activations are strongly associated with specific factual relations. Suppressing a knowledge neuron significantly reduces the model's probability of expressing the associated fact. Meng et al. [24] localize factual associations to specific MLP layers using causal interventions, and propose ROME, a method that edits the weight matrix V^l at those layers to update stored facts.

2.3.2 Attention Head Roles

Attention heads serve specialized roles beyond generic information aggregation. Olsson et al. [44] identify *induction heads*, i.e., attention heads that copy patterns from earlier in the context by attending to the token that followed a similar token previously, as a key mechanistic substrate for in-context learning. Beyond induction heads, different attention heads have been found to specialize in syntactic relations, coreference, positional patterns, and other structured functions. This functional heterogeneity is precisely what makes the component decomposition in Chapter 4 revealing: rather than treating all heads as equally contributing to the prediction, the decomposition shows that some heads are strong task solvers while others actively hurt performance.

2.4 Localization and Attribution Methods

Attributing model behavior to specific components ("*localization*") requires methods that estimate each component's importance. Several complementary approaches are used through-

out this dissertation:

2.4.1 Activation Analysis

The simplest approach assigns importance to neurons based on their activation magnitudes [47]. High activation values are interpreted as evidence that the corresponding neuron is active for the behavior of interest. Activation-based attribution is computationally cheap but does not directly measure causal importance.

2.4.2 Network Pruning

Traditionally used to reduce model size, network pruning [49] is employed here as an occlusion-based attribution method. We identify key model components by measuring the change in loss following their removal as a proxy for their importance. Specifically, we learn sparse masks over neurons to identify a minimal subset of neurons sufficient for reproducing a target behavior. Chapter 3 adapts two pruning methods, network slimming [50] and hard concrete masking [51], to localize memorized sequences and demonstrates their effectiveness.

2.4.3 Activation Patching

Activation patching [24, 45] identifies causal components by replacing the activations at a specific layer and position in one model run with those from a reference run, and measuring the effect on output. A large effect indicates that the patched component is causally important for the behavior of interest. Chapter 5 adapts cross-model activation patching to the quantization setting, restoring specific activations in a quantized model to their full-precision counterparts to identify which components drive large quantization errors.

2.4.4 Logit Lens (Early Exit)

Logit lens [52], a.k.a early exit and early decoding, projects intermediate hidden states through the final output embedding matrix to obtain token-level interpretations at each layer. This technique, extended by Geva et al. [47], can also be applied to other component activations, providing a vocabulary-space view of what each component is contributing. Chapter 4 uses logit lens as the basis of its component decomposition, which can compute the individual accuracy of each attention head on downstream tasks.

2.5 LLM Memorization

LLMs trained on large corpora tend to memorize portions of their training data. Given a prefix from a training document, an LLM may reproduce the corresponding suffix nearly verbatim using greedy decoding, a phenomenon called *verbatim memorization* [17]. Memorization raises important privacy and security concerns: sensitive personal information, medical records, and copyrighted text present in training data may be extractable through targeted queries [20]. These risks motivate research into *machine unlearning* [21, 22], the task of making a model selectively forget specific training examples after the fact.

One promising approach to machine unlearning is to first identify which model components are responsible for storing a given memorized sequence, and then surgically modify or suppress only those components, leaving the rest of the model intact. Chapter 3 addresses the foundational question underlying this approach: do existing localization methods actually identify the right components, and how can we rigorously evaluate them?

2.6 In-Context Learning

In-context learning (ICL) [5] is a prompting paradigm that enables LLMs to adapt to new tasks using a small number of labeled demonstrations provided in the context, without any

parameter updates. Given a sequence of K input-label demonstration pairs $[x_1, y_1, \ldots, x_K, y_K]$ and a test input x_{test}, the model predicts:

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} P(y \mid x_1, y_1, \ldots, x_K, y_K, x_{\text{test}}), \tag{2.9}$$

where $\mathcal{Y}$ is the set of possible label words. Because ICL requires no gradient updates, it enables rapid deployment across diverse tasks using only a single pretrained model.

Despite its impressive capabilities, ICL is surprisingly sensitive to superficial prompt choices. The order of demonstrations, the choice of label words, and minor formatting differences can cause dramatic swings in accuracy [29, 53]. This sensitivity is puzzling from a task-inference perspective: if the model were simply inferring the task from the demonstrations, these surface-level variations should not matter. The sensitivity instead suggests that ICL performance is governed by specific internal components that respond heterogeneously to different prompt formats, with some robustly extracting the task signal and others introducing noise or bias. Chapter 4 directly investigates this hypothesis by decomposing the ICL output logits into individual attention head and MLP contributions, revealing a rich internal structure that standard LLM evaluation misses.

2.7 LLM Quantization

As LLMs scale to billions of parameters, loading and running them requires substantial memory. Quantization reduces the memory footprint by representing model parameters in lower bit precision. *Weight-only quantization* retains full-precision (16-bit) activations while quantizing model weights to 3–4 bits [10], achieving significant memory savings and maintaining reasonable performance on standard benchmarks. Chapter 5 leverages interpretability tools to provide mechanistic insights on why model quantization disproportionately affects certain examples and how to mitigate the quantization errors.

Chapter 3

Do Localization Methods Actually Localize Memorized Data in LLMs? A Tale of Two Benchmarks

3.1 Introduction

Large language models (LLMs) memorize many sequences from their pretraining corpora [17, 18, 19]. For example, Carlini et al. [20] show that GPT2 [4] can leak some private contact information verbatim. This paper studies whether we can *localize* a piece of memorized data, i.e., identify components in LLMs responsible for generating a sequence (near) verbatim. Successful localization may inform further work in machine unlearning [21, 22]; for instance, one could apply "neural surgery" to the located components to make the LLM forget a piece of sensitive information.

Prior work on knowledge editing suggests that we can locate a small set of LLM parameters that store factual knowledge [23, 24]. These works demonstrate localization success by showing knowledge editing success when updating only the located LLM parameters. However, Hase et al. [54] argue that editing success and localization are actually uncorrelated. Similarly, prior methods that identify subnetworks in LLMs [55, 56] usually focus on the

performance of downstream classification tasks, lacking direct evaluation on localization per se. Hence, the degree of existing methods' localization success remains unclear.

This paper studies the open question, "Do localization methods actually localize memorized data in LLMs?" We first propose decoupling localization success from downstream success in our INJ Benchmark. Our key insight is to actively create the ground-truth weights responsible for data memorization. Specifically, we force LLMs to use a small set of pre-decided weights to memorize a piece of new information unseen during pretraining. Therefore, we have the ground-truth locations where the new information is injected. We can then directly evaluate how well different localization methods recall the indices of the injected weights.

We further apply the localization methods to a real-world scenario: identifying a small set of neurons in an LLM responsible for memorizing a pretrained sequence. In this setting, evaluating localization success is more challenging because the ground-truth "location" of each memorized sequence is unknown. We propose the DEL Benchmark, inspired by knockouts [44], a reverse-engineering approach that removes a set of nodes from the computation graph to observe their importance for specific model behavior. We first collect a set of memorized sequences, and for each sequence, we drop out the located neurons to measure their importance to memorizing that target sequence. A successful localization should cleanly erase the target sequence from an LLM without hurting the memorization of the other sequences in the set after dropout. Our two benchmarks complement each other: the INJ Benchmark provides a direct evaluation of localization methods under a well-controlled setup, while DEL Benchmark answers if the methods can localize pretrained sequences that LLMs have already memorized.

We systematically evaluate five methods on our two benchmarks, including existing localization methods (Activations, [47]; IG, [23]), a brute-force method that searches for the most important neurons (Zero-Out), and two methods we adapt from network pruning [57, 49], Slimming and Hard Concrete. Our two benchmarks rank the five methods in

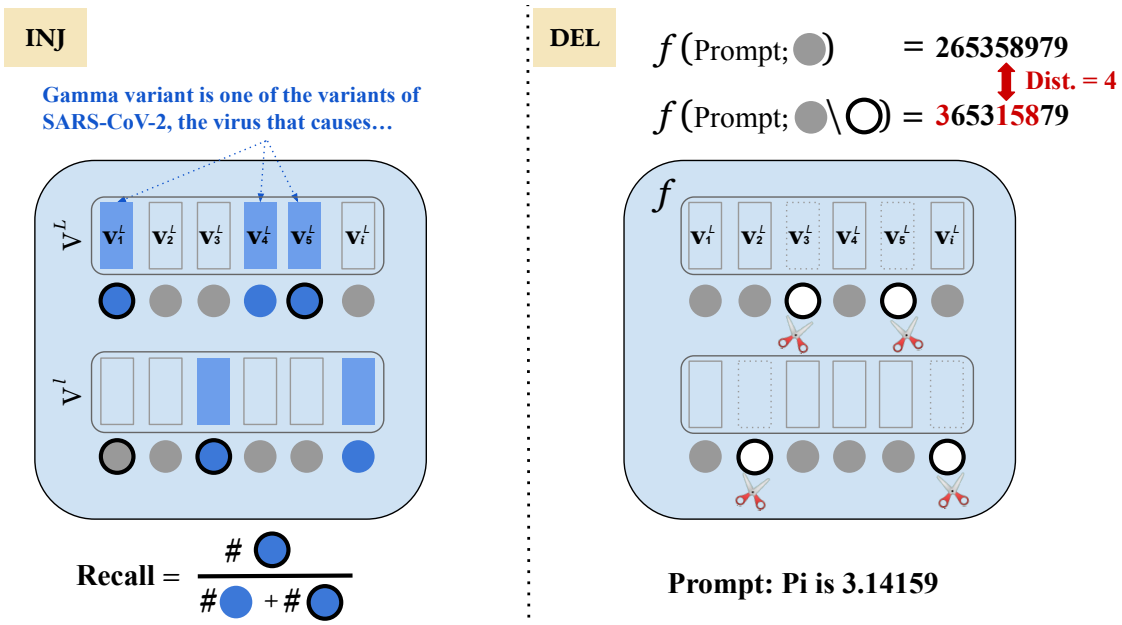


Figure 3.1: **Left:** INJ Benchmark updates a small set of LLM weights to store the new piece of data, where the fine-tuned weight vectors and the corresponding neurons are filled with blue. The neurons predicted by a localization method are circled with black. ● denotes true-positive, ● false-positive, and ● false-negative neurons. **Right:** DEL Benchmark drops out the predicted neurons ○ on a memorized pretrained sequence. A large change in Levenshtein distance after dropout indicates that ○ were important for LLM f to retrieve the memorized sequence.

the same order, showing especially strong localization ability for HARD CONCRETE. For example, dropping out only 0.5% of neurons in Pythia-6.9B [48] identified by HARD CONCRETE makes the model forget 57.7% of the target memorized tokens on average. On the other hand, the DEL Benchmark shows all methods struggle to balance between erasing the target sequence and retaining other memorized data, indicating that the identified neurons are also relevant for memorizing some other sequences. Overall, both benchmarks agree all evaluated localization methods are promising, but precise localization of a single sequence remains difficult.

3.2 Terminology

A Transformer layer [6] consists of multi-head self-attention and a feed-forward network (FFN). Prior work shows that LLMs use their FFNs rather than self-attention as "memories" to store knowledge [46, 47, 24]. Here, an FFN has two fully connected layers with a non-linear activation function σ:

$$h^l = \sigma(W^l \mathbf{x}^l) \tag{3.1}$$

$$o^l = V^l h^l, \tag{3.2}$$

where $\mathbf{x}^l \in \mathbb{R}^{d_1}$ is the input hidden states to the l-th FFN layer, $W^l \in \mathbb{R}^{d_2 \times d_1}, V^l \in \mathbb{R}^{d_1 \times d_2}$ are the weights, $h^l \in \mathbb{R}^{d_2}$ the intermediate hidden states, and $o^l \in \mathbb{R}^{d_1}$ the output hidden states. Geva et al. [47] rewrite Eq. 3.2 as a linear combination of columns of V^l. Let $\mathbf{v}_i^l \in \mathbb{R}^{d_1}$ be the i-th column of V^l and $h_i^l \in \mathbb{R}$ be the i-th neuron activation of $h^l \in \mathbb{R}^{d_2}$. We have:

$$o^l = V^l h^l = \sum_{i=1}^{d_2} h_i^l \cdot \mathbf{v}_i^l \tag{3.3}$$

They show that different concepts are stored in different $\mathbf{v}_i^l$, and that we can view each activation h_i^l as a memory coefficient to retrieve a concept.

Neurons. Dai et al. [23] observe the existence of knowledge neurons, a small set of neurons in FFN hidden states h^l that corresponds to a relational fact, where a neuron means a component of the vector h^l. For example, given the input "*The capital of Ireland is* ___", they can increase the model probability on the correct token "*Dublin*" by amplifying the activation h_i^l of the identified knowledge neurons. With Eq. 3.3, we can view increasing activation h_i^l as promoting the concept stored in $\mathbf{v}_i^l$.

In this work, we only search for neurons in FFNs responsible for memorizing a sequence, following Dai et al. [23]. In the INJ Benchmark, we ensure that FFNs act as neural memories

by only updating a set of weight vectors $\mathbf{v}_i^l$ to memorize the new information. As each $\mathbf{v}_i^l$ corresponds to a neuron in h^l, locating the updated weights is equivalent to locating the corresponding neurons. In the rest of the paper, we refer to neurons as the neurons in $\{h^l\}_{l=1}^L$, where L is the number of layers.

Dropout. Different from Srivastava et al. [58], we drop out located neurons at test time to erase a memorized sequence from the LLM. We can view dropping out the i-th neuron in h^l as excluding the contribution of $\mathbf{v}_i^l$ from the output o^l in Eq. 3.3.

Memorized Sequences. Consider a sequence $x = (p, s)$ that consists of a prefix p and a suffix s. Given the prefix as the prompt, if an LLM is able to *nearly reconstruct* the suffix with greedy decoding, we say x is a memorized sequence by the LLM. We discuss in 3.3.2 our criteria on suffix reconstruction, where we tolerate near-verbatim memorization; we also ensure every sequence has a non-trivial suffix.

Localization. Hase et al. [54] provides a general definition of localization: identifying components of a model responsible for a certain behavior. Under this definition, we consider components as a small set of neurons and behavior as the LLM's generation of a memorized sequence. Although some components are necessary for generation, e.g., the input and output token embeddings, we do not consider them as they are not specific to any target sequence.

Localization Methods. Given an LLM, a memorized sequence x, and a fixed number k, a localization method outputs $k\%$ of neurons at each layer as the predictions to localize sequence x in the LLM.

3.3 Two Localization Benchmarks

How do we know whether a method is successful in localization? We propose two benchmarking approaches: one *injects* a new piece of information into specific parameters in LLMs, while another *deletes* an existing memorized sequence from LLMs via dropout. A successful localization method should do well on both benchmarks.

3.3.1 The INJ Benchmark

A principal challenge in evaluating localization methods is the lack of ground-truth location. We propose the INJ Benchmark, which first creates ground truth by actively injecting a piece of unseen information into a small subset of LLM weights. We can then directly evaluate the correctness of a localization method in predicting the indices of those injected weights.

Data. The ECBD-2021 dataset [59] contains 156 definition sentences of new entities that rose to popularity during the year 2021, e.g., "*Gamma variant, also known as lineage P.1...*". Since all LLMs we use are trained on corpora released before 2021, the injected weights are the only parameters in the LLMs responsible for memorizing each new definition sequence x.

Information Injection. For each new sequence x_i in the dataset, we randomly sample $r\%$ of the weight vectors $\{\mathbf{v}_1^l, \ldots, \mathbf{v}_{d_2}^l\}_{l=1}^L$ across all L layers, and fine-tune them to memorize x. We keep the rest of the model parameters frozen. To simulate how LLMs learn data during pretraining, we fine-tune with the normal language modeling loss on x_i (Eq. 1). To ensure the entire sequence is well memorized, we keep fine-tuning until we reach a loss < 0.05; therefore, we can simply set the first token as the prefix p, and the rest of the sequence as the suffix s. Note we sample a different set of weight vectors ϕ_i for each sequence x_i and fine-tune a separate model $\tilde{\theta}_i$. Algo. 1 shows the exact injection process.

Evaluation. For each model $\tilde{\theta}_i$ injected with a sequence x_i, a localization method predicts $k\%$ of neurons at each layer and we calculate Recall@$k\%$. Specifically, given the set of ground-truth neurons corresponding to all the injected weight vectors across layers, Γ_i, and the set of all predicted neurons, $\hat{\Gamma}_i$, the recall is $\frac{|\Gamma_i \cap \hat{\Gamma}_i|}{|\Gamma_i|}$. Finally, we average the recall scores across sequences, and thus average over different choices of weights ϕ_i for injection.

Algorithm 1 Information Injection

Input: The set of new sequences $\mathcal{X}_{\text{ECBD}} = \{x_i\}_{i=1}^{N}$; pretrained LLM θ with L layers; injection ratio r
Output: The set of fine-tuned LLMs $\mathcal{M} = \{\tilde{\theta}_i\}_{i=1}^{N}$
Initialize $\mathcal{M} \leftarrow \varnothing$.
for $i \leftarrow 1$ to N **do**
  $\tilde{\theta}_i \leftarrow \theta$ // Initialize from pretrained weights.
  Retrieve all the FFN weight vectors $\Phi_i = \{\mathbf{v}_1^l, \ldots, \mathbf{v}_{d_2}^l\}_{l=1}^{L}$ from layers l of $\tilde{\theta}_i$.
  Set the random seed to i.
  $\phi_i \leftarrow$ Randomly sample $r\%$ of weight vectors from Φ_i. // $\phi_i \subset \Phi_i \subset \tilde{\theta}_i$
  Fine-tune ϕ_i with the language modeling loss on x_i (Eq. 1) with remaining weights $\tilde{\theta}_i \setminus \phi_i$ frozen.
  $\mathcal{M} \leftarrow \mathcal{M} \cup \tilde{\theta}_i$.
end for
return $\mathcal{M}$

3.3.2 The DEL Benchmark

The DEL Benchmark studies whether we can localize a naturally memorized sequence after pretraining, which is not answered by the INJ Benchmark. We first collect a set of memorized pretrained sequences, and then apply localization methods to identify the responsible neurons for each sequence. Without ground-truth neurons, we adopt knockouts [60, 44, 61] for evaluation, which measures the importance of model components based on the effect of removing them. We drop out the located neurons to observe how much they account for memorizing a sequence. We quantify memorization with two scores: Accuracy and Levenshtein distance.

Accuracy. Recall that a sequence $x = (p, s)$ consists of a prefix p and suffix s. Accuracy calculates the percentage of correct suffix tokens generated by teacher-forcing and argmax

decoding. Formally,

$$\hat{s_t} = \underset{w \in \text{Voc}}{\text{argmax}}\ P_\theta(w|p, s_{<t}),\ t = 1, \ldots, T \tag{3.4}$$

$$\text{Accuracy} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{1}\{\hat{\mathbf{s_t}} = \mathbf{s_t}\}, \tag{3.5}$$

where T denotes the suffix sequence length, s_t the t-th true suffix token, $s_{<t} = [s_1, \ldots, s_{t-1}]$, $\hat{s_t}$ the t-th generated token, P_θ the probability distribution of the LLM parameterized by θ, and Voc the vocabulary. Higher Accuracy indicates better memorization of the sequence.

Levenshtein distance. While Accuracy is defined at a token level, we note tokens often contain several characters, e.g., "159". For sequences like "3.14159265", every character is important; thus, we also define a memorization score at the character level. With Eq. 3.4, we have $\hat{s} = [\hat{s_1}, \ldots, \hat{s_T}]$. We calculate Levenshtein distance between the generated suffix $\hat{s}$ and the true suffix s. Lower Levenshtein distance indicates better memorization.

Data. We collect a set of sequences memorized by each LLM, including Pythia-deduped-2.8B, Pythia-deduped-6.9B, and GPT2-XL. For Pythia models, the pretraining corpus the Pile-dedupe [62] is open-sourced, and we use the following criteria to determine which sequences are memorized. For each candidate sequence x, we set the first 32 tokens as the prefix p to prompt the LLM to reconstruct the suffix s of 48 tokens. First, we filter out sequences with Accuracy (Eq. 3.4, 3.5) lower than 0.9. Second, we use greedy decoding to generate the suffix, filtering out those with a Levenshtein distance greater than 20 characters to the true suffix. Third, we discard sequences with repetitive tokens (less than 16 distinct tokens in the suffix). Finally, we deduplicate the remaining sequences based on n-gram Jaccard index. We obtain 505 memorized sequences for each Pythia model. For GPT2-XL, we do not have access to its pretraining corpus and find very few memorized sequences from several public corpora with our criteria. Thus, we actively search for potentially memorized sequences, discovering 105 memorized sequences and categorizing them manually (Table 3.1). See A.8

for details and example sequences.

We sample 5 sequences as the dev set to tune the hyperparameters of different methods (see A.9), using the rest of the collected sequences as the test set. We quantify the memorization of LLMs on the collected test sets. Table A.4 in the appendix shows that all LLMs have a high average Accuracy ($\sim 100\%$) and a low Levenshtein distance (~ 1 character) to the true suffix, suggesting that the sequences we collect are indeed well memorized.

Evaluation. When we evaluate one sequence x in the collected test set $\mathcal{X}$, we consider the rest of the memorized sequences, $\mathcal{X} \setminus \{x\}$, as negative examples. A successful localization method should make LLMs forget the target sequence (large changes in memorization scores), but still remember the other negative examples (small changes in memorization scores) after dropping out the predicted $k\%$ of neurons at each layer.[1] We also calculate the absolute change in perplexity on a batch of 2048 sequences sampled from the Pile-dedupe, $\mathcal{D}$, to evaluate whether the general language modeling ability remains intact after dropout.

[1]We do not drop out neurons in the bottommost layer, as it hurts LLMs' overall memorization indiscriminately (3.5.4).

Category	**Examples**	**Count**
Quotes	Churchill, Steve Jobs, Trump	17
Quotes (Book)	Dune, 1984, Bible	14
Ordered items	Zodiac Signs, US Presidents	11
Terms of use	MIT License	10
Poems	The Second Coming	9
Code	GitHub	9
Contact Info	A journalist's email	7
URLs	Reddit, file link	5
Others	long COINBASE ID, meme, Bill of Rights, Pi digits	23

Table 3.1: Collected sequences memorized by GPT2-XL.

Despite similarities to the evaluation of model editing [63, 64], we can better reflect localization success. Unlike Meng et al. [24] that edit the located weights with gradients, we restrict our operation to neuron dropout. Because dropout has limited freedom in changing LLMs behavior, successful deletion via dropout requires successful localization; in contrast, gradient-based editing could succeed even without good localization [54].

3.4 Localization Methods

We benchmark five localization methods. Each method assigns an attribution score $\mathcal{A}^l(i)$ to each neuron n_i^l, the i-th neuron in the l-th layer, representing its importance in memorizing a sequence x. At test time, we select the top-k% of neurons in each layer for each method in terms of attribution scores as the located neurons for x by that method.

Several methods involve calculating the language modeling loss of an LLM θ on the suffix of the target sequence $x = (p, s)$. We denote the loss as *memorization loss*, $\ell_\theta^{\text{mem}}(x)$. Formally,

$$\ell_\theta^{\text{mem}}(x) = \frac{1}{T}\sum_{t=1}^{T} -\log P_\theta(s_t|p, s_{<t}) \tag{3.6}$$

Zero-Out. We introduce an exhaustive method that drops out neurons one by one and uses the resulting change in memorization loss on x as the attribution score to each neuron n_i^l:

$$\mathcal{A}^l(i) = \ell_{\theta\setminus n_i^l}^{\text{mem}}(x) - \ell_\theta^{\text{mem}}(x) \tag{3.7}$$

We denote $\ell_{\theta\setminus n_i^l}^{\text{mem}}$ as the memorization loss of the LLM θ after dropping out a neuron n_i^l. The larger the change in the loss, the more important the neuron is for memorization. ZERO-OUT is closely related to the occlusion-based attribution method [65].

Activations. We can view the neuron activation h_i^l as the memory coefficients (3.2). Thus, similar to Geva et al. [47], we set the attribution $\mathcal{A}^l(i)$ as the absolute value of h_i^l multiplied

by the vector norm of $\mathbf{v}_i^l$, averaged across the suffix length T:

$$\mathcal{A}^l(i) = \frac{1}{T}\sum_{t=1}^{T} |h_{i,t}^l| \, \|\mathbf{v}_i^l\|, \tag{3.8}$$

where $h_{i,t}^l$ denotes the activation value at the t-th timestep, when the input consists of all the tokens before s_t, i.e., $[p, s_{<t}]$.

Integrated Gradients (IG). We benchmark integrated gradients [66], an attribution method that has been used to identify knowledge neurons and privacy neurons [23, 67]. IG cumulates the gradients at all points along the path from a zero vector to the original hidden state h^l. See A.2 for more details.

	GPT2 124M			**GPT2-XL 1.5B**			**Pythia-deduped 2.8B**			**Pythia-deduped 6.9B**		
	R@1%	R@2%	R@5%	R@1%	R@2%	R@5%	R@1%	R@2%	R@5%	R@1%	R@2%	R@5%
ratio = 1%												
HARD CONCRETE	**49.5**$_{0.48}$	**70.2**$_{0.54}$	**87.4**$_{0.33}$	**29.7**$_{0.47}$	**37.1**$_{0.49}$	**48.1**$_{0.46}$	34.3$_{0.33}$	50.1$_{0.43}$	**72.1**$_{0.47}$	36.8$_{0.45}$	**55.1**$_{0.51}$	**76.4**$_{0.41}$
SLIMMING	48.1$_{0.64}$	66.7$_{0.69}$	80.7$_{0.54}$	19.3$_{0.49}$	29.2$_{0.59}$	41.1$_{0.59}$	**37.0**$_{0.43}$	**50.7**$_{0.47}$	61.5$_{0.44}$	**39.9**$_{0.38}$	**55.1**$_{0.38}$	66.5$_{0.35}$
ZERO-OUT	24.9$_{0.78}$	37.5$_{1.05}$	53.8$_{1.24}$	4.1$_{0.13}$	7.2$_{0.23}$	13.7$_{0.42}$	10.6$_{0.20}$	15.0$_{0.24}$	21.4$_{0.30}$	-	-	-
IG	20.5$_{0.55}$	32.1$_{0.80}$	49.9$_{0.99}$	4.3$_{0.13}$	7.2$_{0.21}$	13.3$_{0.37}$	11.6$_{0.22}$	16.9$_{0.28}$	23.9$_{0.34}$	12.8$_{0.23}$	18.7$_{0.29}$	27.2$_{0.35}$
ACTIVATIONS	3.0$_{0.09}$	5.2$_{0.13}$	13.3$_{0.32}$	2.1$_{0.05}$	5.0$_{0.10}$	12.0$_{0.16}$	7.8$_{0.11}$	12.8$_{0.20}$	30.5$_{0.53}$	7.9$_{0.11}$	12.4$_{0.17}$	27.3$_{0.43}$
RANDOM	1.0$_{0.04}$	2.1$_{0.06}$	5.0$_{0.09}$	1.0$_{0.01}$	2.0$_{0.02}$	5.0$_{0.03}$	1.0$_{0.01}$	2.0$_{0.02}$	5.0$_{0.03}$	1.0$_{0.01}$	2.0$_{0.02}$	5.0$_{0.02}$
ratio = 0.1%	@0.1%	@0.2%	@0.5%	@0.1%	@0.2%	@0.5%	@0.1%	@0.2%	@0.5%	@0.1%	@0.2%	@0.5%
HARD CONCRETE	56.4$_{0.83}$	79.6$_{0.89}$	93.7$_{0.52}$	**47.5**$_{0.40}$	**59.1**$_{0.47}$	68.0$_{0.46}$	**48.5**$_{0.49}$	**67.3**$_{0.50}$	**86.7**$_{0.34}$	46.4$_{0.60}$	**66.3**$_{0.71}$	**82.3**$_{0.48}$
SLIMMING	**58.9**$_{0.59}$	**83.5**$_{0.68}$	**94.4**$_{0.49}$	35.4$_{0.56}$	55.9$_{0.64}$	**69.5**$_{0.55}$	48.3$_{0.43}$	63.5$_{0.46}$	73.9$_{0.43}$	**48.5**$_{0.57}$	60.9$_{0.60}$	71.0$_{0.71}$
ZERO-OUT	54.1$_{0.68}$	77.8$_{0.78}$	90.9$_{0.70}$	14.3$_{0.62}$	21.8$_{0.94}$	31.9$_{1.27}$	16.5$_{0.48}$	21.1$_{0.57}$	26.6$_{0.66}$	-	-	-
IG	53.5$_{0.78}$	74.1$_{0.92}$	84.8$_{0.80}$	13.8$_{0.53}$	20.3$_{0.79}$	29.7$_{1.06}$	18.0$_{0.49}$	23.3$_{0.60}$	30.2$_{0.68}$	29.3$_{1.03}$	34.4$_{1.02}$	39.6$_{0.97}$
ACTIVATIONS	11.1$_{0.43}$	26.5$_{0.84}$	51.5$_{1.06}$	7.5$_{0.35}$	15.9$_{0.61}$	30.6$_{0.76}$	21.6$_{0.72}$	34.6$_{0.98}$	52.5$_{1.07}$	34.0$_{1.03}$	45.9$_{1.02}$	59.5$_{0.97}$
RANDOM	0.1$_{0.03}$	0.2$_{0.06}$	0.5$_{0.07}$	0.1$_{0.01}$	0.2$_{0.02}$	0.5$_{0.03}$	0.1$_{0.01}$	0.2$_{0.02}$	0.5$_{0.03}$	0.1$_{0.01}$	0.2$_{0.02}$	0.5$_{0.02}$

Table 3.2: The INJ Benchmark. We experiment with injection ratio at 1% (**Top**) and 0.1% (**Bottom**) and report the Recall@k% and standard errors of different localization methods averaged across the sequences in ECBD-2021.

Slimming. We introduce SLIMMING, a localization method adapted from prior work [50, 68] on network pruning. Pruning aims to reduce the model size by finding a subnetwork that can achieve a low loss on the task, e.g., sentiment analysis. In our setting, we find a small set of neurons that are crucial for maintaining a low memorization loss $\ell_\theta^{\text{mem}}(x)$ on *one* target sequence x (Eq. 3.6). Specifically, SLIMMING minimizes the memorization loss while

learning a sparse mask $m^l \in \mathbb{R}^{d_2}$ on the hidden state h^l in every layer, with mask value m_i^l on neuron n_i^l. At each layer, we transform h^l to $h^l \odot m^l$ before computing further layers, where $\odot$ denotes element-wise multiplication. The sparse mask encourages the LLM to use only a small set of neurons to recall a piece of memory. All the weights of the LLM are kept frozen during the training; only the mask m^l is learnable. Formally,

$$\min_{\substack{m^l \\ l=1,\ldots,L}} \ell_\theta^{\text{mem}}(x) + \lambda \sum_{l=1}^{L} \|m^l\|_1, \tag{3.9}$$

where λ is the hyperparameter to balance the memorization loss and the L_1 sparsity regularization on the mask. After training, we set the attribution score $\mathcal{A}^l(i) = m_i^l$, as m_i^l learns the importance of the existence of a neuron to the memorization loss.

Hard Concrete. The limitation of SLIMMING is that it tends to assign mask values m_i^l between 0 and 1 on most neurons, creating a mismatch between training and testing. In particular, at inference time we either activate (equivalent to setting $m_i^l = 1$) or drop out ($m_i^l = 0$) a neuron. Thus, we adapt another pruning method HARD CONCRETE [51, 69] for localization, which improves over SLIMMING by encouraging mask values m_i^l to be approximately binary. Similar to SLIMMING, HARD CONCRETE learns parameters $m^l \in \mathbb{R}^{d_2}$ in every layer. But instead of directly using m^l as the mask, the mask $\bar{m}^l$ in HARD CONCRETE is a random variable (r.v.) that depends on m^l. Specifically, HARD CONCRETE derives the mask value $\bar{m}_i^l$ from a binary concrete [70, 71] random variable, $\hat{m}_i^l$. A binary concrete distribution $\hat{m}_i^l \sim \text{Concrete}(m_i^l, \beta)$ is parameterized by the location m_i^l and temperature β. When the hyperparameter $\beta \to 0$, sampling from the binary concrete distribution is identical to sampling from a Bernoulli distribution but loses the differentiable property. With $\beta > 0$, we allow gradient-based optimization of parameter m_i^l through the reparametrization trick.

Formally,

$$u_i \sim \mathcal{U}(0,1), \tag{3.10}$$

$$\hat{m}_i^l = \sigma\left(\frac{1}{\beta}(\log\frac{u_i}{1-u_i} + \log m_i^l)\right), \tag{3.11}$$

where σ denotes the sigmoid function and u_i is a r.v. sampled from uniform distribution $\mathcal{U}(0,1)$. We describe the details about how Louizos, Welling, and Kingma [51] extend a hard concrete r.v. $\bar{m}^l$ from the binary concrete r.v. $\hat{m}_i^l$ and use L_0 regularization $\mathcal{R}(\bar{m}^l)$ to encourage sparsity in A.4.

To learn the parameters m^l, we freeze the LLM weights θ and simultaneously optimize the memorization loss on the target sequence x and the sparsity loss $\mathcal{R}(\bar{m}^l)$. Formally,

$$\min_{\substack{m^l \\ l=1,\ldots,L}} \ell_\theta^{\text{mem}}(x) + \lambda \sum_{l=1}^{L} \mathcal{R}(\bar{m}^l) \tag{3.12}$$

At test time, $\hat{m}_i^l$ can be estimated as $\sigma\left(\log m_i^l\right)$ [51]; thus, we set the attribution score $\mathcal{A}^l(i) = \sigma\left(\log m_i^l\right)$.

3.5 Experiments

3.5.1 INJ Benchmark Results

Table 3.2 shows the average Recall@k% and standard errors of different localization methods on four LLMs under our INJ Benchmark evaluation. When the injection ratio is 1% (Table 3.2; Top), there are 1% of weight vectors injected with each new sequence, yielding 1% of ground truth neurons, and every method predicts $k = \{1, 2, 5\}$% of neurons at each layer. When the injection ratio is 0.1% (Table 3.2; Bottom), every method predicts $\{0.1, 0.2, 0.5\}$% of neurons at each layer. We also study the alternative that predicts top-k neurons *across* layers in A.11, which shows results consistent with Table 3.2 but with lower recall overall.

	Δ Self-Acc ↓		Δ Self-Dist ↑		Δ Neg-Acc ↑		Δ Neg-Dist ↓		Δ Rand-PPL ↓	
dropout ratio =	0.1%	0.5%	0.1%	0.5%	0.1%	0.5%	0.1%	0.5%	0.1%	0.5%
GPT2-XL 1.5B										
HARD CONCRETE	**-34.6%**	-57.1%	**42.9**	74.0	-2.4%	-4.8%	2.5	5.4	0.03	0.11
SLIMMING	-30.5%	**-57.8%**	37.7	**75.4**	-3.5%	-6.4%	4.1	7.5	0.02	0.17
ZERO-OUT	-29.8%	-46.1%	33.0	55.2	-3.1%	-4.8%	3.5	5.5	0.03	0.09
IG	-25.8%	-40.8%	27.0	46.0	**-2.2%**	**-3.4%**	**2.3**	**3.7**	**0.01**	**0.05**
ACTIVATIONS	-14.8%	-29.5%	16.9	36.4	-3.0%	-4.7%	3.1	5.4	0.11	0.16
RANDOM	-0.2%	-0.5%	0.2	0.4	-0.2%	-0.5%	0.1	0.4	0.00	0.03
Pythia-deduped 2.8B										
HARD CONCRETE	**-29.0%**	**-53.2%**	**55.3**	**99.8**	-3.7%	-10.5%	7.7	22.1	0.23	0.56
SLIMMING	-17.4%	-45.1%	32.9	80.8	-3.3%	-7.0%	6.6	13.9	0.26	0.49
ZERO-OUT	-14.8%	-25.9%	26.4	45.2	-1.1%	-2.5%	2.1	5.0	0.21	0.35
IG	-16.7%	-30.3%	29.1	52.5	**-0.9%**	**-2.1%**	**1.8**	**4.4**	**0.09**	**0.18**
ACTIVATIONS	-13.0%	-25.5%	27.5	52.2	-3.1%	-6.1%	6.6	12.9	0.11	0.20
RANDOM	-0.1%	-0.3%	0.1	0.5	-0.1%	-0.3%	0.2	0.5	0.00	0.02
Pythia-deduped 6.9B										
HARD CONCRETE	**-29.2%**	**-57.7%**	**58.5**	**109.9**	-3.8%	-14.7%	8.7	32.6	0.16	0.52
SLIMMING	-24.1%	-48.7%	48.8	92.1	-4.2%	-11.3%	9.1	23.6	0.23	0.58
IG	-16.9%	-32.3%	31.4	57.8	**-2.3%**	**-4.9%**	**5.3**	**11.5**	0.27	**0.37**
ACTIVATIONS	-11.5%	-26.8%	25.5	51.5	-2.5%	-8.1%	5.5	17.2	**0.12**	0.45
RANDOM	-0.1%	-0.2%	0.1	0.4	-0.1%	-0.2%	0.1	0.3	0.00	0.02

Table 3.3: The DEL Benchmark. HARD CONCRETE is the most effective method in erasing the target sequence (Self), while IG can best maintain the LLM performance on unrelated sequences (Neg and Rand) after dropout.

All methods can do localization. First, all five localization methods greatly outperform RANDOM, which randomly predicts the same number of neurons at each layer. Interestingly, when the injection ratio is lower (0.1%), all localization methods achieve higher recall, possibly because the information is more concentrated in the injected weights and thus easier to identify.

Pruning-based methods perform the best. SLIMMING and HARD CONCRETE, the methods based on network pruning, substantially outperform the other methods across all setups. Specifically, HARD CONCRETE achieves Recall@0.5% higher than 80 in three out of four LLMs. ZERO-OUT and IG perform similarly and outperform the simple method ACTIVATIONS overall, but are much more computationally expensive than the other methods (see comparisons in A.7).

Our results hold under more data and different random seeds. In the appendix, we show that our conclusions hold when expanding the INJ Benchmark to the newly released ECBD 2022, 2023 dataset [72] (A.5), and they are robust to the choice of random seed, which controls the choice of injected weights (A.6).

3.5.2 DEL Benchmark Results

Table 3.3 shows to what extent dropping out $k = \{0.1, 0.5\}\%$ of neurons predicted by different methods makes LLMs forget the target sequence x (Self), while still memorizing the other sequences $\mathcal{X} \setminus \{x\}$ (Neg), and keeping the perplexity on the random batch $\mathcal{D}$ (Rand-PPL) intact. We evaluate one target sequence at a time and report the average absolute changes (Δ) in Accuracy (Acc), Levenshtein distance (Dist), and perplexity after dropout.

All methods show evidence of localization. Randomly dropping out the same number of neurons (RANDOM) barely changes the LLM behavior. In comparison, all five localization methods successfully identify neurons that contribute much more to memorizing the target sequence than to negative examples, showing evidence of their localization ability on real-world memorized data.

Methods trade off between Δ Self and Δ Neg. We find SLIMMING and HARD CONCRETE much more effective than other methods in erasing the target sequence itself. However, they are worse at preserving LLM memorization of the negative examples and the perplexity of randomly sampled sequences. For example, dropping out 0.5% of GPT2 neurons predicted by SLIMMING decreases Accuracy by 57.8% and increases 75.4 characters in Levenshtein distance on the target sequence, but it also hurts the Accuracy on negative examples by 6.4% and increases Levenshtein distance by 7.5 on average. On the other hand, IG best maintains the performance on negative examples and perplexity, but is not as successful in erasing the target sequence itself. Interestingly, although ZERO-OUT assigns the attribution scores with a leave-one-out approach, it does not perform the best on either target

sequences or negative examples, implying that the individual neuron dropout effect does not reliably predict the collective effect of dropping out many neurons at the same time. Overall, it is challenging for methods to effectively and specifically locate the target sequence at the same time.

Which negative examples are forgotten? We analyze how the negative examples affected by dropout are related to the target sequence. Figure 3.2 is the confusion matrix on a representative subset of GPT2 memorized data, $\mathcal{Y} \subset \mathcal{X}$, where each row shows how dropping out 0.5% of the neurons predicted by Hard Concrete on a target sequence changes the Accuracy of every sequence in $\mathcal{Y}$. We group sequences under the same category (see Table 3.1) in adjacent rows. We find Hard Concrete sometimes confuses related data; for example, in the Address category consisting of mailing addresses, dropping out the neurons of an address sequence also causes substantial Accuracy drops on other addresses. We also find confusion across the Poems, Shakespeare, and Bible categories of literary sequences. Qualitatively, we found several web pages containing famous quotes from different poems and books; such co-occurrences may also appear multiple times in GPT2's pretraining corpus and may explain why in Figure 3.2, a small set of neurons affect quotes from different sources. While these findings could suggest that Hard Concrete struggles to pinpoint neurons that are specific to a target sequence, it may also be that LLMs actually use a shared set of neurons to memorize several related sequences. Figure A.2 in A.8 shows the confusion matrices of other methods and Figure A.3 is the matrix of the entire dataset $\mathcal{X}$. Both figures share patterns similar to Figure 3.2.

3.5.3 Concurrence of the two benchmarks

This section studies if the two benchmarks rank the methods similarly [73] and whether the differences between methods are significant.

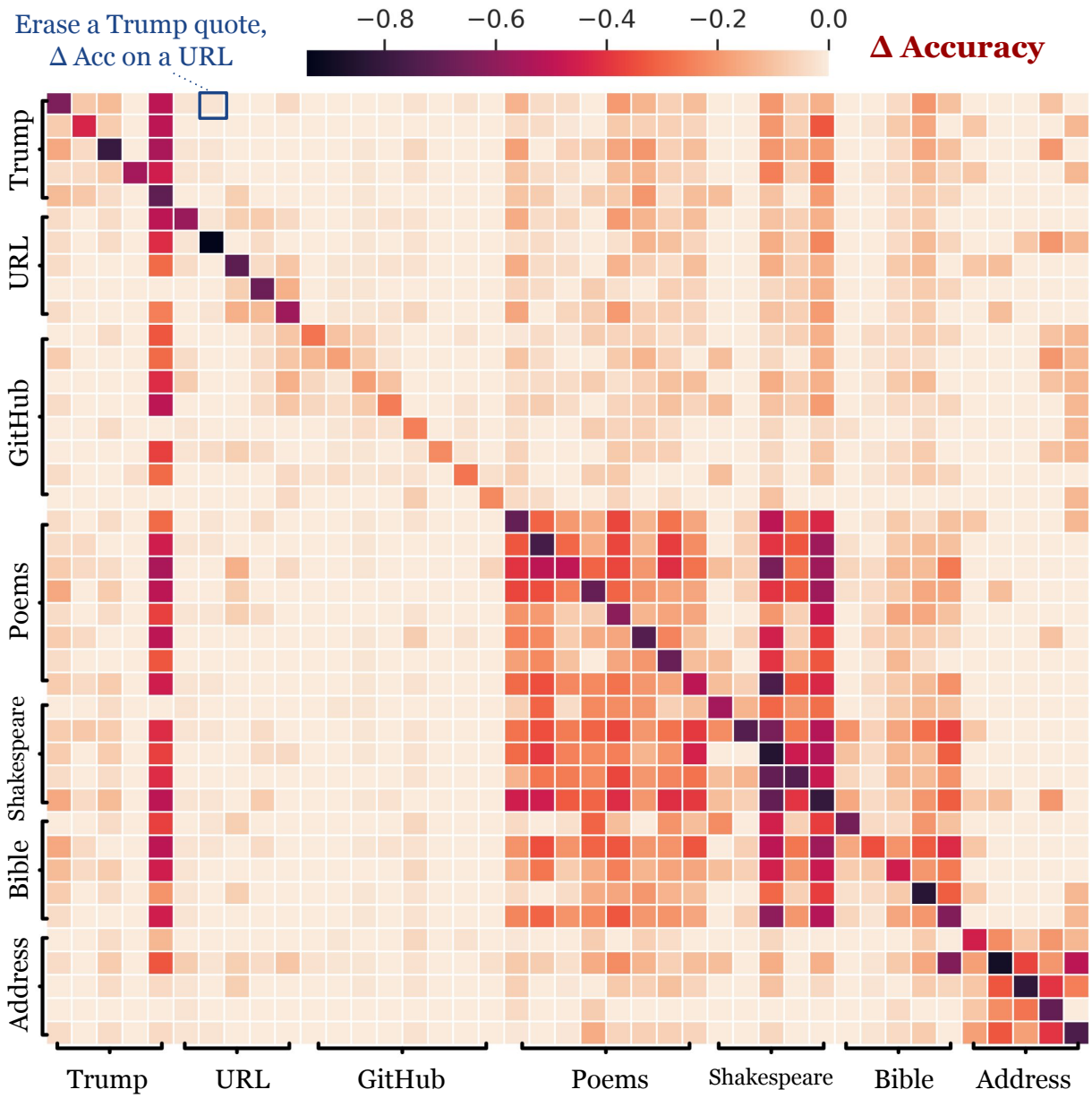


Figure 3.2: The confusion matrix of HARD CONCRETE on a subset of data memorized by GPT2-XL.

Rankings of localization methods. The INJ Benchmark, which solely evaluates the injected target sequences,[2] and the Self- part of the DEL Benchmark show consistent rankings: HARD CONCRETE performs slightly better than SLIMMING, followed by ZERO-OUT and IG; ACTIVATIONS performs the worst but still substantially outperforms RANDOM. This

[2]INJ Benchmark does not have negative examples, since we do not have ground-truth neurons of pretrained sequences.

consistency suggests that despite the differences in data and setups, the two benchmarks reflect the same underlying localization abilities of different methods. We believe the reason pruning methods perform better is that they learn to mask multiple neurons simultaneously, therefore consider the collective effect of different components, while other methods only consider the importance of each neuron individually.

Tests of significance. We run t-tests to test if pruning-based methods outperform IG significantly. For the INJ Benchmark, each method has 24 Recall@k% scores in Table 3.2; we run 24 one-tailed paired t-tests accordingly. With Bonferroni correction, we set the significance level $\alpha = \frac{0.05}{24}$. Table A.7 in the appendix shows that for HARD CONCRETE vs. IG and SLIMMING vs. IG, respectively, there are 23/24 and 24/24 tests that have p-values $< \alpha$. Similarly, in the DEL Benchmark, each method has 6 Δ Self-Acc scores in Table 3.3; thus, we run 6 paired t-tests. Table A.8 shows that 5/6 and 6/6 tests have p-values $< \frac{0.05}{6}$, for SLIMMING vs. IG and HARD CONCRETE vs. IG, respectively. Notably, for both benchmarks, most tests have p-values $< 10^{-10}$. Overall, these results support our claims that the difference between pruning-based methods and IG is significant.

3.5.4 Is the memory of a piece of information distributed over layers?

To understand the individual effect of each layer on memorization, we study the alternative that drops out the same number of neurons in a *single* layer. In 3.5.2, a method predicts top-0.1% of neurons in *every* layer after the bottommost layer; thus, we have a "budget" of $N = 0.1\% \times 6400 \times (48 - 1)$ neurons for GPT2-XL. Here, the alternative strategy drops out the top-N neurons in a single layer in terms of the attribution scores assigned by a method.

Using the attribution assigned by IG, Figure 3.3 compares the two dropout strategies, illustrating their Δ Self-Acc and Δ Neg-Acc scores (see more methods in A.10). First, we find dropping out neurons in multiple layers much more efficient in erasing the target sequence,

as the horizontal blue line shows a greater decrease in Self-Acc than the solid blue line, suggesting that memorized information is stored in a distributed fashion over many layers, not concentrated in a single layer. The only exception is dropping out neurons in Layer 1; however, it also greatly hurts Neg-Acc. The large memorization decreases on all sequences may imply that the bottom layers of LLMs mainly work on processing basic and general information [74], instead of focusing on a specific sequence.

3.6 Related Work and Discussion

Localization identifies function-specific components, including neurons [75, 76], layers [77], or subnetworks [78, 79, 80]. For example, Dai et al. [23] find knowledge neurons for each relational fact. Meng et al. [24] locate relational facts to middle FFNs, specifically when LLMs process the last token of the subject. Bayazit et al. [81] discover sparse knowledge subnetworks in GPT2 with a differentiable weight masking method. However, there is no standard approach to evaluate the effectiveness of localization methods. We are the first to systematically and directly compare different methods on LLMs of different sizes.

We take the view that LLM memorization of a sequence is different from learning a type of knowledge. Memorization is reproducing a long sequence (near) verbatim. In contrast, knowledge, often represented as a ¡subject, relation, object¿ triplet, occurs in variable contexts, where paraphrases are treated as equivalent expressions of the same knowledge. Localization of verbatim memorization helps unlearn private or copyrighted data, for example, Wu et al. [67] apply IG to localize and then erase private data from a BERT fine-tuned on Enron Email dataset [82]. Our DEL Benchmark differs from Wu et al. [67] in two main ways: (1) we delete sequences that LLMs have naturally memorized during pretraining, (2) we locate neurons for each sequence independently, rather than finding a shared set of neurons, as our collected datasets cover diverse sequences. Localization can also prevent overfitting: Maini et al. [83] drop out pre-allocated neurons tied to memorizing mislabeled

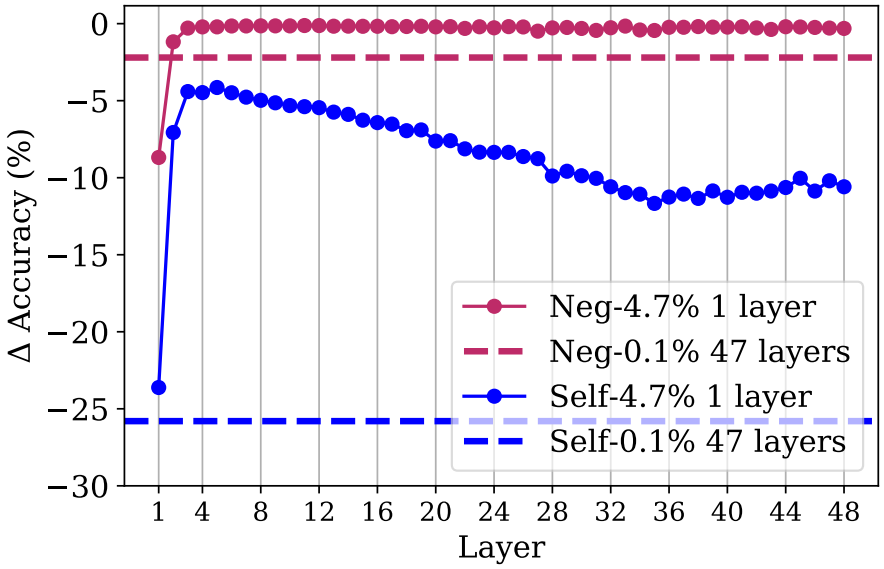


Figure 3.3: Dropout in one layer vs. multiple layers.

examples. In contrast with these works, we focus on benchmarking localization ability, since successful localization is the basis of its downstream applications.

3.7 Conclusion

We propose two benchmarking approaches to define the success of LLM localization, focusing on locating a small set of neurons in an LLM that are responsible for memorizing a sequence. The INJ Benchmark enables a direct evaluation of localization methods, while the DEL Benchmark evaluates methods on naturally memorized sequences, using dropout to measure localization success. The two benchmarks complement each other and show consistent rankings of methods. We find promising localization ability of all five methods we evaluate, especially for Hard Concrete. Meanwhile, all methods confuse memorized sequences in the same or related categories. This finding suggests a need for better localization methods and poses the open question of whether LLMs use a shared set of neurons to memorize related sequences such that perfect localization is not possible.

Chapter 4

When Parts Are Greater Than Sums: Individual LLM Components Can Outperform Full Models

4.1 Introduction

The rapid progress in large language models (LLMs) has popularized prompting, which guides LLMs to perform tasks with instructions or examples. Notably, in-context learning (ICL; [5]) adapts LLMs to a new task using only a few labeled examples without parameter updates. However, how LLMs react to the in-context examples is sometimes unintuitive [26]. Recently, Sclar et al. [27] and Voronov, Wolf, and Ryabinin [28] find that even for instruction-tuned [84] or very large models, adding a space or newline in prompts can greatly affect accuracy.

We look into the LLM internals to understand what causes the surprising behavior across various ICL settings. Our work stands in contrast to prior studies, which often treat LLMs as black boxes and alter either the input [31, 85] or output [29, 30]. We introduce a new view of ICL by decomposing the output of an LLM into the sum of individual contributions of MLPs and attention heads, denoted "components." Figure 4.1 reveals three types of curious

components: good-performing ones (blue) that individually perform well or even outperform the full model, bad-performing ones (red) that perform below chance, and label-biased ones (green) that predict the same label on the entire test set. We observe these three classes of components on Llama-2-7B, Llama-2-13B [86], Llama-3-8B [87], and Mistral-Instruct-7B [8] across 8 classification tasks.

We study the sensitivity of LLM components to multiple prompts formed by different demonstrations and templates. We also construct contrast sets of templates, pairs of similar templates that yield large differences in ICL accuracy. Despite large variance in full-model accuracy, we find that component accuracies correlate well across different demonstrations ($r = 0.80$ on average) and contrast set templates ($r = 0.57$). The top-performing components in contrast set pairs overlap and achieve decent accuracy even when the full model performs near random (Figure 4.2). Nonetheless, the component accuracies of two sampled templates are less correlated ($r = 0.34$). Further, good-performing components generalize well to out-of-distribution test sets. For instance, the top-1 component for MNLI outperforms the full Llama-2-13B model by 9.1% on MedNLI; Figure 4.1 also shows that components are transferrable from SST2 to Yelp. We conclude that components are relatively consistent in their behavior across prompts and datasets.

Inspired by our findings, we propose component reweighting. Compared to prior work that selects prompts from a large pool of labeled data to improve ICL accuracy [88], component reweighting softly selects components by learning weights from few-shot examples to scale component activations. Training these weights only involves learning a linear layer, which takes less than a minute on one CPU. Overall, component reweighting better utilizes the same labeled examples, improving over 24-shot ICL by $6.0\%, 2.2\%, 5.1\%, 1.6\%$ on Llama-2-7B, Llama-2-13B, Mistral-Instruct-7B, and Llama-3-8B, respectively. At the same time, it enjoys similar inference speed as 4-shot ICL.

Finally, we study the training dynamics of components using the Pythia pretraining checkpoints [48]. During pretraining, good-performing components emerge well before the

full model performs well. These findings suggest that LLMs acquire the internal ability to perform in-context learning early in training, but this ability only surfaces in the full model's behavior later on.

Overall, our work conducts extensive analysis of LLM internals, which motivates a practical method to improve ICL. We hope to inspire future work that further sheds light on LLM internals in order to improve performance. Our implementation is available at `https://github.com/terarachang/LLMDecomp`.

4.2 Decomposing the Transformer in ICL

We introduce a new view of in-context learning by decomposing the Transformer architecture [6]. Our decomposition is exact, a mathematically equivalent formula for the model's outputs, and enables us to analyze model internals without training additional parameters (unlike, e.g., probing). We first discuss what our new view offers over the standard view of ICL, and then walk through the mathematical details.

4.2.1 A New View of In-Context Learning

Standard view. An LLM performs in-context learning (ICL) on a task based on a few demonstrations without training, where each demonstration is a templated example (x, y) consisting of an input x and a label word y. We refer to a sequence of K demonstrations $[x_1, y_1, \ldots, x_K, y_K]$ as a *prompt*. The LLM makes predictions on a test input x_{test} conditioned on the prompt, denoted by $\arg\max_{y \in \mathcal{Y}} P(y|\textit{prompt}, x_{\text{test}})$, where $\mathcal{Y}$ is the set of possible label words in a classification task.

Our view. The residual stream of an LLM directly carries the information of the initial hidden state, every attention head, and every MLP, collectively named "components,"

towards the output layer. We view this information as the direct contributions[1] of components to the output logits, and derive a formula for logits, $\sum_j \mathbf{g}_j$, where $\mathbf{g}_j$ is the direct contribution of the component indexed by j. We can obtain the predictions of component j with $\arg\max_{y \in \mathcal{Y}} \mathbf{g}_j$, and then calculate its individual ICL accuracy. Specifically, we derive $\mathbf{g}_j = U \cdot C_j$ in Eq. 4.8 below, where U is the output embedding matrix and C_j is the post-layernorm activations of component j. We name the operation $(C_j \mapsto U \cdot C_j)$ as early decode, sharing the same spirit as nostalgebraist [89] and Geva et al. [47], which interpret hidden representations by decoding through U. Compared to the standard view, we can directly study the behavior of individual components (Figure 4.2), characterizing them and scaling their contributions to the model output.

4.2.2 A Walkthrough of the Decomposition

A Transformer of L layers consists of a multi-headed attention (MHA) and MLP in every layer. Let $a^{(l)} \in \mathbb{R}^d$ and $m^{(l)} \in \mathbb{R}^d$ be the output of the MHA and MLP at layer l, respectively. Due to residual connections, the hidden state $x^{(l)} \in \mathbb{R}^d$ is:

$$x^{(l)} = x^{(l-1)} + a^{(l)} + m^{(l)}, \tag{4.1}$$

$$x^{(L)} = x^{(0)} + \sum_{l=1}^{L} \left(a^{(l)} + m^{(l)}\right). \tag{4.2}$$

Note that GPT2-like LLMs apply layernorm before MHA and MLP [4]; thus, layernorm is already taken into account as part of the formula for computing $a^{(l)}$ and $m^{(l)}$ (see Appendix B.3).

An MHA $a^{(l)}$ is composed of n attention heads:

$$a^{(l)} = W_o^{(l)} \cdot \text{Concat}([h_1^{(l)}, \ldots, h_n^{(l)}]) \tag{4.3}$$

[1] In comparison, a component has indirect contributions to the output by affecting other components in later layers [45]. This paper focuses on direction contributions.

		SST2	BoolQ	QQP	WiC	RTE	MNLI	AGNews	ARC-Easy	Avg.
Llama2 7B	FULL	$75.8_{18.1}$	$69.2_{12.0}$	$61.3_{9.9}$	$52.4_{3.0}$	$\mathbf{68.9}_{3.2}$	$34.4_{1.7}$	$70.0_{19.9}$	$\mathbf{57.5}_{14.4}$	61.2
	ORACLE-T1	$\mathbf{91.7}_{0.9}$	$\mathbf{69.7}_{7.7}$	$\mathbf{67.8}_{4.3}$	$\mathbf{57.8}_{1.1}$	$64.6_{2.7}$	$\mathbf{46.3}_{3.3}$	$\mathbf{80.8}_{5.2}$	$54.5_{10.1}$	**66.6**
	ORACLE-B1	$12.1_{2.7}$	$34.1_{7.3}$	$32.5_{3.9}$	$42.9_{1.2}$	$34.7_{2.8}$	$24.1_{2.4}$	$3.0_{1.1}$	$12.7_{4.2}$	24.5
Llama2 13B	FULL	$89.0_{5.3}$	$\mathbf{77.6}_{6.8}$	$71.0_{6.8}$	$55.0_{3.8}$	$75.1_{2.3}$	$45.7_{7.9}$	$70.8_{20.6}$	$\mathbf{73.2}_{13.7}$	69.7
	ORACLE-T1	$\mathbf{92.5}_{0.6}$	$77.5_{6.0}$	$\mathbf{73.5}_{2.9}$	$\mathbf{60.4}_{1.2}$	$\mathbf{75.7}_{2.3}$	$\mathbf{56.4}_{4.7}$	$\mathbf{84.6}_{3.6}$	$73.1_{7.9}$	**74.2**
	ORACLE-B1	$8.2_{1.0}$	$27.1_{9.7}$	$31.8_{3.4}$	$39.5_{1.6}$	$27.9_{2.8}$	$18.6_{2.6}$	$1.8_{0.9}$	$5.4_{3.5}$	20.0
Mistral Ins 7B	FULL	$90.1_{2.9}$	$\mathbf{81.3}_{2.1}$	$70.9_{7.2}$	$58.5_{4.2}$	$80.5_{1.7}$	$56.1_{5.0}$	$83.0_{5.7}$	$\mathbf{79.8}_{1.4}$	75.0
	ORACLE-T1	$\mathbf{91.9}_{0.7}$	$80.8_{2.0}$	$\mathbf{75.6}_{2.6}$	$\mathbf{60.6}_{2.2}$	$\mathbf{81.3}_{0.8}$	$\mathbf{61.5}_{3.3}$	$\mathbf{83.7}_{4.3}$	$78.5_{2.2}$	**76.7**
	ORACLE-B1	$8.1_{0.9}$	$19.5_{2.5}$	$25.8_{4.1}$	$39.3_{2.8}$	$20.0_{1.7}$	$14.6_{2.9}$	$1.8_{0.7}$	$4.6_{1.3}$	16.7
Llama3 8B	FULL	$91.4_{1.7}$	$\mathbf{79.2}_{7.2}$	$74.0_{8.0}$	$58.7_{4.7}$	$\mathbf{76.5}_{2.2}$	$59.4_{3.7}$	$\mathbf{84.0}_{6.6}$	$\mathbf{87.4}_{5.5}$	76.3
	ORACLE-T1	$\mathbf{92.3}_{1.0}$	$77.4_{7.3}$	$\mathbf{77.4}_{3.7}$	$\mathbf{64.5}_{2.7}$	$76.3_{2.9}$	$\mathbf{60.7}_{1.5}$	$81.4_{5.7}$	$86.0_{5.9}$	**77.0**
	ORACLE-B1	$9.0_{0.9}$	$22.5_{7.4}$	$23.5_{3.9}$	$36.7_{3.3}$	$23.4_{2.1}$	$10.0_{4.2}$	$1.4_{0.6}$	$1.9_{0.9}$	16.0
	RANDOM	50.0	50.0	50.0	50.0	50.0	33.3	25.0	25.0	41.7

Table 4.1: $\{3, 4\}$-shot ICL accuracy of 8 tasks and the average accuracy (**Avg.**). We run 15 prompts for each task (see Section 4.3) and report the mean accuracy and standard deviation. We show the existence of good components (ORACLE-T1) inside LLMs that individually perform on par with the full model (FULL) on diverse tasks. Similarly, there exist bad components (ORACLE-B1) that perform substantially below chance (RANDOM).

for $h_i^{(l)} \in \mathbb{R}^{d_{\text{head}}}$ a head and $W_o^{(l)} \in \mathbb{R}^{d \times n d_{\text{head}}}$ the output projection in MHA aggregating all heads. Elhage et al. [43] rewrite Eq. 4.3 by segmenting $W_o^{(l)}$ into n matrices $W_{o_i}^{(l)} \in \mathbb{R}^{d \times d_{\text{head}}}$:

$$a^{(l)} = \sum_{i=1}^{n} \left(W_{o_i}^{(l)} \cdot h_i^{(l)} \right) = \sum_{i=1}^{n} \tilde{h}_i^{(l)}, \tag{4.4}$$

$$\text{where } [W_{o_1}^{(l)}, \ldots, W_{o_n}^{(l)}] = W_o^{(l)} \tag{4.5}$$

Thus, we can treat each head as a single component adding $\tilde{h}_i^{(l)} = W_{o_i}^{(l)} \cdot h_i^{(l)}$ to the residual stream.

Finally, through the output embedding matrix $U \in \mathbb{R}^{|\text{Vocab}| \times d}$, the output logits are:

$$\begin{aligned} \text{logits} &= U \cdot \text{LN}(x^{(L)}) \\ &= U \cdot \text{LN}\left(x^{(0)} + \sum_{l=1}^{L} \sum_{i=1}^{n} \tilde{h}_i^{(l)} + \sum_{l=1}^{L} m^{(l)} \right) \\ &= U \cdot \text{LN}\left(\sum_{j=1}^{1+L \times n+L} z_j \right), \end{aligned} \tag{4.6}$$

where $z = [x^{(0)}, \tilde{h}_1^{(1)}, \ldots, \tilde{h}_n^{(L)}, m^{(1)}, \ldots, m^{(L)}]$ in Eq. 4.6 and we index every term in the summation with j. $\text{LN}(\cdot)$ denotes the final layernorm, specifically, RMSNorm [90] for LLMs in our paper (see Appendix B.3). In Eq. 4.6, $\text{LN}(\sum_j z_j) = \frac{\sum_j z_j}{\text{RMS}(\sum_j z_j)} \odot \gamma$, where RMS denotes root mean square, $\odot$ denotes element-wise multiplication, and $\gamma \in \mathbb{R}^d$ is the affine parameters. By pre-computing $\hat{\gamma} = \frac{\gamma}{\text{RMS}(\sum_j z_j)}$, we have:

$$\text{logits} = U \cdot \left(\sum_j z_j \odot \hat{\gamma} \right) \tag{4.7}$$

$$= \sum_j U \cdot C_j, \text{ where } C_j = z_j \odot \hat{\gamma} \tag{4.8}$$

We refer to all $C_j \in \mathbb{R}^d$ as the component activations, which include the activations of attention heads and MLPs after the final layernorm.[2] Now that we have broken down the Transformer output into simple additions in Eq. 4.8, we can easily analyze the direct contribution of each component to the logits through the residual stream, $\mathbf{g}_j = U \cdot C_j$.

In ICL, we only need to do the decomposition when LLMs start to generate, i.e, when processing the last token of the input. The computations on the other tokens are the same as the standard ICL. In all our experiments, we use single-token label words. We use multiple templates from Bach et al. [91] that cover diverse label words for each task.

4.3 Characterizing Components for ICL

We conduct in-context learning across 8 classification tasks on 4 LLMs: Llama-2-7B, Llama-2-13B, Mistral-Instruct-7B, and Llama-3-8B. ICL is sensitive to prompts, so we randomly sample 5 disjoint sets of demonstrations formatted with 3 templates and report the standard deviation across the 15 runs. To avoid majority and recency biases [29], each prompt consists of the same number of demonstrations from every class in shuffled order. We use $K = 3$

[2]Empirically, we find that $x^{(0)}$ has near-random ICL accuracy on all the tasks, so we omit it in the rest of the paper.

demonstrations for 3-way classification tasks and $K = 4$ for the other tasks. Except for Section 4.5.1, we refer to $K = \{3, 4\}$ without further notice. We sample 2000 examples with balanced labels as the test set for every task. Please see Appendix B.1 for details about the tasks and templates.

4.3.1 Good and Bad-Performing Components

Across all the tasks and LLMs, we observe good-performing components that perform well or even outperform the full model, and bad-performing components that individually perform much worse than chance (blue and red dots in Figure 4.1, respectively). Table 4.1 compares the full model (FULL) with the top-1 (ORACLE-T1) and bottom-1 (ORACLE-B1) components selected on the test set. On average, ORACLE-T1 outperforms FULL by 5.4%, 4.5%, 1.7%, 0.7% on Llama-2-7B, Llama-2-13B, Mistral-Instruct-7B, and Llama-3-8B, respectively; ORACLE-B1 underperforms random guessing (RANDOM) by 17.2%, 20.7%, 25.0%, 25.7%.

4.3.2 Label-Biased Components

Besides good and bad-performing components, we also observe label-biased components, which predict a certain label on the entire test set (the green dots in Figure 4.1). These components exist in all the tasks and LLMs we study, accounting for 29.1%, 26.4%, 22.8%, 29.7% of components on average in Llama-2-7B, Llama-2-13B, Mistral-Instruct-7B, and Llama-3-8B, respectively (Table B.13). In Appendix B.2, we show that even when we prompt the model with all demonstrations of positive labels, the most biased component still insists on predicting "negative" on the entire test set, and vice versa.

4.3.3 Mechanistic Understanding of Bad-Performing Heads

Prior work studies the mechanism of certain components in LLMs, showing that there are negative mover attention heads that write in the opposite direction of the expected answer

[45] and copy suppression heads that suppress the prediction of a prior token in the context [92]. Inspired by them, we investigate the mechanism behind bad-performing heads identified by our decomposition. We focus on label tokens in the context, as Wang et al. [93] show that label words serve as anchors in ICL.

We conduct a case study on Llama-2-7B with 4-shot balanced in-context examples from SST2. We examine the bottom-5 attention heads that have the worst ICL accuracy on SST2. We find that three of these heads, L19H15, L15H14, and L18H9, assign top attention probabilities to all 4 label tokens of the 4-shot in-context examples when predicting test examples. Furthermore, despite their poor ICL accuracy, these heads actually assign higher attention to the correct in-context label tokens than the incorrect ones most of the time ($> 70\%$ of the test examples). In other words, when a test example has a positive label, these heads assign higher attention[3] to the tokens "positive" in the context than the tokens "negative". We also observe that the more the heads attend to "positive" in the context, the lower the inner product between the head and the output embedding of the token "positive", with the correlation $r = -0.97, -0.96, -0.89$ for L19H15, L15H14, and L18H9, respectively.

In summary, we show that some bad-performing heads attend highly to prior label tokens and decrease the output probability of the correct one, which shares similarities with the copy suppression heads and negative mover heads [92, 45]. However, we do not observe similar behavior in other tasks, where the bad-performing heads usually attend to "¡s¿", "?", or "\n". We invite future work to further analyze how bad-performing heads function in general.

4.4 Transferability of Components

We observe moderate to high component transferability across demonstrations, minimally contrastive templates, and data distributions, whereas there is little transferability across

[3]We average the attention probabilities of the same label tokens and then compare the average ones of the two labels.

		SST2	BoolQ	QQP	AGNews	ARC
Corr	(1) Demo	0.81	0.84	0.60	0.89	0.88
	(2) Temp	0.40	0.16	0.03	0.68	0.44
	(3) Cst T	0.72	0.63	0.23	0.82	0.46
IoU	(1) Demo	0.36	0.74	0.27	0.63	0.70
	(2) Temp	0.12	0.01	0.01	0.20	0.20
	(3) Cst T	0.40	0.23	0.02	0.36	0.45

Table 4.2: The average correlation and IoU between (1) two random sets of demonstrations, (2) two random templates, and (3) two minimally contrastive templates.

randomly sampled templates. Our decomposition uncovers hidden abilities of individual components when the full model performs poorly.

4.4.1 Transfer across Prompt Variants

We first measure the agreement in component accuracies between (1) two disjoint sets of demonstrations with a fixed template, (2) two randomly sampled templates with fixed demonstrations, and (3) two minimally-contrastive templates with fixed demonstrations. Recall that we have 5 sets of demonstrations and 3 templates in total (Section 4.3); here, we calculate the average agreement between every pair. For (3), we construct contrast sets [94] by minimally editing the worst-performing template out of the 3 templates into a good template, which yields at least 10% improvement in average accuracy. Our edits include adding a space, removing a newline, or changing label words (see Table B.12). We use two metrics to measure the agreement between each pair: Pearson correlation of the accuracies of all components and the intersection over union (IoU) on the sets of top-5 components, which measures whether the top-performing components of the pair overlap.

Table 4.2 summarizes the results on Llama-2-7B; Appendix B.6 shows similar findings on other models. (1) The accuracies of the internal components are highly consistent across dif-

ferent choices of demonstrations, having strong correlations and an average of 0.54 IoU. (2) The components have much weaker agreement across randomly sampled templates, having a near 0 IoU on BoolQ and QQP. (3) Nevertheless, there is agreement between minimally contrastive templates (Cst T), with an average correlation of 0.57 across tasks, despite contrasting full-model accuracy. For example, Figure 4.2 demonstrates that full-model accuracy changes dramatically (39% vs 89%) in a minimal pair of templates, but internal components have a high correlation of 0.81 and the pair shares top-performing components. Combining (2) and (3) suggests components behave similarly on similar templates, but this similarity decreases as the templates diverge.

4.4.2 Transfer to Out-of-Distribution Test Sets

We further study whether the best component selected on the test set can still perform well on an out-of-distribution (OOD) test set. We name this method, which uses a single component to make predictions, as TRANSFER-1. Specifically, we study component transferability from SST2 to Yelp-polarity, MNLI to MedNLI, and BoolQ to BoolQ Contrast Set. We compare TRANSFER-1 with using the full model (FULL) on the OOD test sets. To understand the best possible TRANSFER-1 accuracy, we also report the best component accuracy directly selected on the OOD set, ORACLE-1.

Table 4.3 shows that TRANSFER-1 closely matches ORACLE-1 overall, suggesting that the top-performing components are transferable across data distribution. Moreover, TRANSFER-1 sometimes outperforms FULL, especially on Llama2 models, showing the hidden abilities of the internal components.

4.4.3 Transfer between Two Opposite Tasks

We conduct a case study of component transferability across instructions using Task069 and Task070 of Super-NaturalInstructions [95], both of which are binary abductive NLI tasks [96]. The instruction for Task069 asks for correct answers, while Task070 asks for incorrect

ones ("pick the one that makes less sense;" see Figure B.4 for the full instructions). Examples in the two tasks are not parallel.

We find that Mistral-Instruct-7B achieves good accuracy across 15 runs on Task069 (76.8 ± 2.4), but below chance on Task070 (40.6 ± 5.4). We observe a strong negative correlation, $r = -0.60$ on average, between the component accuracies of the two tasks. The worst-performing components in Task069 become the top-performing in Task070 and vice versa. The correlation suggests that the model has the ability to solve Task070, but misunderstands negation. Thus, we apply the TRANSFER-1 method (Section 4.4.2) but select the worst-performing component from Task069 and then calculate its individual accuracy on Task070. TRANSFER-1 achieves 58.7 ± 4.8 accuracy across the 15 runs, an **improvement of 18.1**% over the full model. These results suggest that components behave consistently even across tasks with opposite instructions, as the active components in Task069 are also active in Task070.

4.5 Component Reweighting

4.5.1 Proposed Method

Our findings in Section 4.4 show the promising direction of selecting internal components to improve ICL. Therefore, we propose a method that reweights components by learning a weight $w_j \in \mathbb{R}$ on every component activation C_j. Reweighting is a soft version of selection, which can be learned by gradient descent on very few examples.

Given K labeled examples, instead of using all of them as ICL demonstrations, we divide them into a demonstration set $\mathcal{D}_{\text{demo}}$ and a training set $\mathcal{D}_{\text{train}}$. We first randomly sample $K' = \{3, 4\}$ examples with balanced labels as demonstrations and use the remaining examples as $\mathcal{D}_{\text{train}}$ to train the component weights. Specifically, we can rewrite Eq. 4.8 as logits $= \sum_j w_j(U \cdot C_j)$, where $w_j = 1$ for all j. Because of the existence of good and bad-performing components, weighing all components equally may not be optimal. Therefore, we tune the

		Yelp-polarity	MedNLI	BoolQ Cst
Llama2 7B	FULL	$84.7_{15.4}$	$34.3_{1.7}$	$\mathbf{64.9}_{9.8}$
	TRANSFER-1	$\mathbf{94.9}_{3.1}$	$\mathbf{42.6}_{4.7}$	$64.3_{7.9}$
	ORACLE-1	$96.9_{0.7}$	$48.8_{2.3}$	$66.2_{5.7}$
Llama2 13B	FULL	$95.9_{1.4}$	$46.8_{9.6}$	$72.0_{7.6}$
	TRANSFER-1	$\mathbf{96.0}_{1.8}$	$\mathbf{55.9}_{4.0}$	$\mathbf{72.3}_{6.5}$
	ORACLE-1	$97.1_{0.4}$	$57.0_{3.7}$	$73.0_{6.1}$
Mistral Ins 7B	FULL	$\mathbf{97.0}_{0.5}$	$57.3_{5.7}$	$\mathbf{74.6}_{3.5}$
	TRANSFER-1	$95.6_{1.6}$	$\mathbf{61.9}_{4.8}$	$73.7_{3.7}$
	ORACLE-1	$97.1_{0.4}$	$62.7_{4.1}$	$74.5_{3.6}$
Llama3 8B	FULL	$\mathbf{97.8}_{0.4}$	$61.0_{2.2}$	$\mathbf{77.3}_{7.5}$
	TRANSFER-1	$95.9_{4.4}$	$\mathbf{61.3}_{0.8}$	$73.9_{8.4}$
	ORACLE-1	$97.9_{0.5}$	$61.6_{0.6}$	$74.8_{8.9}$

Table 4.3: The average ICL accuracy and standard deviation on OOD test sets. The components selected on the in-distribution test sets (TRANSFER-1) can transfer to OOD sets, performing similarly to the oracle components (ORACLE-1) directly selected on the OOD sets.

weights $w \in \mathbb{R}^N$ of N components on $\mathcal{D}_{\text{train}}$ with cross-entropy loss and L_1 regularization, while keeping the LLM frozen:

$$\mathcal{L} = \sum_{(x,y)\in\mathcal{D}_{\text{train}}} -\log P_{rw}(y|x) + \lambda\|w\|_1, \tag{4.9}$$
$$P_{rw}(y|x) = \text{softmax}\left(\sum_{j=1}^{N} w_j \, (U_{\mathcal{Y}} \cdot C_j)\right)_y,$$

where $U_{\mathcal{Y}} \in \mathbb{R}^{|\mathcal{Y}|\times d}$ is a submatrix of U that comprises the output embeddings of label words, P_{rw} is the probability distribution of the LLM after reweighting, and λ is the hyperparameter of the L_1 loss to encourage sparsity on the component weights. We obtain the activations $\{C_j\}_{j=1}^N$ of all components in one K'-shot forward pass, computed on the prompt derived

Algorithm 2 Component Reweighting

1: **Input:** K labeled examples, a test set $\mathcal{D}_{\text{test}}$, a set of label words $\mathcal{Y}$, an LLM $\mathcal{M}$, the number of components N
2: **Output:** $\mathcal{Z}$, the predictions of $\mathcal{M}$ on $\mathcal{D}_{\text{test}}$
3: Split K examples into a *prompt* consists of K' demonstrations and a training set $\mathcal{D}_{\text{train}}$ of $K - K'$ examples
4: $U_{\mathcal{Y}} \leftarrow$ concatenate the output embeddings of $\mathcal{Y}$ in $\mathcal{M}$
5: Initialize $\mathcal{G}^{\text{train}} \leftarrow \varnothing$
6: **for** $(x, y) \in \mathcal{D}_{\text{train}}$ **do**
7: $\quad \{C_j\}_{j=1}^{N} \leftarrow \mathcal{M}(prompt, x)$ ▷ K'-shot ICL
8: $\quad$ **for** $j \leftarrow 1$ to N **do**
9: $\quad\quad \mathcal{G}^{\text{train}} \leftarrow \mathcal{G}^{\text{train}} \cup (U_{\mathcal{Y}} \cdot C_j)$ ▷ early decode
10: $\quad$ **end for**
11: **end for**
12: Initialize $w \leftarrow [1, \ldots, 1] \in \mathbb{R}^N$
13: Train the weights w on $\mathcal{G}^{\text{train}}$ with Eq. 4.9
14: Initialize $\mathcal{Z} \leftarrow \varnothing$ ▷ Start Inference
15: **for** $(x, y) \in \mathcal{D}_{\text{test}}$ **do**
16: $\quad \{C_j\}_{j=1}^{N} \leftarrow \mathcal{M}(prompt, x)$ ▷ K'-shot ICL
17: $\quad$ Initialize $\mathbf{g} \leftarrow [0, \ldots, 0] \in \mathbb{R}^{|\mathcal{Y}|}$
18: $\quad$ **for** $j \leftarrow 1$ to N **do** ▷ Test-Time Overhead
19: $\quad\quad \mathbf{g} \leftarrow \mathbf{g} + w_j (U_{\mathcal{Y}} \cdot C_j)$ ▷ early decode
20: $\quad$ **end for**
21: $\quad \hat{y} \leftarrow \arg\max_{y \in \mathcal{Y}} \mathbf{g}$
22: $\quad \mathcal{Z} \leftarrow \mathcal{Z} \cup \hat{y}$
23: **end for**
24: **return** $\mathcal{Z}$

from $\mathcal{D}_{\text{demo}}$, followed by x. Our method scales each component's direct contributions to the logits ($U_{\mathcal{Y}} \cdot C_j \in \mathbb{R}^{|\mathcal{Y}|}$) by w_j. In practice, we cache these contributions on the entire training set as input features to the linear layer w, which allows us to discard the entire LLM while training w (line 9 and 13 in Algorithm 2), saving tremendous training time and GPU memory. The cache only requires $O(|\mathcal{Y}| \times N \times |\mathcal{D}_{\text{train}}|)$ space. At inference time, the computation overhead of our method over K'-shot ICL is to early decode N components and apply the learned weights, i.e., $\sum_{j=1}^{N} w_j \, (U_{\mathcal{Y}} \cdot C_j)$. As both $|\mathcal{Y}|$ and N are small ($N < 2000$ for all LLMs in this paper), the overhead is negligible compared to the computation of the backbone LLM itself.

4.5.2 Baselines

Standard ICL. The simplest baseline is to use all the K labeled examples as demonstrations. Since the other methods use K' examples as demonstrations, we report the accuracy of standard K'-shot ICL using the same $\mathcal{D}_{\text{demo}}$ for reference.

Prompt Selection. Liu et al. [88] improve ICL accuracy by selecting demonstrations from a pool of labeled data for each test example. Here, we select from the given K labeled examples. Following Rubin, Herzig, and Berant [97], we use SBERT [98] to encode examples into sentence embeddings and select the $K' = \{3, 4\}$ nearest neighbors under cosine similarity as the demonstrations for each test example.

Calibration. As LLMs tend to predict a certain class over others, Zhao et al. [29] reweight the output class probabilities. They use context-free inputs, such as "N/A", to calibrate the probability distribution. However, Fei et al. [99] and Zhou et al. [100] find context-free inputs sometimes ineffective, because in-domain context is important for calibration. Thus, we introduce CALIB+, which calibrates the original probabilities $\mathbf{p} \in \mathbb{R}^{|\mathcal{Y}|}$ with a training set of in-distribution labeled examples, $\mathcal{D}_{\text{train}}$. We train the calibration weights $\mathbf{v} \in \mathbb{R}^{|\mathcal{Y}|}$ on $\mathcal{D}_{\text{train}}$ with cross-entropy loss and obtain the calibrated probabilities $\hat{\mathbf{p}} = \text{softmax}(\mathbf{v} \cdot \mathbf{p})$. For direct comparisons, we split the K examples into the same $\mathcal{D}_{\text{demo}}$ and $\mathcal{D}_{\text{train}}$ sets as component reweighting for CALIB+, where $|\mathcal{D}_{\text{demo}}| = K'$. We include the training details of both methods in Appendix B.8.

4.5.3 Results

We set $K = \{12, 24\}$. Table 4.4 compares our component reweighting (COMPRW) with standard ICL (STANDARD), prompt selection (PROMPTS), and calibration (CALIB+). First, we find that simply increasing the number of demonstrations from 4 to 24 has limited improvements in ICL accuracy, while the longer prompt greatly increases the inference time. For

		SST2	BoolQ	QQP	WiC	RTE	MNLI	AGNews	ARC-Easy	Avg.
Llama-2-7B	STANDARD 3, 4	75.8 $_{18.1}$	69.2 $_{12.0}$	61.3 $_{9.9}$	52.4 $_{3.0}$	68.9 $_{3.2}$	34.4 $_{1.7}$	70.0 $_{19.9}$	57.5 $_{14.4}$	61.2
	STANDARD 12	77.8 $_{19.6}$	**71.6** $_{8.0}$	63.6 $_{7.8}$	52.5 $_{2.4}$	**71.1** $_{2.1}$	37.0 $_{2.8}$	69.0 $_{20.8}$	**59.6** $_{13.9}$	62.8
	PROMPTS 12	73.8 $_{19.2}$	69.4 $_{10.5}$	62.2 $_{6.1}$	53.1 $_{2.7}$	65.5 $_{1.8}$	35.5 $_{1.6}$	59.1 $_{28.7}$	58.7 $_{11.9}$	59.7
	CALIB+ 12	85.1 $_{6.0}$	69.2 $_{13.6}$	**73.6** $_{6.1}$	55.1 $_{5.1}$	70.3 $_{2.7}$	45.5 $_{7.8}$	77.8 $_{12.2}$	58.6 $_{14.6}$	66.9
	COMPRW 12	**88.5** $_{2.8}$	70.4 $_{11.2}$	71.4 $_{5.4}$	**56.3** $_{3.4}$	70.0 $_{2.8}$	**48.3** $_{4.8}$	**87.4** $_{2.3}$	58.3 $_{13.6}$	**68.8**
	STANDARD 24	77.8 $_{19.5}$	71.6 $_{7.3}$	66.4 $_{5.0}$	53.2 $_{3.3}$	**71.9** $_{1.5}$	39.9 $_{3.6}$	71.1 $_{20.0}$	58.3 $_{16.2}$	63.8
	PROMPTS 24	74.2 $_{20.4}$	68.9 $_{10.2}$	62.1 $_{4.9}$	53.6 $_{1.9}$	64.8 $_{0.9}$	36.4 $_{1.5}$	57.5 $_{30.2}$	58.0 $_{12.5}$	59.4
	CALIB+ 24	87.6 $_{5.0}$	70.3 $_{11.9}$	**73.4** $_{5.5}$	55.8 $_{4.9}$	70.4 $_{2.7}$	46.4 $_{6.7}$	78.4 $_{11.8}$	**59.2** $_{14.4}$	67.7
	COMPRW 24	**90.6** $_{1.7}$	**71.7** $_{9.4}$	71.9 $_{4.4}$	**57.1** $_{3.0}$	70.0 $_{4.1}$	**49.8** $_{4.0}$	**88.1** $_{2.1}$	58.8 $_{13.6}$	**69.8**
Llama-2-13B	STANDARD 3, 4	89.0 $_{5.3}$	77.6 $_{6.8}$	71.0 $_{6.8}$	55.0 $_{3.8}$	75.1 $_{2.3}$	45.7 $_{7.9}$	70.8 $_{20.6}$	73.2 $_{13.7}$	69.7
	STANDARD 12	**91.3** $_{1.9}$	78.1 $_{7.4}$	70.5 $_{7.3}$	**59.6** $_{2.4}$	74.4 $_{3.5}$	55.1 $_{6.2}$	84.7 $_{7.8}$	71.2 $_{16.4}$	73.1
	PROMPTS 12	83.8 $_{10.2}$	74.9 $_{6.6}$	64.6 $_{5.7}$	57.0 $_{2.1}$	69.5 $_{3.5}$	48.1 $_{5.4}$	64.4 $_{29.6}$	74.2 $_{9.3}$	67.1
	CALIB+ 12	89.4 $_{3.2}$	**78.4** $_{6.1}$	72.1 $_{4.1}$	58.1 $_{5.1}$	75.3 $_{1.9}$	57.3 $_{4.5}$	81.5 $_{8.7}$	74.7 $_{9.3}$	73.3
	COMPRW 12	89.1 $_{3.2}$	77.7 $_{6.7}$	**72.7** $_{3.3}$	58.7 $_{4.0}$	**76.2** $_{2.0}$	**60.2** $_{3.7}$	**88.1** $_{1.7}$	**76.2** $_{6.8}$	**74.9**
	STANDARD 24	**91.9** $_{0.6}$	77.7 $_{8.2}$	69.5 $_{8.5}$	60.6 $_{1.6}$	74.7 $_{3.3}$	58.2 $_{7.0}$	85.8 $_{4.4}$	69.1 $_{17.7}$	73.5
	PROMPTS 24	81.9 $_{13.2}$	75.1 $_{5.7}$	64.9 $_{4.8}$	57.3 $_{1.8}$	69.5 $_{1.7}$	49.8 $_{5.1}$	65.2 $_{28.9}$	74.2 $_{9.4}$	67.2
	CALIB+ 24	90.7 $_{2.1}$	**78.6** $_{6.2}$	73.1 $_{4.3}$	**59.5** $_{3.2}$	75.9 $_{1.9}$	58.4 $_{2.8}$	82.0 $_{8.4}$	75.2 $_{9.1}$	74.2
	COMPRW 24	91.0 $_{1.8}$	78.2 $_{6.4}$	**74.2** $_{3.1}$	58.5 $_{4.1}$	**77.1** $_{1.8}$	**62.0** $_{3.7}$	**88.8** $_{1.4}$	**76.1** $_{7.2}$	**75.7**
Mistral-Instruct-7B	STANDARD 3, 4	90.1 $_{2.9}$	81.3 $_{2.1}$	70.9 $_{7.2}$	58.5 $_{4.2}$	80.5 $_{1.7}$	56.1 $_{5.0}$	83.0 $_{5.7}$	79.8 $_{1.4}$	75.0
	STANDARD 12	91.4 $_{0.9}$	81.2 $_{2.2}$	67.9 $_{8.7}$	57.7 $_{2.8}$	79.1 $_{1.6}$	57.2 $_{3.6}$	85.4 $_{3.6}$	77.7 $_{5.6}$	74.7
	PROMPTS 12	90.3 $_{2.5}$	81.1 $_{1.9}$	68.7 $_{5.8}$	57.1 $_{2.7}$	79.1 $_{1.6}$	56.7 $_{3.2}$	84.9 $_{3.0}$	79.0 $_{3.0}$	74.6
	CALIB+ 12	**91.5** $_{1.6}$	**81.3** $_{1.8}$	**75.8** $_{2.6}$	58.3 $_{6.6}$	81.0 $_{1.3}$	61.9 $_{4.7}$	85.4 $_{4.0}$	**79.6** $_{1.6}$	76.9
	COMPRW 12	89.9 $_{2.7}$	80.7 $_{2.7}$	75.1 $_{2.9}$	**60.0** $_{4.9}$	**81.1** $_{1.3}$	**64.7** $_{4.6}$	**87.6** $_{2.1}$	79.2 $_{1.2}$	**77.3**
	STANDARD 24	91.2 $_{1.0}$	80.8 $_{2.3}$	65.3 $_{8.4}$	57.4 $_{4.0}$	75.6 $_{1.7}$	56.6 $_{6.5}$	85.8 $_{4.3}$	68.8 $_{16.9}$	72.7
	PROMPTS 24	90.5 $_{2.6}$	**81.3** $_{2.0}$	68.9 $_{5.6}$	57.1 $_{2.1}$	79.1 $_{1.7}$	57.4 $_{3.1}$	86.0 $_{2.1}$	78.7 $_{3.3}$	74.9
	CALIB+ 24	**91.6** $_{1.5}$	80.9 $_{2.0}$	76.1 $_{2.4}$	59.5 $_{5.4}$	81.2 $_{0.9}$	62.7 $_{4.3}$	85.9 $_{3.7}$	**80.1** $_{1.2}$	77.2
	COMPRW 24	90.8 $_{1.8}$	80.6 $_{2.1}$	**76.4** $_{1.7}$	**60.7** $_{4.4}$	**81.6** $_{1.0}$	**65.3** $_{3.4}$	**88.0** $_{1.8}$	79.0 $_{1.6}$	**77.8**
Llama-3-8B	STANDARD 3, 4	91.4 $_{1.7}$	79.2 $_{7.2}$	74.0 $_{8.0}$	58.7 $_{4.7}$	76.5 $_{2.2}$	59.4 $_{3.7}$	84.0 $_{6.6}$	87.4 $_{5.5}$	76.3
	STANDARD 12	**92.2** $_{0.6}$	**79.6** $_{6.9}$	73.1 $_{5.0}$	**63.3** $_{2.4}$	77.5 $_{2.2}$	62.7 $_{3.8}$	87.7 $_{2.0}$	82.1 $_{18.1}$	77.3
	CALIB+ 12	91.1 $_{1.5}$	79.2 $_{5.8}$	**77.9** $_{3.3}$	60.5 $_{9.3}$	77.3 $_{2.1}$	65.2 $_{3.2}$	86.4 $_{4.7}$	**87.7** $_{4.0}$	78.2
	COMPRW 12	90.7 $_{2.0}$	78.3 $_{6.7}$	77.2 $_{2.9}$	61.8 $_{6.4}$	**78.0** $_{1.8}$	**66.9** $_{2.4}$	**89.1** $_{1.0}$	87.4 $_{3.8}$	**78.7**
	STANDARD 24	**92.2** $_{0.8}$	78.2 $_{7.2}$	78.0 $_{2.0}$	**63.8** $_{1.8}$	76.2 $_{3.0}$	66.4 $_{2.3}$	87.9 $_{1.9}$	80.6 $_{18.9}$	77.9
	CALIB+ 24	91.7 $_{1.4}$	**80.0** $_{6.1}$	78.3 $_{3.4}$	**63.8** $_{2.9}$	78.1 $_{1.7}$	66.0 $_{2.4}$	86.7 $_{4.9}$	**87.7** $_{3.8}$	79.0
	COMPRW 24	91.6 $_{1.7}$	79.1 $_{6.9}$	**78.8** $_{2.7}$	63.7 $_{3.3}$	**78.5** $_{1.4}$	**67.4** $_{2.6}$	**89.5** $_{1.1}$	87.4 $_{3.2}$	**79.5**

Table 4.4: ICL accuracy of 8 classification tasks and the average accuracy (**Avg.**). The number after a method denotes the number of labeled data used. We run 15 prompts for each task (5 disjoint sets of K labeled data and 3 templates) and report the mean accuracy and standard deviation. COMPRW achieves the best average accuracy in all setups.

example, on Llama-2-7B, STANDARD 24 only improves the average accuracy by 2.6% over STANDARD 3, 4 and the accuracy even decreases on Mistral-Instruct. Second, PROMPTS performs the worst in most setups, likely because it is hard to find similar examples from a small pool of K examples, and a bad selection induces majority label biases. Third, both

calibration (CALIB+) and component reweighting (COMPRW) achieve substantially better accuracy than STANDARD $3, 4$ with little test-time overhead. Overall, COMPRW achieves the best average accuracy in all setups, outperforming STANDARD 12 by $6.0\%, 1.8\%, 2.6\%, 1.4$ on Llama-2-7B, Llama-2-13B, Mistral-Instruct-7B, and Llama-3-8B, respectively, and outperforming STANDARD 24 by $6.0\%, 2.2\%, 5.1\%, 1.6\%$, respectively. We run one-tailed paired t-tests comparing COMPRW with CALIB+ and find that p-values < 0.05 in all 8 setups (see Table B.14), showing that COMPRW performs significantly better CALIB+.

4.6 When Do Good Components Emerge?

We study the dynamics of components during pretraining by monitoring their accuracies on 32 checkpoints of Pythia-6.9B, uniformly spaced from the first to the last checkpoint. For each checkpoint, we run 4-shot ICL on AGNews with 3 templates $\times$ 3 sets of demonstrations. The demonstrations are balanced in labels with randomly shuffled orders. Figure 4.3 shows the average accuracy of the 9 runs shaded by the standard deviation.

While the full model (green) fluctuates and has a large variance across prompts, the top-1 components (solid blue) achieve good accuracy at an early step and plateau quickly. We also backtrack the top-1 components of different prompts at the last checkpoint (dashed blue), monitoring how they perform on average during pretraining. We observe that they are not the top components at the early stage (there are gaps between the two blue lines before the $75k$ steps), but start to perform steadily well from the middle stage. Our findings also hold on SST2 and Pythia-1.4B (see Figure B.6 in the appendix), suggesting that the model's ability to do a task emerges before it is apparent from the full model on these tasks.[4]

[4]On the other hand, Pythia models perform poorly on the other tasks over all checkpoints; thus, the training dynamics of the model components on challenging tasks remain unclear.

4.7 Related Work and Discussion

Improving ICL. Prior work shows that ICL performance varies greatly across different choices of demonstrations and templates [29, 101]. Specifically, Sclar et al. [27] and Voronov, Wolf, and Ryabinin [28] find no universally better prompt template that can transfer across tasks and models, implying that it is not easy to explain ICL through prompt engineering. While several approaches, such as prompt selection [88, 102, 103], prompt ensemble [104, 105, 28], and many-shot ICL [106], substantially improve accuracy, they treat LLMs like black boxes without understanding the internals. Besides, they greatly increase inference time or require a large set of labeled data, which deviates from true few-shot learning [107]. In comparison, our paper studies this problem by looking inside the LLMs. Rather than selecting prompts, we select components in a soft, learnable way. Our method only requires $\{12, 24\}$ examples and has negligible computation overhead over 4-shot ICL at inference.

Components Interpretation. Components interpretation studies the function of different components in a trained model [43, 108], where components could be neurons [75, 109, 76], attention heads [44], and MLPs [46]. To analyze the components, probing [110], knockout [111, 16, 112], patching [45, 113], and early decoding [89, 47] are widely used techniques. For example, Li et al. [114] train a linear probe on every attention head to discover the truthful heads inside LLMs. Michel, Levy, and Neubig [115] and Voita et al. [116] prune away a large percentage of attention heads and show that only a few are critical to the performance. Hendel, Geva, and Globerson [117], Liu et al. [118], Merullo, Eickhoff, and Pavlick [119], and Todd et al. [120] view ICL as compressing demonstrations into function vectors, where they remove the demonstrations and modify (*patch*) the LLM activations at certain layers with the function vectors at test time. Early decoding interprets the investigated components in the textual space by projecting them through the output embedding matrix [47]. Our model decomposition is based on early decoding and we share some similarities with prior work

[121, 122], especially in discovering individual components that perform well on a task. Our contributions lie in providing a new view of in-context learning by decomposition, which reveals the transferability of components across diverse ICL settings.

Our Method vs. Pruning. Our method caches the direct contributions of components to the outputs through the residual stream, i.e., logits $= \sum_j \mathbf{g}_j$. Thus, removing $\mathbf{g}_j$, the direct contribution of the component j, does not alter the contributions of the other components. In comparison, pruning a component changes the activations of the other components in later layers. In Appendix B.7, we show that pruning the good-performing components identified by our method greatly hurts the accuracy, meaning that pruning also defines these components as important [115].

4.8 Conclusion

We introduce a new perspective of ICL via decomposing the model output into the sum of individual contributions of components. We then identify three types of component characteristics across 3 LLMs and 8 classification tasks. Our extensive analyses reveal consistency in component accuracy across prompts and suggest the promising direction of improving ICL by selecting components. To this end, we propose component reweighting, which learns to scale components differently on few-shot examples. Our method achieves the best average accuracy compared to prior methods. We hope this work can deepen our grasp of LLMs while motivating more methods for practical use.

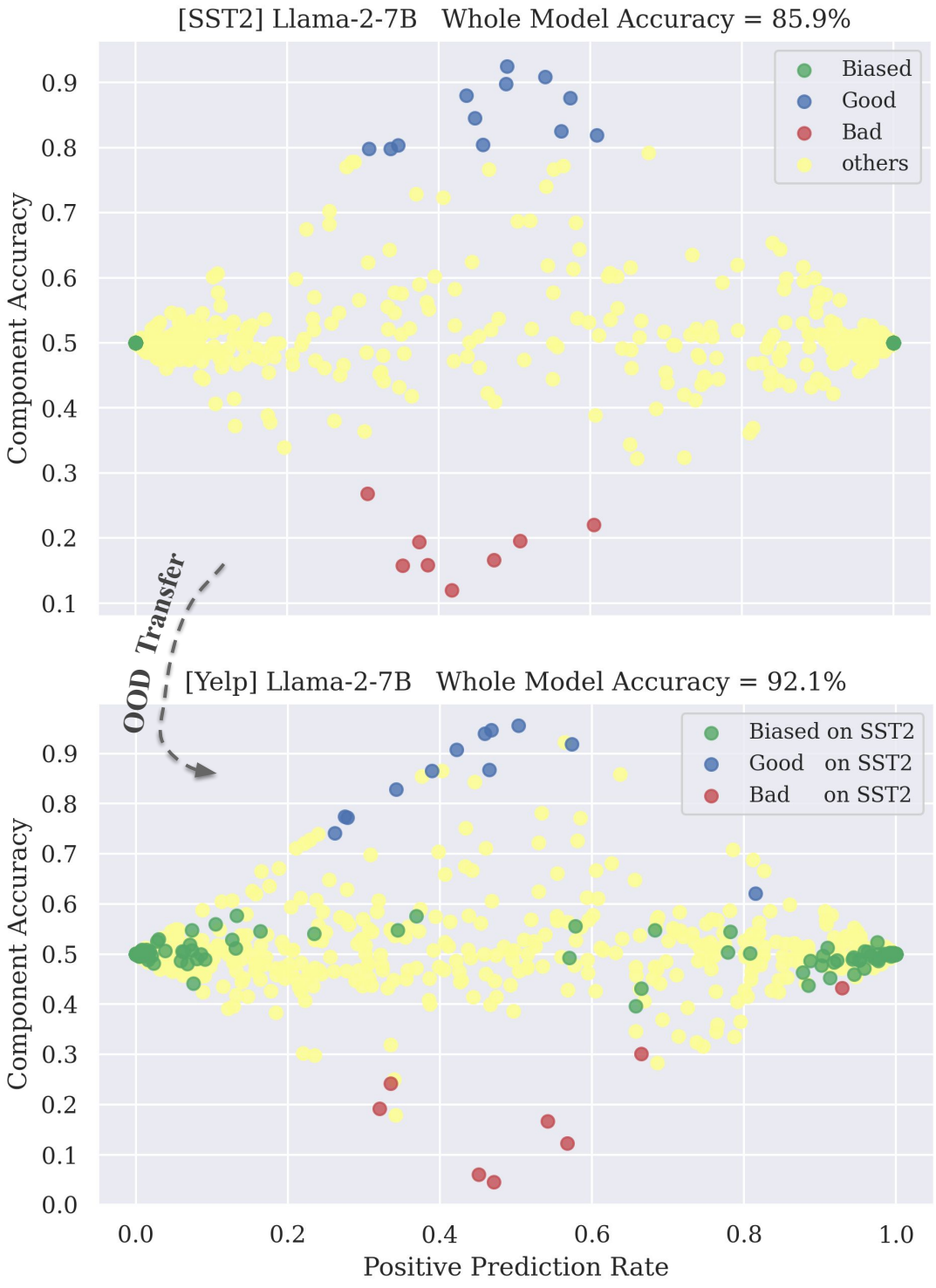


Figure 4.1: Each dot represents a component (attention head or MLP) under 4-shot ICL on Llama-2-7B. The x-axis shows how often a component predicts "positive" on the test set. **Up:** We discover good-performing (blue), bad-performing (red), and label-biased (green) components. **Down:** Most components identified on SST2 show similar characteristics on Yelp-polarity.

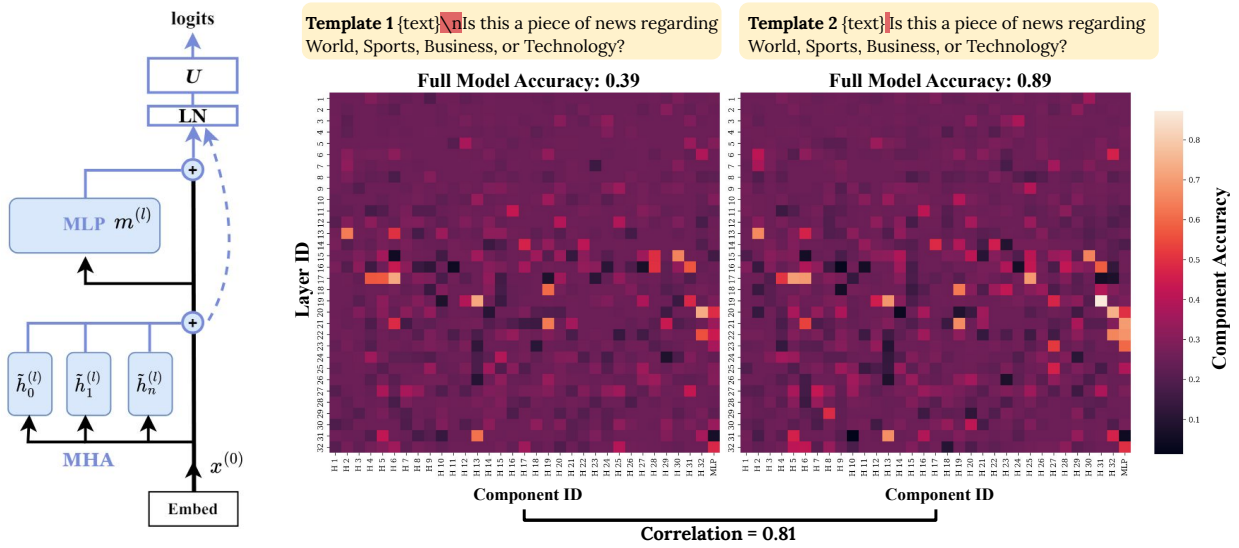


Figure 4.2: **Left:** Transformer decomposition. The components, MLPs and attention heads, are filled with blue, and the blue lines show the flow of early decoding. **Right:** We can calculate the individual accuracy of every component after decomposition. Although a pair of templates that only differ slightly yield very different accuracies (0.39 vs 0.89 on AGNews with Llama-2-7B), the accuracies of their internal components are highly correlated. The top components for Template 1 overlap with the ones for Template 2 and achieve > 0.7 accuracy despite the poor full-model accuracy.

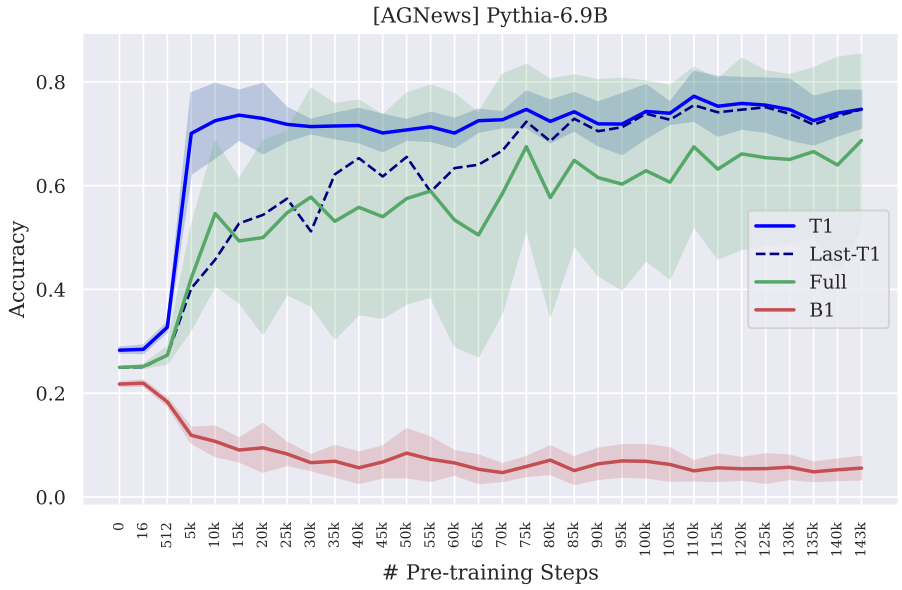


Figure 4.3: The ICL accuracy of the full model (green) fluctuates greatly during pretraining. However, good-performing components (T1) emerge in the early steps.

Chapter 5

Why Do Some Inputs Break Low-Bit LLM Quantization?

5.1 Introduction

Many large language models (LLMs; [123, 9, 124, 125]) use 16-bit precision to represent each parameter by default, requiring substantial GPU memory for inference. Quantization reduces memory usage by reducing the bit-precision of model weights or activations. Low-bit weight-only quantization [10, 126, 127, 128], which quantizes model weights to ≤ 4 bits and keeps the activations in 16 bits, greatly reduces the memory footprint while maintaining reasonable performance across various benchmarks [33].

However, Dettmers and Zettlemoyer [34] find that at 3-bits, LLMs start to show noticeable accuracy degradation. Kumar et al. [129] show that LLMs trained on more data become harder to quantize. Prior efforts focus on *outliers*, the abnormally large activations or weights that cause large errors in LLM quantization [12]. Quantization methods that combat outliers lead to overall better performance [35, 130, 36, 131]; however, we observe that some examples still suffer from large quantization errors under various methods. Why does quantization disproportionately affect certain examples? What factors are indicative of quantization errors? Which parts of the LLMs lead to large errors? We study these

underexplored questions across diverse 3- and 4-bit weight-only quantization methods.

First, we observe that the quantization errors of various pairs of methods are highly correlated (avg. $\rho = 0.82$) and that certain examples yield large errors across methods, where we define errors as the increase in negative log-likelihoods on sampled pretraining documents after quantization. A possible hypothesis is that the errors are largely predetermined by the full-precision model. Since Lin et al. [11] show that activation outliers amplify the errors in quantized weights, we examine whether the activation magnitudes of the full-precision models are indicative of the future quantization errors.

Surprisingly, we find strong *negative* correlations between quantization errors and the magnitudes of the residual stream. For instance, the correlations computed on the residuals across the last 30 layers of Llama3-70B often exceed -0.6 and reach -0.8 in the final layer. The strong correlations support our hypothesis that the full-precision model largely decides the future quantization errors. Meanwhile, it is unexpected that smaller residual activations are associated with large quantization errors. Our further analysis reveals that the RMS LayerNorm [90] tends to *reverse* the relative activation magnitudes. These findings altogether explain why certain examples have large quantization errors across methods: examples that inherently have smaller residual magnitudes will have larger post-RMSNorm magnitudes over multiple layers, which continually amplify the errors in the quantized weights and ultimately result in large quantization errors in outputs.

We further apply LLM localization techniques to study where quantization errors arise from. First, we use early exiting [132] to project the residual hidden states across layers to the output probabilities. Early exiting reveals that examples with large quantization errors rely heavily on later layers to adjust the model output distributions. However, restoring the weights of later layers to 16 bits only marginally reduces perplexity, implying that the problem lies with the accumulated errors in activations over layers. Next, we adapt cross-model activation patching [133], which restores specific activations of the quantized model to the clean counterparts from the full-precision model. We find that restoring the outputs

of the MLP gate projection [134] can substantially reduce the perplexity.

Finally, we investigate what kinds of data tend to have large quantization errors. We classify the large-error examples and find that they cover diverse topics. Because Ahia, Kreutzer, and Hooker [135] and Marchisio et al. [136] show that model compression has a disparate impact on long-tail examples, we also study the relationship between quantization errors and long-tail examples. Surprisingly, our results on both FineWeb [137] and PopQA [138] datasets suggest that quantization has a milder impact on long-tail examples and that well-represented examples are not immune to large degradation in log-likelihood and accuracy.

Our work centers on why, where, and what leads to large quantization errors, offering systematic analyses across diverse methods. We provide a mechanistic explanation of why certain examples suffer from large quantization errors and identify key model components that are critical to maintaining the perplexity. Our study could motivate future methods to reduce errors from MLP gate projections or to introduce LayerNorm scaling factors [139, 11] that consider the reversal effects in RMSNorm.[1]

5.2 Quantization Methods

This section introduces the quantization methods in our study, covering diverse, widely-used approaches for low-bit weight-only quantization.

5.2.1 GPTQ.

Frantar et al. [10] propose GPTQ, an approximate second-order method that quantizes one weight at a time while updating not-yet-quantized weights to compensate for the quantization loss. GPTQ minimizes the loss on a small calibration set of C4 [140] samples. We adopt common GPTQ settings, using activation sorting and a Hessian damping coefficient of 0.01

[1] Code: `https://github.com/terarachang/QError`

when not otherwise specified.

5.2.2 AWQ.

Lin et al. [11] observe that the activation outliers magnify the quantization errors in the corresponding weight dimensions. They propose AWQ, which smoothes the activations by introducing a scaling factor corresponding to each input dimension of weights. AWQ grid-searches the scaling factors based on the activation magnitudes.

5.2.3 NormalFloat.

Dettmers et al. [141] propose NormalFloat (NF), a 4-bit data type that is optimal for normally distributed weights based on information theory. NF employs non-uniform quantization with Gaussian quantiles and requires a lookup table. Guo et al. [142] accelerate the lookup table operation and extend NF to lower bit precision.

5.2.4 EfficientQAT.

Chen et al. [143] propose EfficientQAT (EQAT), a lightweight quantization-aware fine-tuning [144] method. EQAT applies straight-through estimator [145] for backpropagation.

5.2.5 LayerNorm in Quantization.

Kovaleva et al. [146] and Wei et al. [147] find that outliers in the LayerNorm parameters of BERT [3] cause difficulties in model compression. Given the importance of LayerNorm, all the quantization methods we discuss above leave LayerNorm unquantized. Specifically,

LLMs in Llama [7], Mistral [8], and Qwen [148] families use RMSNorm [90]:

$$\text{RMSNorm}(z) = \frac{z}{\text{RMS}(z)} \odot \gamma, \tag{5.1}$$

$$\text{RMS}(z) = \sqrt{\frac{1}{d}\sum_{i=1}^{d} z_i^2}, \tag{5.2}$$

where z is the d-dimensional input, $\odot$ denotes element-wise multiplication, and $\gamma \in \mathbb{R}^d$ is the unqauntized weights for affine transformation.

5.3 Problem Setups

5.3.1 Datasets

FineWeb. We create a dataset $\mathcal{D}$ of 10k examples sampled from the FineWeb corpus [137], consisting of deduplicated English web documents. We evaluate the negative log-likelihood (NLL) of different models on $\mathcal{D}$. Formally, given a model θ and an example $x = [x_0, x_1, \ldots, x_T]$ of T tokens, prepended with a start of sentence token x_0, we define the NLL of x as:

$$\text{NLL}(x;\theta) = \frac{1}{T}\sum_{t=1}^{T} -\log p_\theta(x_t|x_{<t}), \tag{5.3}$$

where p_θ is the probability distribution of the model θ. We only sample from long documents and truncate all the sampled documents to 512 tokens.

Quantization Errors. NLL is a common loss term for LLM pretraining, and prior quantization research uses perplexity (exponentiated average NLL) as a standard metric. Therefore, we define the quantization error on a FineWeb example x as the increase in NLL. Formally, given the full-precision LLM θ and its quantized counterpart $\tilde{\theta}$, we define the quantization error of $\tilde{\theta}$ on x as:

$$e(x;\tilde{\theta}) = \text{NLL}(x;\tilde{\theta}) - \text{NLL}(x;\theta) \tag{5.4}$$

Large-Error Set and Control Set. For fine-grained analyses, we create a large-error set $\mathcal{D}_{\text{large}} \subset \mathcal{D}$, consisting of the top-10% (1k) examples by quantization errors and a control set $\mathcal{D}_{\text{ctrl}}$ of 1k examples sampled from the bottom-50%.[2]

PopQA. We use PopQA [137] to study how quantization affects LLMs' factual knowledge of entities under different popularity levels. PopQA contains 14k questions, each supplemented with the entity popularity. Following Penedo et al. [137], we use 15-shot prompting, greedy decoding, and exact match accuracy.

5.3.2 Models and Quantization Methods

We study four LLMs: Qwen2.5-7B [149], Llama3-8B [123], Mistral-Nemo-12B,[3] and Llama3-70B. We study quantization methods in the 3–4 bit, weight-only quantization regime: GPTQ, AWQ, NF, and EQAT (see Section 5.2). We follow original implementations and detail the calibration sets, quantization configurations, and perplexity of each method in Section C.1.

5.4 Do Methods Fail on the Same Inputs?

First, we investigate whether diverse quantization methods cause large errors on the same inputs. Recall that we define the example-level quantization error $e(x^{(i)}; \tilde{\theta}) \in \mathbb{R}$ in Eq. 5.4. Given a dataset $\mathcal{D} = \{x^{(i)}\}_{i=1}^{N}$ of N examples, we define the errors of the quantized model $\tilde{\theta}$ on the dataset $\mathcal{D}$ as:

$$\mathcal{E}(\mathcal{D}; \tilde{\theta}) = [e(x^{(1)}; \tilde{\theta}), \ldots, e(x^{(N)}; \tilde{\theta})] \ \in \mathbb{R}^N,$$

Let $\tilde{\theta}_1$ and $\tilde{\theta}_2$ be the quantized models of the same base LLM θ, produced using different methods. We compute their quantization errors on the dataset, $\mathcal{E}_1 \triangleq \mathcal{E}(\mathcal{D}; \tilde{\theta}_1)$ and $\mathcal{E}_2 \triangleq$

[2] See Section C.2 for detailed dataset construction.

[3] `https://mistral.ai/news/mistral-nemo`

$\mathcal{E}(\mathcal{D}; \tilde{\theta}_2)$, respectively, and report the Pearson correlation between the errors, denoted as $\rho(\mathcal{E}_1, \mathcal{E}_2)$.

5.4.1 Strong Correlations Between Methods

Table 5.1 shows the correlations between the quantization errors of every two methods. We include the results of Qwen2.5-7B and Llama3-8B in Table C.19 in appendix. Across four LLMs, we observe high correlations, an average of $\rho(\mathcal{E}_1, \mathcal{E}_2) = 0.82$ over all 50 pairs of methods, although Qwen2.5-7B has lower correlations (avg. 0.66) compared to other LLMs.

Model	**(Method1, Method2)**	$\boldsymbol{\rho(\mathcal{E}_1, \mathcal{E}_2)}$
Mistral-12B	(AWQ4 , NF4)	0.78
	(AWQ4 , GPTQ4)	0.86
	(AWQ4 , NF3)	0.80
	(AWQ4 , GPTQ3)	0.83
	(AWQ3 , NF3)	0.90
	(AWQ3 , GPTQ3)	0.95
	(NF4 , GPTQ4)	0.82
	(NF4 , AWQ3)	0.81
	(NF4 , GPTQ3)	0.78
	(NF3 , GPTQ3)	0.89
	(GPTQ4, AWQ3)	0.90
	(GPTQ4, NF3)	0.85
Llama3-70B	(AWQ4 , NF4)	0.80
	(AWQ4 , GPTQ4)	0.96
	(AWQ4 , GPTQ3)	0.86
	(AWQ4 , EQAT3)	0.81
	(AWQ3 , EQAT3)	0.85
	(NF4 , GPTQ4)	0.81
	(NF4 , GPTQ3)	0.80
	(NF4 , AWQ3)	0.81
	(NF4 , EQAT3)	0.75
	(GPTQ4, AWQ3)	0.94
	(GPTQ4, EQAT3)	0.83
	(GPTQ3 , AWQ3)	0.96
	(GPTQ3, EQAT3)	0.83

Table 5.1: The quantization errors of different methods are strongly correlated on the FineWeb. The numbers in the method names denote 3- or 4-bit precision.

The overall strong correlation indicates that diverse methods have a similar effect on LLMs' likelihoods. We further find that the top-10% large-error examples of different methods on the same LLMs are highly overlapped, with an average Jaccard similarity of 0.51, far above the ~ 0.05 expected values by chance.

Since the configurations of the GPTQ algorithm can greatly affect the quantization outcomes [10, 150], we further examine correlations in quantization errors across different GPTQ variants. We focus on two factors: (1) the damping coefficient that improves the numerical stability of the inverse Hessian, varied over the range $[0.005, 0.01, 0.02]$, and (2) whether activation-based sorting is applied to change the weight quantization order. We find that the 3-bit Qwen2.5-7B model variants produced by different GPTQ configurations show a strong average correlation (0.93) in quantization errors on FineWeb (see Table C.20 in the appendix).

5.5 Why Do Examples Have Large Errors?

We further explore why certain examples have large quantization errors across methods. Inspired by Lin et al. [11], we first study whether the activation magnitudes from the full-precision models are predictive of the quantization errors.

5.5.1 Notations

Formally, a Transformer layer [6] in modern LLMs consists of a multi-headed attention (MHA), an MLP, and two Pre-LNs [151], LN_1 and LN_2:

$$r^{(l)} = z^{(l-1)} + \text{MHA}^{(l)}\left(\text{LN}_1^{(l)}(z^{(l-1)})\right), \tag{5.5}$$

$$z^{(l)} = r^{(l)} + \text{MLP}^{(l)}\left(\text{LN}_2^{(l)}(r^{(l)})\right), \tag{5.6}$$

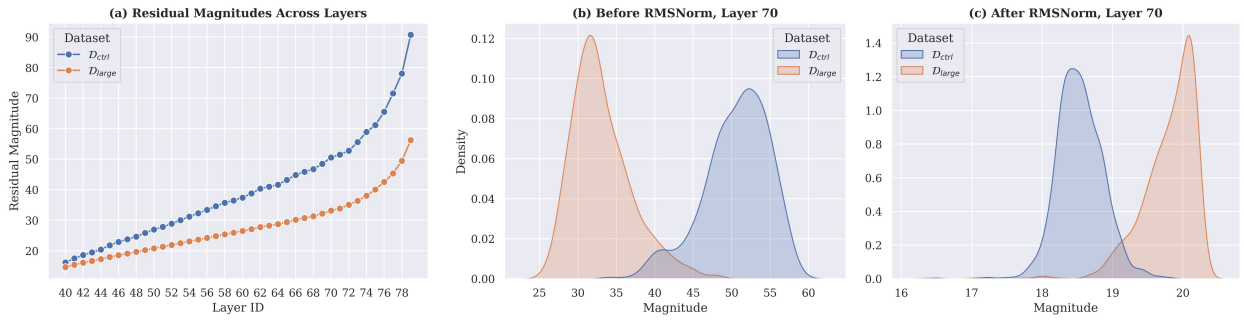


Figure 5.1: (a) The residual magnitudes of the large-error set $\mathcal{D}_{\text{large}}$ are smaller than those of $\mathcal{D}_{\text{ctrl}}$ across layers. (b) & (c) RMSNorm reverses the relative activation magnitudes. Before RMSNorm (b), the residual magnitudes of $\mathcal{D}_{\text{large}}$ are smaller than $\mathcal{D}_{\text{ctrl}}$, as shown in (a). After applying RMSNorm (c), the activation magnitudes of $\mathcal{D}_{\text{large}}$ become larger. We observe clear reversal effects in the last 10 layers of the full-precision Llama3-70B.

where l is the layer index and $z^{(0)} \in \mathbb{R}^d$ is the input to the first layer. We observe that the representations after the residual operations, $r^{(l)}$ and $z^{(l)}$, are both indicative of quantization errors. As $r^{(l)}$ shows clearer patterns, we name it as residual state and include the results of $z^{(l)}$ in appendix. Note that all the studied methods quantize the weights in MHA and MLP, but keep the weights γ in LN_1 and LN_2 in 16 bits (see Section 5.2).

Given an example x of T tokens, we quantify the magnitude of its residual states at layer l by averaging the Euclidean norm of each token representation $r_t^{(l)} \in \mathbb{R}^d$ over all T tokens:

$$\text{norm}\left(x; r^{(l)}\right) = \frac{1}{T}\sum_{t=1}^{T} \|r_t^{(l)}\|_2 \tag{5.7}$$

We name $\text{norm}\left(x; r^{(l)}\right) \in \mathbb{R}$ as the residual magnitude of example x at layer l. Given a dataset $\mathcal{D} = \{x^{(i)}\}_{i=1}^{N}$ of N examples, we compute $\text{norm}\left(x; r^{(l)}\right)$ for each example $x^{(i)}$ and denote the residual magnitudes on $\mathcal{D}$ as $\mathbb{S}(\mathcal{D}; \theta; l) \in \mathbb{R}^N$.

Finally, given an full-precision model θ with a layer index l, a quantized model $\tilde{\theta}$, and a dataset $\mathcal{D}$ of N examples, we calculate the Pearson correlation between the residual magnitudes $\mathbb{S}(\mathcal{D}; \theta; l)$ and the quantization errors $\mathcal{E}(\mathcal{D}; \tilde{\theta})$, denoted as $\rho\left(\mathbb{S}(l), \mathcal{E}\right)$. Note that $\mathcal{E}(\mathcal{D}; \tilde{\theta})$ measures the final errors of the quantized model and thus does not depend on the layer l.

5.5.2 Residual Magnitudes From Full-Precision Models Indicates Quantization Errors

We observe a strong negative correlation between the quantization error $\mathcal{E}(\mathcal{D}; \tilde{\theta})$ and the residual magnitudes $\mathbb{S}(\mathcal{D}; \theta; l)$ computed on the last few layers of the full-precision model, where $\mathcal{D}$ is the FineWeb dataset of 10k examples. For example, let $l := L$ be the last Transformer layer, we report the correlation $\rho\left(\mathbb{S}(L), \mathcal{E}\right)$ of different LLMs and quantization methods in Table 5.2. The average correlation over 3-bit methods on Qwen2.5-7B, Llama3-8B, Mistral-12B, and Llama3-70B are $-0.66, -0.76, -0.78$, and -0.80. As shown in Section C.3, the correlations become increasingly negative in upper layers. The strong negative correlations suggest that (1) the full-precision LLMs can indicate an example's future quantization error, and (2) small residual magnitudes in upper layers are associated with large quantization errors.

To further investigate the cause of large errors, we compare the large-error set $\mathcal{D}_{\text{large}}$ with the control set $\mathcal{D}_{\text{ctrl}}$ (Section 5.3.1). We compute the average residual magnitudes over examples in $\mathcal{D}_{\text{large}}$ and $\mathcal{D}_{\text{ctrl}}$, respectively, and then compare their average magnitudes across

Model	**Method**	$\boldsymbol{\rho\left(\mathbb{S}(L), \mathcal{E}\right)}$
Qwen2.5-7B	AWQ3	-0.66
	NF3	-0.58
	GPTQ3	-0.74
Llama3-8B	AWQ3	-0.76
	NF3	-0.76
	EQAT3	-0.75
Mistral-12B	AWQ3	-0.80
	NF3	-0.71
	GPTQ3	-0.83
Llama3-70B	AWQ3	-0.78
	GPTQ3	-0.81

Table 5.2: The final-layer residual magnitudes from the full-precision model have a strong negative correlation with the quantization errors of 3-bit methods.

the upper half layers of Llama3-70B. In Fig. 5.1 (a), we find that for both datasets, the average residual magnitudes increase over layers; however, the $\mathcal{D}_{\text{large}}$ (orange) consistently has smaller magnitudes than $\mathcal{D}_{\text{ctrl}}$ (blue). This implies that having a small residual magnitude in the final layer is the cumulative result of continually having smaller residual magnitudes over the upper layers. Our observation seems to contradict the previous belief that *large* activations are detrimental to weight-only quantization [11]. We investigate the cause of this discrepancy in the next section.

5.5.3 From Residuals to Errors

The reversal effect of RMSNorm. Why are smaller residual magnitudes in upper layers associated with larger quantization errors? The RMSNorm applied to the residual states before MLP (Eq. 5.6), plays a crucial role. We observe a "reversal effect", where the activations with smaller magnitudes often end up with larger magnitudes after applying RMSNorm. Formally, we denote the hidden state after RMSNorm as $h^{(l)} = \text{RMSNorm}^{(l)}(r^{(l)})$. Following Eq. 5.7, we can compute the magnitude of $h^{(l)}$, denoted as $\text{norm}\left(x; h^{(l)}\right)$. Fig. 5.1 (b) & (c) present the density of $\text{norm}\left(x; r^{(l)}\right)$ and $\text{norm}\left(x; h^{(l)}\right)$, respectively. Initially, $\mathcal{D}_{\text{large}}$ has smaller $\text{norm}\left(x; r^{(l)}\right)$ than $\mathcal{D}_{\text{ctrl}}$, but after RMSNorm, it has larger $\text{norm}\left(x; h^{(l)}\right)$. We observe the reversal effect in the upper layers of all four LLMs.

Activations amplify errors in weights. Since the hidden state after RMSNorm is the immediate input to the quantized MLP (Eq. 5.6), we hypothesize that a larger $\text{norm}\left(x; h^{(l)}\right)$ leads to amplification of errors in the quantized weights. Let $h^{(l)}$ and $\tilde{h}^{(l)}$ be the inputs to the l-th layer MLP in the full-precision and quantized models, respectively, and let $o^{(l)}$ and $\tilde{o}^{(l)}$ be the corresponding MLP outputs. We compute the mean squared error between $o^{(l)}$ and $\tilde{o}^{(l)}$ to quantify the layer-wise output error, denoted as $\text{MSE}^{(l)}$. We confirm that the input magnitudes $\text{norm}\left(x; h^{(l)}\right)$ have positive correlations (> 0.6) with $\text{MSE}^{(l)}$ in most upper layers.

Full Hypothesis. Combining our findings above, we establish a full hypothesis: certain examples inherently have smaller residual magnitudes over layers. Due to the reversal effect, they tend to have larger activation magnitudes after RMSNorms. When the model is quantized, large activations will amplify the quantization errors in the weights over multiple layers and eventually result in large quantization errors. Both the residual states and errors are cumulative over layers, which may explain why the final-layer residual magnitudes $\mathbb{S}(\mathcal{D};\theta;L)$ have the strongest correlation with the final errors $\mathcal{E}(\mathcal{D};\tilde{\theta})$.

Why does $\mathcal{D}_{\text{large}}$ have smaller residual magnitudes? We find that examples in the large-error set have fewer/weaker outliers in the residual states $r^{(l)}$. Following Bondarenko, Nagel, and Blankevoort [152] and Gong et al. [153], we use the Kurtosis value to quantify the severity of outliers. We measure the Kurtosis value $\kappa^{(l)}$ of the l-th layer residual states $r^{(l)}$, where $\kappa^{(l)} = \mathbb{E}\left[\left(\frac{r^{(l)}-\mu}{\sigma}\right)^4\right]$. A large $\kappa^{(l)}$ indicates more outliers [154]. Fig. C.8 in appendix shows that the Kurtosis values of $\mathcal{D}_{\text{large}}$ (orange) are smaller than those of $\mathcal{D}_{\text{ctrl}}$ (blue) across the upper layers of LLMs, suggesting that $\mathcal{D}_{\text{large}}$ has fewer outliers in the residual states. Because outliers dominate the activation magnitudes, having fewer outliers leads to smaller residual magnitudes.

How does RMSNorm reverse the relative magnitudes? We observe that the RMSNorm weights $\gamma \in \mathbb{R}^d$ suppress many outliers in the residual states by assigning smaller weight values to the corresponding dimensions. Specifically, we rank the entries in γ by

Notation		Definition
$r^{(l)}$	$\mathbb{R}^d$	layer l residual states (Eq. 5.5)
$h^{(l)}$	$\mathbb{R}^d$	$\text{RMSNorm}^{(l)}(r^{(l)})$
$o^{(l)}$	$\mathbb{R}^d$	$\text{MLP}^{(l)}(h^{(l)})$
γ	$\mathbb{R}^d$	RMSNorm weights (Eq. 5.1)
$\text{norm}\left(x; r^{(l)}\right)$	$\mathbb{R}$	layer l residual magnitude of x
$\mathbb{S}(\mathcal{D};\theta;l)$	$\mathbb{R}^N$	layer l residual magnitudes of $\mathcal{D}$

Table 5.3: Notation summary.

their absolute values and find that in the last few layers, the largest outliers[4] often have the smallest RMSNorm weights across all four LLMs. We include the statistics and discussion on other outlier dimensions in Section C.4. We summarize the mechanism behind the reversal effect: a residual state $r^{(l)}$ with fewer outliers has a smaller magnitude, but RMSNorm normalizes the magnitude and downweights outliers with γ, yielding a hidden state $h^{(l)}$ with a relatively larger magnitude. Since $\mathcal{D}_{\text{large}}$ examples have smaller kurtosis values but larger norm $\left(x; h^{(l)}\right)$, the large quantization errors are driven by the overall larger activations in $h^{(l)}$.

5.6 Where Do Errors Arise From?

We study which parts of LLMs lead to large quantization errors with localization techniques: early exiting [132, 47] and activation patching [24].

5.6.1 Early Exiting

nostalgebraist [132] first introduces logit lens, an early exiting technique that projects hidden representations into the vocabulary space using the LLMs' output embeddings. We use early exiting to measure the degree to which a residual state is ready for decoding. Formally, let $r^{(l)}$ be the residual state at layer l and $\phi(\cdot)$ be the final RMSNorm operation applied before the output embedding matrix U. We define the probability distribution from decoding $r^{(l)}$:

$$p^{(l)} = \text{Softmax}\left(U \cdot \phi(r^{(l)})\right)$$

Let $p_\theta^{(l)}$ and $p_{\tilde{\theta}}^{(l)}$ be the early-exiting probabilities from the full-precision and quantized models. We use $p_\theta^{(l)}$ and $p_{\tilde{\theta}}^{(l)}$ to compute the average negative log-likelihood (NLL) over dataset $\mathcal{D}$ at each layer, denoted $\text{nll}_\theta^{(l)}(\mathcal{D})$ and $\text{nll}_{\tilde{\theta}}^{(l)}(\mathcal{D})$, respecitvely. A $\text{nll}_\theta^{(l)}(\mathcal{D})$ value near the fi-

[4]The "most-activated" dimension in $r^{(l)}$ that has the highest average absolute activation over the FineWeb dataset.

Model	Method	16-bit	3-bit	Patch h_{down}	Patch h_{gate}	Patch h_{up}	Patch h_{attn}
Qwen2.5-7B	AWQ3	9.42	12.26 (+2.84)	9.73 (+0.31)	10.20 (+0.78)	12.64 (+3.22)	11.12 (+1.70)
	NF3		12.53 (+3.11)	9.66 (+0.24)	10.61 (+1.19)	12.97 (+3.55)	11.35 (+1.93)
Llama3-8B	AWQ3	5.62	10.68 (+5.06)	5.95 (+0.33)	6.42 (+0.80)	9.97 (+4.35)	9.27 (+3.65)
	NF3		11.02 (+5.40)	5.92 (+0.30)	6.43 (+0.81)	10.48 (+4.86)	9.49 (+3.87)
Mistral-12B	AWQ3	5.60	8.67 (+3.07)	6.05 (+0.45)	6.13 (+0.53)	12.49 (+6.89)	7.61 (+2.01)
	NF3		10.14 (+4.54)	5.92 (+0.32)	7.12 (+1.52)	13.75 (+8.15)	8.71 (+3.11)
Llama3-70B	AWQ3	2.66	5.25 (+2.59)	2.73 (+0.07)	2.94 (+0.28)	3.24 (+0.58)	4.73 (+2.07)

Table 5.4: The perplexity (↓) on $\mathcal{D}_{\text{large}}$ before/after patching activations from 16-bit full-precision models into 3-bit models, quantized with AWQ3 and NF3 methods, respectively. Both patching the entire MLP outputs (h_{down}) and patching only the MLP gate outputs (h_{gate}) can substantially close the gap between the 3-bit and 16-bit models.

nal model NLL means that the residual state at layer l contains sufficient information for decoding; otherwise, further layers are needed to refine the output probability.

We investigate two key questions: (1) Can we observe patterns from the early exiting dynamics that distinguish $\mathcal{D}_{\text{large}}$ and $\mathcal{D}_{\text{ctrl}}$? (2) At which layer do $\text{nll}_{\theta}^{(l)}(\mathcal{D})$ and $\text{nll}_{\tilde{\theta}}^{(l)}(\mathcal{D})$ begin to diverge? To answer these questions, we perform four runs of early exiting, one for each combination of model type (full-precision and quantized) and dataset split ($\mathcal{D}_{\text{large}}$ and $\mathcal{D}_{\text{ctrl}}$). We then compute $\text{nll}_{\theta}^{(l)}(\mathcal{D}_{\text{large}})$, $\text{nll}_{\tilde{\theta}}^{(l)}(\mathcal{D}_{\text{large}})$, $\text{nll}_{\theta}^{(l)}(\mathcal{D}_{\text{ctrl}})$, and $\text{nll}_{\tilde{\theta}}^{(l)}(\mathcal{D}_{\text{ctrl}})$ across the upper half layers.

Fig. 5.2 compares how the four NLL values evolve across the upper layers of Mistral-12B, where the quantized model $\tilde{\theta}$ is produced by AWQ3. We observe a clear distinction between $\mathcal{D}_{\text{large}}$ and $\mathcal{D}_{\text{ctrl}}$ on the full-precision model: $\mathcal{D}_{\text{large}}$ (purple) has higher NLLs than $\mathcal{D}_{\text{ctrl}}$ (green) until layer 36, around which its NLLs begin to decrease sharply. This observation supports our claim in Section 5.5 that the properties of the full-precision model reflect the future quantization errors. Moreover, the sharp NLL decrease in later layers suggests that $\mathcal{D}_{\text{large}}$ heavily relies on these layers to make accurate predictions. This reliance requires the quantized model to keep the residual states precise through to the later layers, which is a challenging task because quantization errors propagate over layers. Otherwise, the model cannot decode the noisy representations, resulting in large quantization errors. By

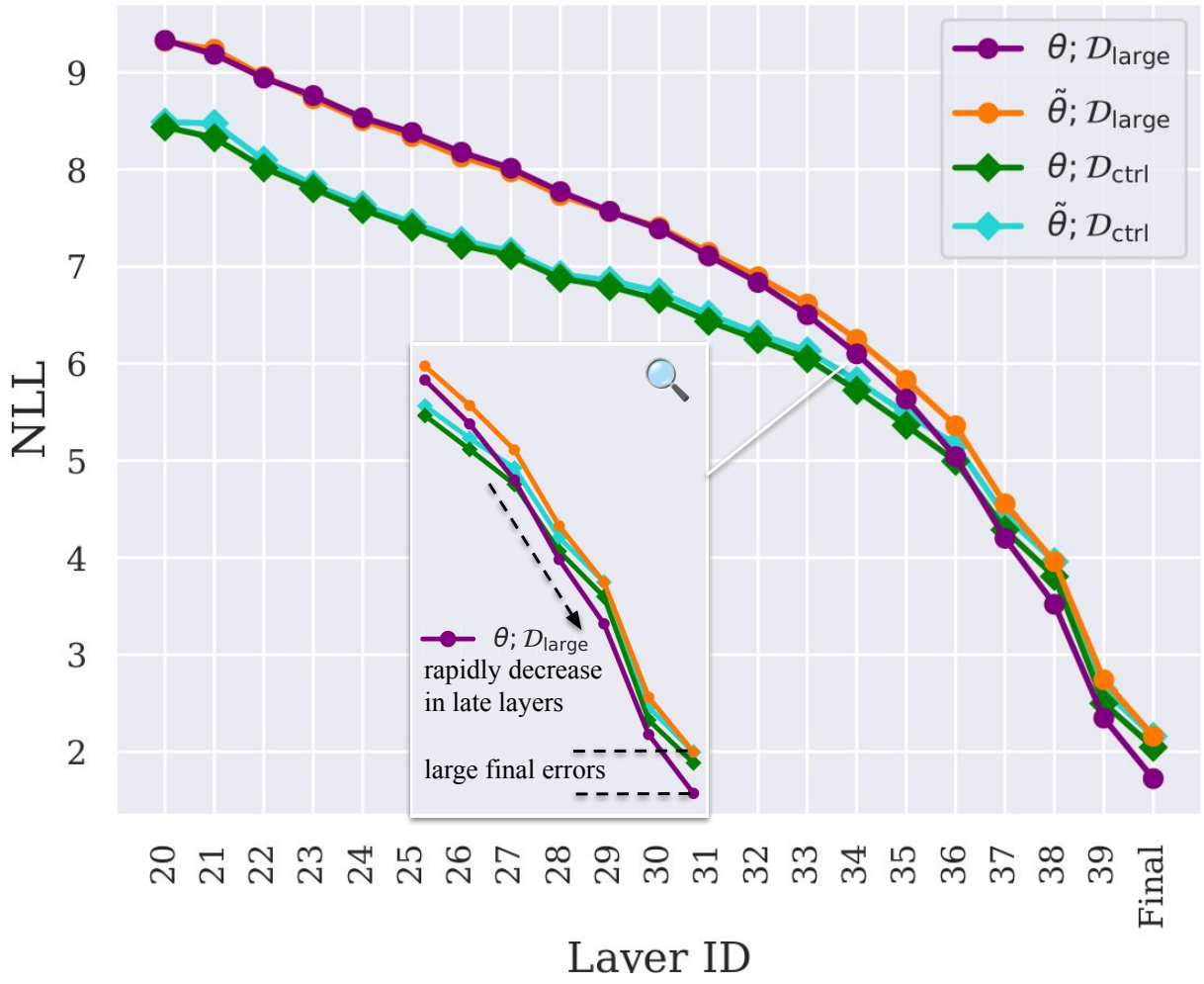


Figure 5.2: Early exiting across layers shows that the NLLs of $\mathcal{D}_{\text{large}}$ (purple and orange) dramatically decrease in the late layers, suggesting that $\mathcal{D}_{\text{large}}$ heavily relies on the late layers to adjust the output probabilities. Meanwhile, the quantized model (orange) does not show obvious deviation from the full-precision one (purple) until layer 33.

comparison, $\mathcal{D}_{\text{ctrl}}$ relies less on the later layers, and the quantized model (cyan) shows less deviation from the full-precision one (green).

An alternative hypothesis is that the precision of weights in the later layers is crucial to $\mathcal{D}_{\text{large}}$. To test this hypothesis, we keep all the weights above layer 32 in full-precision, because Fig. 5.2 shows that on $\mathcal{D}_{\text{large}}$, the quantized model (orange) starts to diverge from the full-precision model (purple) at layer 33. This experiment restores 7 layers (18%) from 3 bits to 16 bits. However, the perplexity on $\mathcal{D}_{\text{large}}$ only slightly decreases from 8.67 to 8.05, versus 5.60 for the full-precision model. We show similar results on other LLMs in

Section C.5.

In summary, our early exiting experiment suggests that various methods struggle with $\mathcal{D}_{\text{large}}$ because it requires the quantized models to maintain precise residual states up to the late layers. We also find that leaving late layers unquantized has little effect on NLLs, implying that the cumulative error in residual states, not the precision of upper-layer weights, is the primary cause for large errors.

5.6.2 Cross-Model Activation Patching

Prakash et al. [133] introduce cross-model activation-patching (CMAP) to identify how fine-tuned models improve over the pretrained ones. We adapt CMAP to track where the quantized models differ from the full-precision ones, aiming to identify the sources of quantization errors. For a given example from $\mathcal{D}_{\text{large}}$, we first run a forward pass on the full-precision model and cache its activations. We then run the same example through the quantized model, but replace the activations of specific modules with those from the full-precision model. We patch the outputs of MHA, denoted h_{attn}, and the outputs of different modules in MLP. Specifically, all LLMs we study implement MLP with Gated Linear Units [155]:

$$
\begin{aligned}
h_{\text{gate}} &= \sigma(W_{\text{gate}}\, h), \\
h_{\text{up}} &= W_{\text{up}}\, h, \\
h_{\text{down}} &= W_{\text{down}}\, (h_{\text{gate}} \odot h_{\text{up}}),
\end{aligned}
$$

where h is the post-RMSNorm input to MLP, W_{gate}, W_{up}, and W_{down} are the MLP weights, h_{down} is the output of MLP, and $\sigma(\cdot)$ is the activation function. To trace quantization errors, we perform four runs of CMAP, corresponding to patching h_{gate}, h_{up}, h_{down}, and h_{attn}, in the upper half layers of the models. Patching h_{down} is a much stronger intervention than patching either h_{up} or h_{down}, as it overrides the final outputs of MLPs.

We apply AWQ3 and NF3 methods and report the perplexity on $\mathcal{D}_{\text{large}}$ in Table 5.4.

First, we find that patching the outputs of MLP h_{down} greatly narrows the perplexity gaps between the 3-bit and 16-bit full-precision models, bringing the gaps below 0.5 across all LLMs. In contrast, patching the outputs of MHA h_{attn} only leads to marginal effects, suggesting that the quantization errors from MLPs are dominating the results. We further find that inside MLPs, patching h_{up} alone is ineffective, but patching h_{gate} results in substantial perplexity reduction, approaching the effect of patching the entire MLP outputs h_{down}. These results highlight the importance of having precise outputs from MLP gates, which control the information passed on to subsequent layers.

We conduct two additional experiments to understand the source of quantization errors in MLP gates: (1) only patching the inputs to W_{gate}, and (2) restoring all W_{gate} weights to 16-bit precision. We find that patching the gates' inputs (1) performs nearly as well as patching the outputs, h_{gate}, while restoring the weights (2) is ineffective. For instance, on Llama-3-8B quantized with NF3, (1) achieves a perplexity of 6.53 compared to 10.45 for (2). These results further support our conclusion in Section 5.6.1 that noisy activations are the primary source of large quantization errors.

5.7 What Data Cause Large Errors?

Understanding what kinds of data lead to large quantization errors is crucial for future improvement. This section studies whether examples in $\mathcal{D}_{\text{large}}$ cluster in certain domains or are unusual, long-tail data.

5.7.1 Categorizing Examples in $\mathcal{D}_{\text{large}}$

To better understand the distribution of $\mathcal{D}_{\text{large}}$, we apply the topic classifier from Wettig et al. [156], which categorizes web pre-training data into 24 topics.[5] We find that the $\mathcal{D}_{\text{large}}$ sets of both Mistral-12B and Llama3-70B cover all the 24 topics, and we plot the topic distributions

[5] `https://huggingface.co/WebOrganizer/TopicClassifier-NoURL`

in Fig. 5.3. For both models, `Entertainment`, `Finance & Business`, `Politics`, and `Health` are among the top five topics, although none account for more than 12% of the dataset. We also manually inspect $\mathcal{D}_{\text{large}}$ examples and do not find them unusual.

5.7.2 Quantization Errors vs. Long-Tail Data

Ogueji et al. [157] and Marchisio et al. [136] show that model compression has a disparate effect for long-tail data; thus, we explore the relationship between quantization errors and data long-tailness.

FineWeb. We use the data likelihood assigned by a strong language model, Llama3-70B, to estimate the long-tail characteristics of the examples. We call the FineWeb examples that have small log-likelihoods (large NLLs) on the full-precision Llama3-70B as long-tail examples. Fig. 5.4a shows that long-tail examples (x-axis $\geq$ 4) have small quantization errors under both AWQ3 (avg. 0.14) and GPTQ3 (avg. 0.30) methods. On the other hand, examples that suffer from large errors have widespread NLLs before quantization, and many "head data" (small NLLs; x-axis $\leq$ 1) have large quantization errors (avg. 0.67 for AWQ3 and 1.00 for GPTQ3). We observe the same findings on the other LLMs (see Fig. C.18 in appendix).

PopQA. We investigate how quantization affects LLMs' factual knowledge on entities with different popularity levels. We partition the PopQA dataset into buckets based on the examples' $\log_{10}$(popularity) and calculate the accuracies within each bucket. Fig. 5.4b presents the accuracy degradation after quantization under different buckets, where we apply NF3 (blue) and AWQ3 (orange) to quantize Mistral-12B into 3 bits. We find that when the examples have the lowest and highest popularities (the leftmost and rightmost buckets, respectively), the 3-bit models show the lowest degradation. In contrast, examples in the middle buckets (log popularity between 2.8 and 4.8) are most affected by quantization, with accuracies dropped by $>$ 10%. Note that only 6% of examples have log popularity greater

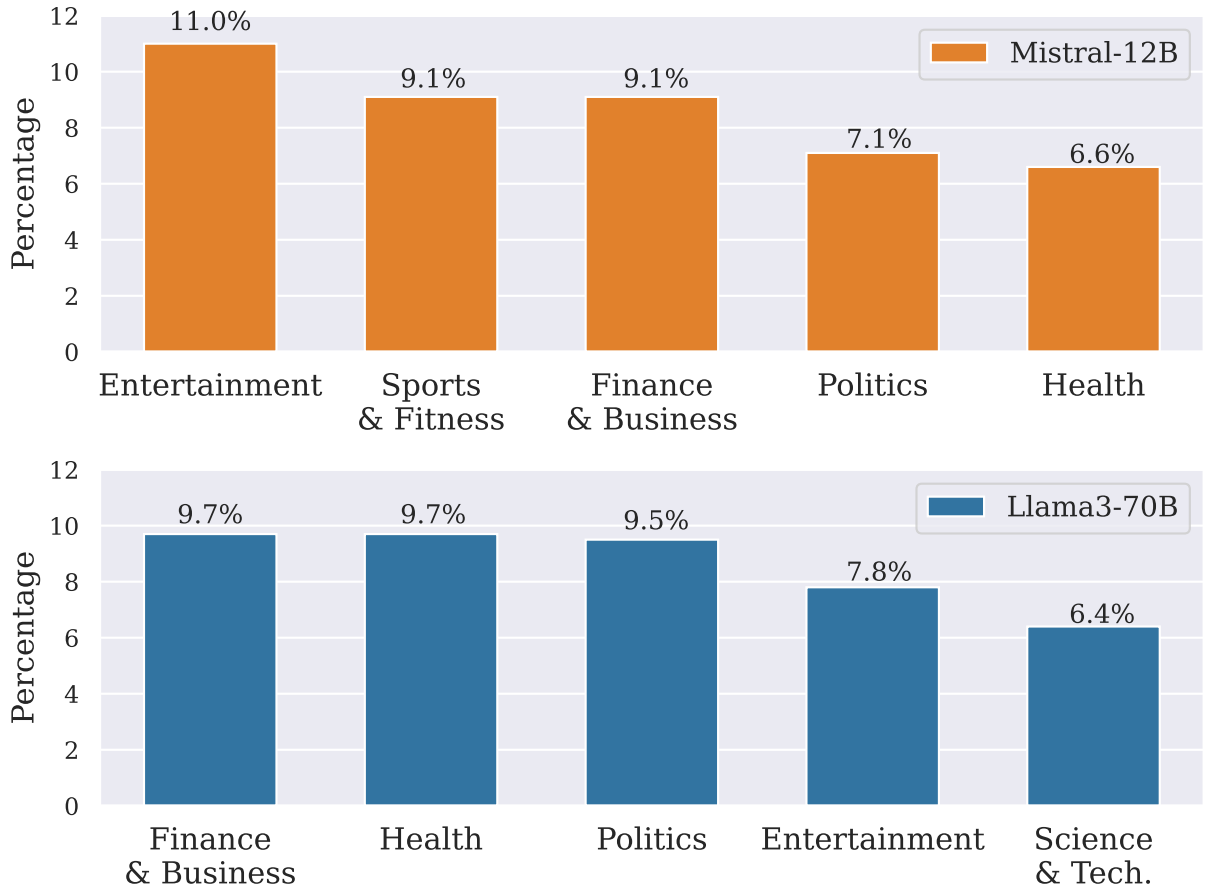


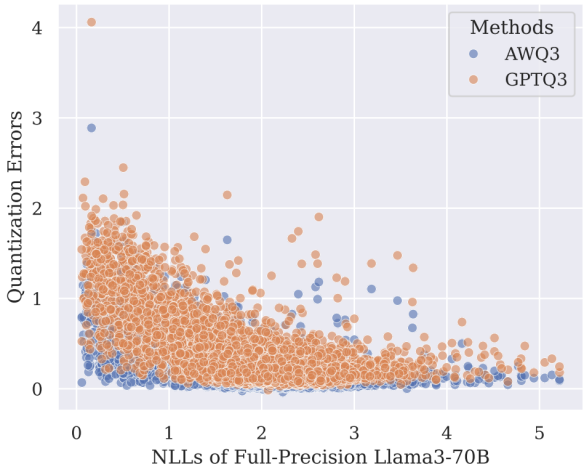
Figure 5.3: Top five topics of $\mathcal{D}_{\text{large}}$ for Mistral-12B (up; orange) and Llama3-70B (down; blue), respectively.

than 4.8, meaning that quantization can severely degrade LLMs' knowledge on entities that have intermediate to high popularity. We include results from other LLMs in Fig. C.19, showing that low popularity examples usually have milder degradation.

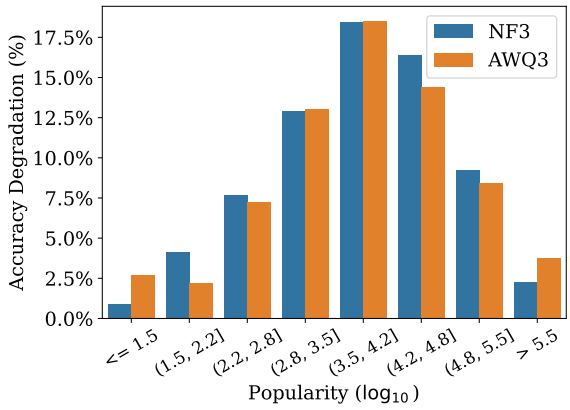
Summary. We find that long-tail examples are relatively less affected by quantization on both FineWeb and PopQA. Meanwhile, well-represented data are not immune to large errors.

5.8 Conclusion

We thoroughly study why certain examples are disproportionately affected by 3–4 bit, weight-only quantization methods. We reveal the distinct nature of large-error examples before



(a) Long-tail samples with large NLLs on the full-precision Llama3-70B have small quantization errors. Each dot represents a FineWeb example. We quantize Llama3-70B with AWQ3 and GPTQ3 methods and compute their quantization errors with Eq. 5.4.



(b) The accuracy degradation of PopQA under different levels of popularity after 3-bit quantization. Samples with log popularity between 2.8 and 4.8 show the greatest degradation.

Figure 5.4: Relationship between NLLs and quantization errors (left), and accuracy degradation on PopQA under different popularity levels after 3-bit quantization (right).

quantization, showing that these examples have smaller residual magnitudes and rely on upper layers to refine output probabilities. Our patching and weight recovery experiments further highlight the importance of having precise activations at MLP gates, and reject naive mixed-precision approaches. Our kurtosis value analysis finds that large-error examples actually have fewer outliers in residual states across upper layers. We further discover the reversal effects of RMSNorm, showing that large quantization errors may stem from overall larger activations across dimensions, rather than from a few outlier dimensions. Finally, we explore the relationship between quantization errors and long-tail data, showing that long-tail examples tend to have less degradation after quantization than other examples.

Overall, our work sheds light on why and where large quantization errors occur. Our findings may inspire quantization error prediction frameworks and encourage future post-training methods or quantization-friendly architectures that specifically address MLP gates and RMSNorm; for instance, by avoiding the Gated Linear Units or redesigning the LayerNorm [158].

Chapter 6

Value-Aware Stochastic KV Cache Eviction for Reasoning Models

6.1 Introduction

Reasoning models [38, 39, 40] leverage test-time compute [41] to improve accuracy by generating extended chains of thought before producing a final answer. While this approach yields high accuracy on complex tasks, it introduces an efficiency bottleneck at test time because the reasoning models tend to generate unnecessarily long outputs. For example, Chen et al. [159] show that models may overthink a simple arithmetic question, using more than 900 tokens to answer "2 + 3 = ?". As a consequence, an auto-regressive large language model that stores the key–value representations of every past token at each decoding step incurs substantial memory and computational overhead as the sequence length grows.

Sparse attention [160, 161, 162, 163] addresses this bottleneck by attending to only a subset of previous tokens; these methods fall into two broad categories. KV cache *selection* methods [164, 165, 166, 167] keep the full KV cache in memory but activate only a sparse subset of KV pairs at each decode step. These methods reduce compute and memory movement, but their memory footprints scale linearly with sequence length. In contrast, KV cache *eviction* methods [13, 42, 168, 169] permanently discard low-importance KV pairs once

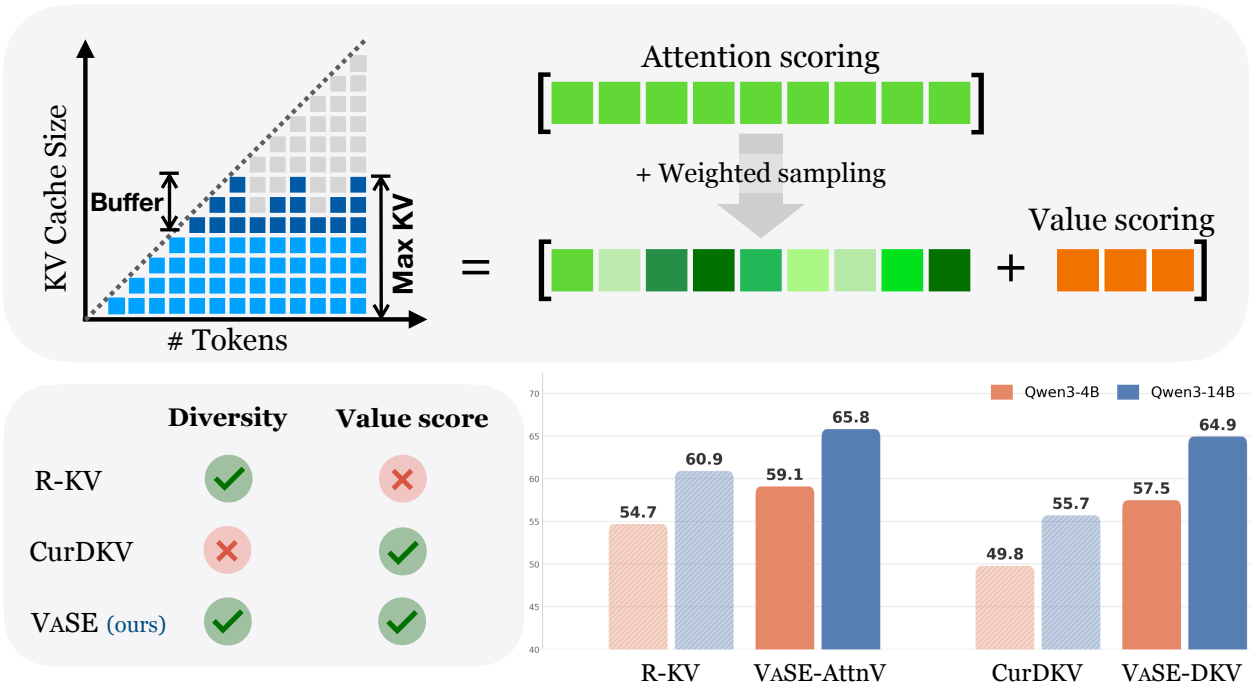


Figure 6.1: *Top*: VASE is a KV Cache eviction method that combines stochastic sampling with value-state magnitude scoring to retain diverse and important KV pairs under a fixed KV cache budget. *Bottom*: By integrating both stochasticity and value awareness, VASE outperforms baseline methods that use either signal alone, improving average pass@1 accuracy across six reasoning benchmarks.

the cache reaches a fixed budget. While eviction methods [15, 170] suffer larger accuracy degradation on reasoning tasks compared to their selection-based counterparts, they achieve static memory footprint and better throughput.

In this chapter, we reveal two key findings to improve KV cache eviction methods. First, the magnitudes of value states exhibit strongly skewed distributions, with a small fraction of tokens having abnormally large norms (see Figure 6.2). Those large-magnitude value states are crucial for maintaining the accuracy: evicting them causes the models to enter repetitive loops, where they endlessly re-examine the context or generate nonsensical outputs without reaching an answer (see Figure D.24 for an example). On the other hand, we find that introducing stochasticity during KV cache eviction effectively improves reasoning task accuracies. We attribute this to improved diversity of the retained KV pairs, which together yield a more representative coverage of the full context. Our case study on GSM8K

(Section 6.4.1) demonstrates that incorporating value scoring boosts Qwen3-4B's accuracy by as much as 16.2%, while introducing stochasticity yields an additional 4.7% improvement.

Building on these findings, we propose **V**alue-**a**ware **S**tochastic KV Cache **E**viction (VASE), a training-free KV cache eviction recipe that prioritizes large-magnitude value states and introduces stochasticity during eviction. We target the decoding step of reasoning models, by applying VASE to Qwen3 models [171] and evaluating on six tasks spanning math, code generation, and science question answering. On average, VASE outperforms the previously SOTA eviction method R-KV [170] by 4.4% on Qwen3-4B and 4.9% on Qwen3-14B. When combined with CurDKV [172], stochastic sampling further boosts its average accuracy by 7.7% and 9.2% on the two models, respectively. VASE even slightly surpasses the average accuracy of SeerAttention-R [167], a strong selection method whose KV cache memory grows linearly with the sequence length, on both models while using static memory. In addition, we benchmark the actual throughput and peak memory of different methods, showing that VASE achieves higher throughput and a lower memory footprint than the R-KV eviction baseline across token budgets. Our results match closely with the theoretical memory compression ratio.

We further connect our findings to KV cache quantization by showing that large-Range value states also incur disproportionately large reconstruction errors under per-token quantization (Section 6.4.3). Together, these findings suggest that value-state magnitude and stochasticity are fundamental axes of KV cache management, with implications that extend beyond eviction to efficient reasoning more broadly.

6.2 Problem Setup

We formalize the decode-phase KV cache eviction setting by drawing the distinction between selection-based and eviction-based methods, and introduce several eviction baselines.

Selection-based method. Selection-based sparse attention methods retain the full KV cache but activate only a sparse subset of KV pairs at each decoding step. SeerAttention-R [167] is a representative method, which uses a trained attention gate to predict attention sparsity patterns and selectively activates different KV cache blocks during inference, reducing computational complexity and memory bandwidth. Compared to eviction-based methods, SeerAttention-R retains all KV pairs and the memory footprint scales linearly as $\mathcal{O}(T)$ with sequence length T.

Eviction-based method. KV cache eviction methods permanently discard low-importance key–value pairs once the cache size reaches a predefined budget, reducing memory cost at the risk of irrecoverable information loss. We specifically focus on eviction methods for the decode phase of autoregressive reasoning models, because models may generate over 10,000 tokens for a math question that has fewer than 200 prompt tokens (see Table D.24 for statistics).

Formally, let N be the total KV cache budget, and d be the dimension of keys and values. As the KV cache budget fills up at the token index t, we have N KV pairs $(\boldsymbol{k}_i, \boldsymbol{v}_i)$ for $i = 1, \ldots, N$, where $\boldsymbol{k}_i \in \mathbb{R}^d, \boldsymbol{v}_i \in \mathbb{R}^d$. Given the current query $\boldsymbol{q}_t \in \mathbb{R}^d$, the attention head computes its output as a weighted sum over the N cached values,

$$\boldsymbol{o}_t \;=\; \sum_{i \le N} \alpha_i^{(t)}\, \boldsymbol{v}_i, \qquad \alpha_i^{(t)} \;=\; \frac{\exp\!\Big(\boldsymbol{q}_t^\top \boldsymbol{k}_i / \sqrt{d}\Big)}{\sum_{j \le N} \exp\!\Big(\boldsymbol{q}_t^\top \boldsymbol{k}_j / \sqrt{d}\Big)}, \tag{6.1}$$

where $\alpha_i^{(t)}$ is the attention weight assigned to token i, obtained by applying softmax to the scaled dot products between the current query and all cached keys [6].

Periodic eviction with budget K and buffer B. We adopt the periodic-eviction framework proposed by Cai et al. [170] and Song et al. [173], which targets decode-phase compression. The framework consists of a persistent budget of K tokens and a buffer of size $B \ll K$ holding the most recent tokens, capping the total cache size at $N = K + B$ tokens.

Since each decoding step adds one KV pair to the cache, the buffer fills every B steps and triggers an *eviction step*: the eviction operator chooses B pairs to discard, clears the buffer, and restores the cache to size K (see the upper-left panel in Figure 6.1). Concretely, the most recent B KV pairs in the buffer are protected from eviction. A *scoring function* assigns a scalar importance score to each of the remaining K candidates; the B candidates with the lowest importance scores will be discarded. In this section, we describe the scoring functions of three representative eviction methods, SnapKV [15], R-KV [170], and CurDKV[172]. We apply the periodic-eviction framework to all eviction methods with a shared buffer size B.

SnapKV. Li et al. [15] rank candidate pairs by their average attention score over a window of recent queries, where we set the window size to be the same as the buffer size B.[1]

$$\bar{\alpha}_i \coloneqq \frac{1}{B} \sum_{\tau=t-B+1}^{t} \alpha_i^{(\tau)}, \tag{6.2}$$

where i is the KV pair index, t is the eviction timestep, and $\alpha_i^{(\tau)}$ is the attention score defined in Eq. 6.1. SnapKV uses keys for the attention computation, but does not consider values in the scoring function.

R-KV. Cai et al. [170] specifically target redundant tokens in reasoning models. They augment the SnapKV score $\bar{\alpha}_i$ with a redundancy penalty r_i to promote diversity among retained KV pairs:

$$s_i^{\mathrm{RKV}} \coloneqq \lambda \cdot \bar{\alpha}_i - (1-\lambda) \cdot r_i, \qquad \text{where} \quad r_i = \mathrm{Softmax}\left(\frac{1}{K} \sum_{j \neq i} \mathrm{CosSim}(\boldsymbol{k}_i, \boldsymbol{k}_j) \right), \tag{6.3}$$

and $\lambda \in [0,1]$ is a hyperparameter. A high r_i indicates that token i has a high cosine similarity to many cached tokens; subtracting it discourages choosing redundant pairs.

[1] Following Li et al. [15], we apply pooling to $\bar{\alpha}_i$ to smooth the attention scores. Following the SnapKV variant for the decode phase [170], we cache the most recent queries and remove the causal mask in attention computation.

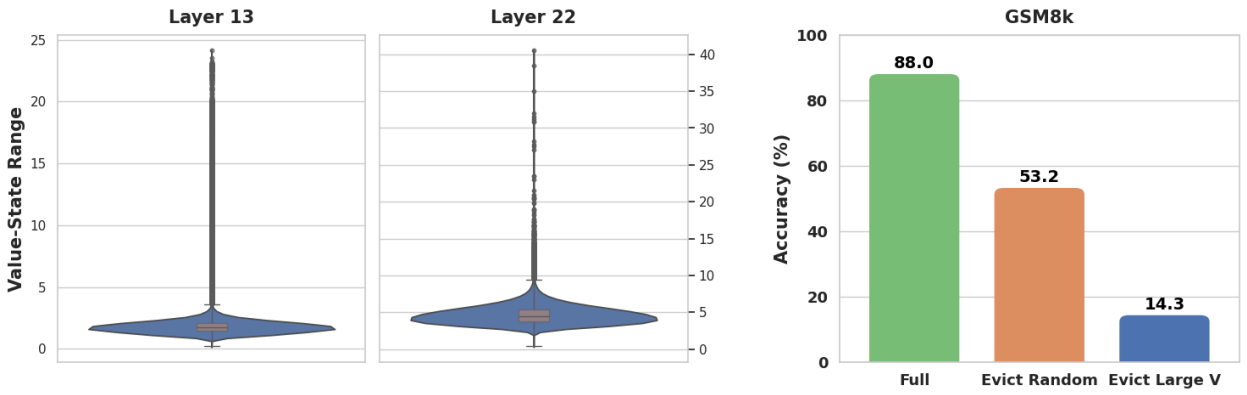


Figure 6.2: *Left*: Range distribution of the value states. The violin plots show the presence of extreme magnitude outliers at different layers. *Right*: Evicting the large magnitude outliers causes accuracy to collapse to 14.3%, greatly underperforming a random eviction baseline at the same budget, suggesting that these large-magnitude value states are crucial to model accuracy.

CurDKV. Sengupta, Chaudhary, and Chakraborty [172] observe the limitations of scoring KV pairs with attention alone, which ignores value states. They compute leverage scores for keys and values respectively via approximate CUR matrix decomposition and combine them into a single importance score. Originally designed for the prefill stage, we extend CurDKV for the decode phase with the periodic eviction framework. Specifically, CurDKV draws a Gaussian projection $\boldsymbol{G} \in \mathbb{R}^{d\times r}$ at the start of generation, with each entry $G_{ab} \overset{\text{iid}}{\sim} \mathcal{N}(0, 1/r)$, and ranks candidates by the product of key and value scores:

$$s_i^{\text{CUR}} := \ell_i^{(K)}(\boldsymbol{G}) \cdot \ell_i^{(V)}(\boldsymbol{G}), \qquad \text{where} \quad \ell_i^{(K)}(\boldsymbol{G}) := \|\boldsymbol{G}^\top \boldsymbol{k}_i\|_2^2, \quad \ell_i^{(V)}(\boldsymbol{G}) := \|\boldsymbol{G}^\top \boldsymbol{v}_i\|_2^2. \tag{6.4}$$

CurDKV considers the values for scoring, but it does not promote diversity among KV pairs.

6.3 Methodology

We hypothesize that incorporating value-awareness and diversity into KV cache scoring is essential for preserving model performance in eviction methods. Motivated by the scoring function design of CurDKV, which favors value states with larger magnitudes, we first isolate

the effect of value state magnitude on eviction performance. Based on these findings, we propose a framework that prioritizes large-magnitude values within the scoring function and introduces stochasticity to promote diversity.

6.3.1 Significance of Value States with Large Magnitudes

To understand the importance of value states, we first analyze the distribution of their magnitudes. We extract the full KV cache from Qwen3-4B while generating responses to GSM8K examples [174]. We define the magnitude[2] of a cached value vector $\boldsymbol{v}$ as:

$$\text{Range}(\boldsymbol{v}) \;:=\; \max_{j\in[d]} v_j \;-\; \min_{j\in[d]} v_j \tag{6.5}$$

where v_j is the j-th entry of $\boldsymbol{v}$, and $\text{Range}(\boldsymbol{v})$ is chosen over $L_2(\boldsymbol{v})$ for its connection to quantization (see Section 6.4.3). Figure 6.2 shows that the distribution of $\text{Range}(\boldsymbol{v})$ has large outliers over layers (see Section D.1 for more layers). We believe that these large-Range outliers are critical to the model accuracy. Because attention output is computed as a weighted summation of values (Eq. 6.1), those values with larger magnitudes have a disproportionate influence on the output.

To validate this, we purposefully evict the B largest outliers ranked by $\text{Range}(\boldsymbol{v}_i)$. This reduces GSM8K accuracy to just 14.3%, under-performing random eviction by 38.9% at the same 512-token budget (Figure 6.2; *Right*). Further inspection of model outputs reveals that the model tends to enter nonsensical loops and never reach the correct answers (see Figure D.24). Prior work [175, 176] observes that low-magnitude values co-occur with attention sinks [14], which are crucial to eviction methods. In this chapter, we demonstrate that large-magnitude values are equally important and should be up-weighted within the scoring function (Section 6.4.1).

[2]We experiment with three variants, L_2 norm, Range, and variance, to capture the magnitude and variety of value vectors in Section D.1. We find that these variants are positively correlated and lead to comparable accuracies.

Method	Family	Key score	Value score	Diversity
SeerAttention-R [167]	Select	Attn Gate	–	–
SnapKV [15]	Evict	$\bar{\alpha}_i$	–	–
R-KV [170]	Evict	$\lambda \cdot \bar{\alpha}_i - (1-\lambda) \cdot r_i$	–	redundancy r_i
CurDKV [172]	Evict	$\ell_i^{(\boldsymbol{K})} = \|(\boldsymbol{K}\boldsymbol{G})_i\|_2^2$	$\ell_i^{(\boldsymbol{V})} = \|(\boldsymbol{V}\boldsymbol{G})_i\|_2^2$	–
VaSE-AttnV (ours)	Evict	$\bar{\alpha}_i$	Range($\boldsymbol{v}_i$)	sample $\bar{\alpha}_i$
VaSE-DKV (ours)	Evict	$\ell_{i,t}^{(\boldsymbol{K})} = \|(\boldsymbol{K}\boldsymbol{G}_t)_i\|_2^2$	$\ell_{i,t}^{(\boldsymbol{V})} = \|(\boldsymbol{V}\boldsymbol{G}_t)_i\|_2^2$	sample $\boldsymbol{G}_t$

Table 6.1: Design space of prior methods. Among eviction methods, prior work covers at most two of the three axes (key-based scoring, value-based scoring, stochastic selection); VaSE is the first to combine all three, instantiated as VaSE-AttnV and VaSE-DKV.

6.3.2 VaSE: Value-Aware Stochastic Eviction

We now introduce the VaSE framework. VaSE (Value-aware Stochastic Eviction) is designed around two principles: upweighting large-magnitude value states in the scoring function, and introducing stochasticity to promote diversity in the retained cache. Both VaSE variants are training-free and require no model modifications. Throughout, we describe each variant at a single decoding step t where the buffer is full and eviction is triggered.

VaSE-AttnV. VaSE-AttnV adds value-aware reservation and stochastic sampling on top of SnapKV. We score each cached value $\boldsymbol{v}_i$ by the Range($\boldsymbol{v}_i$) of its coordinates (Eq. 6.5), which is dominated by the extreme coordinates of $\boldsymbol{v}_i$ and tracks the heavy-tailed magnitudes (Figure 6.2).

Given a value reservation budget $N_v < K$, we select the N_v candidates with the largest Range($\boldsymbol{v}_i$) and unconditionally retain them as the reserved set $\mathcal{R}_{\boldsymbol{V}}$. Following the periodic eviction framework (Section 6.2), we reserve the recent buffer B tokens as $\mathcal{R}_B$, forming the full reserved set $\mathcal{R} = \mathcal{R}_{\boldsymbol{V}} \cup \mathcal{R}_B$. The remaining $K - |\mathcal{R}|$ slots are filled by sampling from the SnapKV attention distribution. Concretely, let $\mathcal{C} = [K] \setminus \mathcal{R}_{\boldsymbol{V}}$ be the candidates not held by value reservation. The remaining $K - |\mathcal{R}|$ slots are drawn from $\mathcal{C}$ by weighted sampling

without replacement, with weights proportional to $\bar{\alpha}_i$, forming the final retained set. This preserves the SnapKV signal that high-attention tokens are more likely to be retained, while replacing the hard top$k$ threshold with a soft probabilistic one.

Stochastic sampling offers a key advantage over top$k$ in terms of token retention flexibility. At step t, let $\pi_i^{(t)}$ denote the probability of token i being retained, conditional on the candidate set $\mathcal{C}_t$ and the current attention weights $\{\bar{\alpha}_j^{(t)}\}$. After T steps, the probability that token i is kept in the KV cache can factorize as $\prod_{t=1}^{T} \pi_i^{(t)}$. Top$k$ forces $\pi_i^{(t)} \in \{0, 1\}$, which permanently discards any token falling below the cutoff. In contrast, sampling without replacement guarantees that for any token with a positive attention weight, $\pi_i^{(t)} \geq 1 - (1 - \bar{\alpha}_i^{(t)}/Z_t)^{K-|\mathcal{R}|} > 0$, where $Z_t = \sum_{j \in \mathcal{C}_t} \bar{\alpha}_j^{(t)}$. Thus, every token with a non-zero attention weight remains eligible at a future step.

VaSE-DKV. VaSE-DKV inherits value state awareness directly from CurDKV's product score, where the projected value norm $\ell_i^{(\boldsymbol{V})}(\boldsymbol{G})$ enters the score as a multiplicative factor that amplifies high-magnitude values. The change is that at each eviction step t, we resample $\boldsymbol{G}_t \in \mathbb{R}^{d \times r}$ with $\mathcal{N}(0, 1/r)$ entries independently of all past projections, and compute the leverage scores following Eq. 6.4. This improves upon CurDKV, in which a fixed $\boldsymbol{G}$ means a token that receives an unfavorable projection score is permanently removed from the retained set. Resampling $\boldsymbol{G}$ at each step resolves this by introducing a similar retention eligibility established above.

6.4 Experiments

6.4.1 Isolating the Effects of Value-Awareness and Stochasticity

We conduct a case study on GSM8K to isolate the individual contributions of value-awareness and stochasticity to the accuracy of Qwen3-4B.

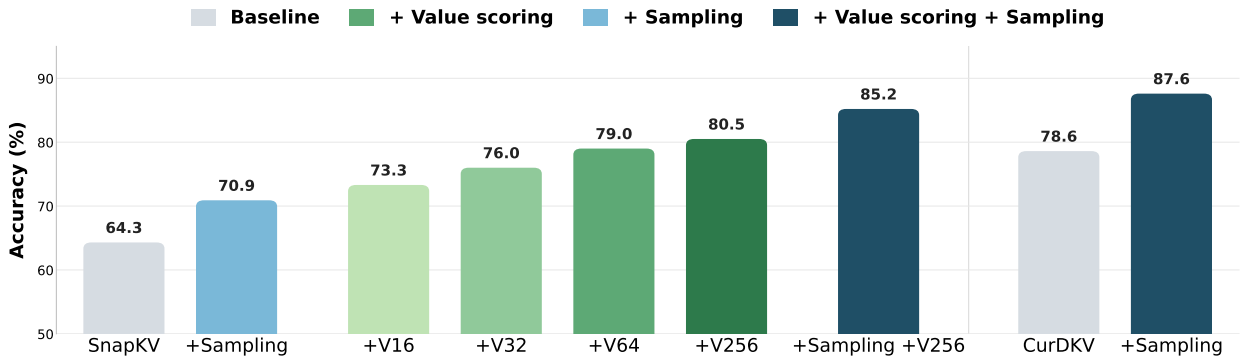


Figure 6.3: Accuracy of Qwen3-4B on GSM8K with a KV cache budget of K=512 (∼4× compression). **+V16** indicates reserving 16 slots in the budget K for value scoring. By combining stochastic sampling with value scoring, our methods ■ achieve accuracy close to the full model (88.4%).

Value Awareness. We experiment with different reservation budgets N_v to evaluate the efficacy of our value-scoring function Range($\boldsymbol{v}_i$) (Eq. 6.5). Building on top of the SnapKV baseline, we fill the remaining $K - N_v$ slots with KV pairs ranked highest by SnapKV's scoring function $\bar{\alpha}_i$ (Eq. 6.2). Figure 6.3 shows that when increasing the number of N_v from 16 to 256 (green bars), the accuracies on GSM8K increase consistently from 73.3%(+9.0%) to 80.5%(+16.2%), showing the importance of keeping large-range values from eviction.

Stochasticity. We introduce stochasticity into SnapKV and CurDKV to promote diversity in KV pair retention. For SnapKV (*leftmost* in Figure 6.3), replacing its deterministic top$k$ selection with $\bar{\alpha}_i$-weighted sampling raises GSM8K accuracy from 64.3% to 70.9%. Similarly, for CurDKV (*rightmost*), resampling Gaussian matrix $\boldsymbol{G}_t$ at each eviction step t throughout generation improves accuracy from 78.6% to 87.6%. The teal bars ■ show that value scoring and stochasticity are complementary, with the combination of both strategies achieving the highest accuracies.

6.4.2 Main Results

Setup. We evaluate VASE on Qwen3-4B and Qwen3-14B [171] across six reasoning tasks: AIME25, AIME26 [177], HMMT25 (Feb and Nov splits combined; HMMT [178]), GPQA-

Model	Method	AIME25	AIME26	HMMT25	GPQA-D	MATH	Avg.
Qwen3-4B	Full	66.41	62.71	45.94	56.19	93.93	65.04
	SeerAttention-R	58.59	60.42	40.42	49.94	84.67	58.81
	SnapKV	47.92	53.33	35.31	36.87	72.30	49.15
	R-KV	49.58	54.58	36.98	**49.05**	83.28	54.69
	CurDKV	48.75	54.58	33.23	37.94	74.42	49.78
	VaSE-DKV	57.29	61.88	39.58	43.94	**84.72**	57.48
	VaSE-AttnV	**59.17**	**62.08**	**44.38**	47.98	81.82	**59.09**
Qwen3-14B	Full	70.21	76.04	54.79	65.25	95.22	72.30
	SeerAttention-R	64.79	65.62	48.65	61.68	86.12	65.37
	R-KV	54.58	60.21	44.58	**59.09**	86.05	60.90
	CurDKV	53.75	61.25	39.58	48.48	75.42	55.70
	VaSE-DKV	63.33	68.75	49.38	56.76	**86.47**	64.94
	VaSE-AttnV	**63.96**	**72.29**	**50.00**	57.39	85.42	**65.81**

Table 6.2: Reasoning-task accuracy (%) of sparse attention methods on Qwen3-4B and Qwen3-14B with $\sim 25\%$ of full KV activated ($4\times$ compression for eviction methods). **Bold** marks the best eviction method of each task. Our eviction methods VaSE achieve comparable average accuracy to SeerAttention-R, the SOTA selection method.

Diamond [179], MATH [180], and LiveCodeBench-v6 [181].[3] All eviction methods operate under the periodic-eviction framework (Section 6.2) with a shared recency buffer of $B = 64$ and a persistent budget K. For each task, we first measure the average number of tokens N_{avg} under the uncompressed full model and then set the eviction budget K to approximately $N_{\text{avg}}/4$, where $K \in \{1024, 2048, 4096\}$. This yields a nominal $4\times$ compression ratio over the uncompressed baseline. SeerAttention-R is configured at the matching 25% activation ratio so that all sparse methods operate at the same effective attention sparsity. We use the models' default top-p configurations for all generations. We report pass@1 results, which are averaged over 16 independent samples for datasets under 100 examples, and 8 samples for larger datasets. We provide details of token statistics and the hyperparameters of different methods in Section D.2.

VaSE-AttnV achieves the best average accuracy. Table 6.2 shows that both VaSE variants outperform every prior eviction baseline on the per-model average and reach parity with the selection method SeerAttention-R. On Qwen3-4B, VaSE-AttnV achieves an aver-

[3]We use the evaluation scripts from SeerAttention-R: `https://github.com/microsoft/SeerAttention`

age accuracy of 59.09%, edging out SeerAttention-R (58.81%) and surpassing the strongest eviction baseline R-KV (54.69%) by 4.4%; VaSE-DKV reaches 57.48%, improving over CurDKV (49.78%) by 7.7%. The same pattern holds on Qwen3-14B: VaSE-AttnV (65.81%) matches SeerAttention-R (65.37%) and outperforms R-KV (60.90%) by 4.9%; VaSE-DKV (64.94%) improves over the deterministic CurDKV baseline (55.70%) by 9.2%. Overall, VaSE-AttnV achieves the best average accuracy among all methods. While SnapKV and R-KV baselines do not consider value states in their deterministic scoring, the results of VaSE-AttnV show the effectiveness of combining value awareness with stochasticity in the eviction scoring function. The substantial improvements of VaSE-DKV over CurDKV further highlight the importance of stochasticity. Since both variants of VaSE belong to the eviction family, we demonstrate that the VaSE recipe can recover accuracy without paying the memory cost of selection methods.

Code generation. Figure 6.5 (*Left*) reports pass@1 of Qwen3-4B on LiveCodeBench-v6-Medium under a 2048-token budget ($\sim$ 20% of full KV size). R-KV (62.6%), VaSE-DKV (61.9%), and VaSE-AttnV (63.5%) achieve comparable performance, while CurDKV only yields 34.6%. Surprisingly, SeerAttention-R (45.3%) underperforms eviction methods, with the exception of CurDKV. This is likely because the learning-based attention gate of SeerAttention-R struggles to generalize under domain shifts, while the training-free eviction methods are less affected.

Token Budgets. Figure 6.4 reports pass@1 accuracy of Qwen3-14B on AIME26 and HMMT25 as the KV cache budget increases from 2048 to 6144 tokens. At the tightest budget of 2048 tokens, which corresponds to roughly 7.5× compression of the full cache on AIME26 and 8.7× on HMMT25, both VaSE variants show clear improvement over R-KV and CurDKV baselines. As the budget increases, the performance gap diminishes with the accuracy of all methods recovering. Nevertheless, VaSE methods (teal lines) consistently outperform the baselines. We observe this pattern in both datasets: the accuracy gains

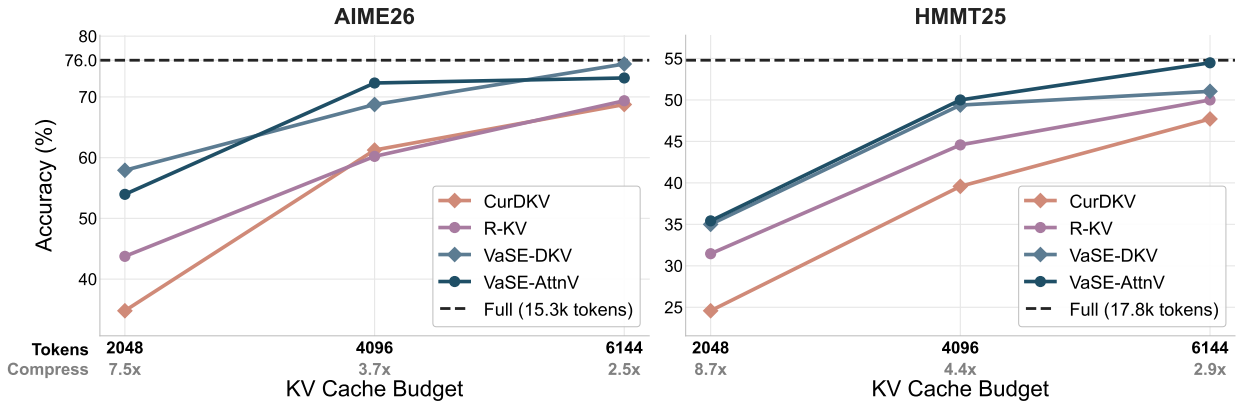


Figure 6.4: Pass@1 accuracy of Qwen3-14B under varying KV cache budgets. The full-cache model uses an average of 15.3k and 17.8k tokens on AIME26 and HMMT25, respectively. Each x-axis label reports the number of budget tokens (e.g., 2048) and its corresponding KV cache compression ratio to the full model (e.g., 7.5×).

from value awareness and stochastic eviction are largest under aggressive compression and diminish gracefully when the cache capacity is abundant (see D.26 in the appendix for results on GPQA-Diamond).

6.4.3 Large-Range Values Correlate with KV Quantization Error

Our findings regarding large-Range values have a deep connection with per-token KV cache quantization [182, 183], an alternative way to compress the KV cache. Formally, in asymmetric b-bit linear quantization, the range of the input value state $\boldsymbol{v}$, $[\min \boldsymbol{v}, \max \boldsymbol{v}]$, is mapped to the full range of the quantized integer space, $[0, 2^b - 1]$. The b-bit integer quantization-dequantization process can be expressed as:

$$Q(\boldsymbol{v}) = \left\lfloor \frac{\boldsymbol{v} - z_{\boldsymbol{v}}}{s_{\boldsymbol{v}}} \right\rceil, \quad \boldsymbol{v}' = Q(\boldsymbol{v}) \cdot s_{\boldsymbol{v}} + z_{\boldsymbol{v}}, \tag{6.6}$$

where $z_{\boldsymbol{v}} = \min \boldsymbol{v}$ is the zero point, $s_{\boldsymbol{v}} = (\max \boldsymbol{v} - \min \boldsymbol{v})/(2^b - 1) = \text{Range}(\boldsymbol{v})/(2^b - 1)$ is the scaling factor, and $\lfloor \cdot \rceil$ is the rounding operation. The scaling factor $s_{\boldsymbol{v}}$ represents the step size of quantization: if $\text{Range}(\boldsymbol{v})$ is small, $s_{\boldsymbol{v}}$ is small, leading to fine-grained quantization, and the reconstructed value $\boldsymbol{v}'$ is close to the original value $\boldsymbol{v}$; on the other hand, if $\text{Range}(\boldsymbol{v})$

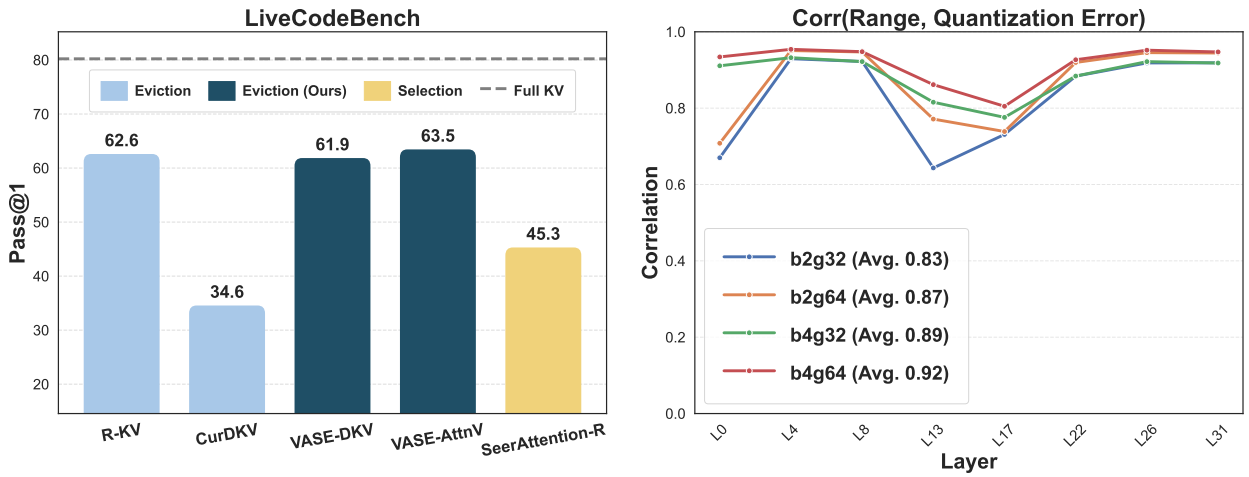


Figure 6.5: *Left*: Pass@1 results of Qwen3-4B on LiveCodeBench under a 2048-token budget (∼20% of full KV). R-KV and our VASE methods achieve the strongest performance. *Right*: Range($\boldsymbol{v}$) and per-token value cache quantization errors are highly correlated under different quantization configurations, where `b2g32` means 2-bit precision with a group size of 32.

is large, a large $s_{\boldsymbol{v}}$ results in a large and coarse step size, causing $\boldsymbol{v}'$ to deviate significantly from $\boldsymbol{v}$, which is known as the outlier issue [12] in LLM quantization.

We validate the relationship between Range($\boldsymbol{v}$) and per-token quantization error on the value cache empirically, using the HQQ quantizer [184] with different bit-widths and quantization group sizes. Quantization error is measured via the mean-squared reconstruction error between $\boldsymbol{v}$ and $\boldsymbol{v}'$ [10]. We extract the full value cache from Qwen3-4B during the generation of GSM8K examples, quantize them, and compute the per-token mean-squared error. Figure 6.5 (*Right*) demonstrates that Range($\boldsymbol{v}$) and the reconstruction error are highly correlated across layers under different quantization configurations, suggesting that large-Range value states lead to large information loss under per-token KV cache quantization.

6.5 Related Work

KV Cache Compression Methods. Prior work reduces the cost of the KV cache through different approaches: reducing the number of tokens via sparse attention [185, 186, 187, 188], reducing the precision via quantization [189, 182, 190, 191], and reducing the hidden

dimension via low-rank decomposition [192, 193, 194]. In this chapter, we focus on sparse attention methods. Our findings on the importance of large-magnitude values also carry profound implications for per-token KV cache quantization [182].

Decoding Phase Compression. Inference proceeds in two phases. In the prefill phase, the model processes the T-token input prompt and computes T key–value pairs in parallel. In the decode phase, the model generates one token at a time and appends the corresponding KV pairs to the cache. Unlike the prefill phase, the decode phase is memory-bound, and the growing cache intensifies the memory bottleneck. For reasoning models, the KV pairs from the decode phase dominate the cost due to the long thinking traces. Therefore, several recent works [170, 173, 167, 195] have targeted KV cache compression during the decoding phase of reasoning models. This chapter also focuses on the decoding phase compression of reasoning models.

Leveraging Value-State Magnitude for KV Cache Scoring. Typically, sparse attention involves scoring key-value pairs by their importance. Because the output of attention is a weighted combination of values, Guo, Kamigaito, and Watanabe [175] and Devoto, Jeblick, and Jégou [196] score KV pairs by weighting attention scores with the norm of their corresponding values. In this chapter, we deeply investigate the importance of large-magnitude values. Rather than multiplying attention and value scores, we propose an alternative, VaSE-AttnV, which reserves dedicated slots in the token budget to ensure that large-magnitude values are always preserved in the cache.

6.6 Discussion and Conclusion

Implications for KV cache quantization. We show that large-Range value states are not only critical for maintaining KV cache eviction accuracy but also a major source of error under quantization. Therefore, a promising future direction is to investigate a mixed-

precision approach [182] to maintain the precision of large-Range value states. For example, they can be placed in a high-precision reserved cache, while the remaining KV cache is quantized to a lower bit-width. Once the high-precision budget is used up, the earliest value state in this reserved cache can be quantized and "evicted" into the low-precision cache.

Large-magnitude value states are crucial for reasoning progression. Our observation in Figure D.24 suggests that large-magnitude value states play a special role in maintaining reasoning progression. After these values are evicted, generations often degenerate into repetitive loops, where the model repeatedly restates intermediate reasoning without making progress. One hypothesis is that these values help transition between latent reasoning steps, preventing the model from collapsing into self-reinforcing paths. Therefore, their removal traps the model in repetitive loops. One potential future direction is to analyze what information large-magnitude values carry and how they influence long-form reasoning.

Conclusion. We present VASE, a training-free KV cache eviction framework designed to address the memory bottleneck of reasoning models. We identify two critical factors for maintaining accuracy during KV cache compression: protecting large-magnitude value states from eviction, and introducing stochasticity to improve KV cache diversity. Taking both factors into consideration, VASE provides a simple and effective recipe for improving existing eviction methods. Our findings highlight the importance of value states in long-form reasoning and suggest new directions for designing memory-efficient inference methods.

Chapter 7

Conclusion

This dissertation advances **component attribution**, the systematic study of how individual model units causally shape model behavior, as a principled framework for diagnosing and mitigating undesirable LLM behaviors. Our research spans three connected pillars: creating rigorous benchmarks that enable fair evaluation of attribution methods, conducting targeted analyses that reveal the model components responsible for specific undesirable behaviors, and translating those findings into actionable interventions that improve model reliability. The model behaviors we studied, verbatim memorization, ICL instability, quantization brittleness, and overthinking, target very different application domains, but they share a common structure: each arises from specific model components behaving in ways that are hidden from input-output analysis alone. We apply component attribution to make these hidden mechanisms visible and provide actionable insights to mitigate the undesirable behaviors.

We believe that as LLMs continue to scale and are deployed in high-stakes applications, the ability to reason about model behavior in terms of internal components will be increasingly important for building reliable AI systems. We hope this dissertation serves as a foundation for future work that deepens our understanding of LLM internals and translates that understanding into models whose behaviors we can predict, explain, and control.

Bibliography

[1] Yoshua Bengio et al. "A neural probabilistic language model". In: *Journal of machine learning research* 3.Feb (2003), pp. 1137–1155.

[2] Matthew E. Peters et al. "Deep Contextualized Word Representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Ed. by Marilyn Walker, Heng Ji, and Amanda Stent. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 2227–2237. DOI: 10.18653/v1/N18-1202. URL: https://aclanthology.org/N18-1202/.

[3] Jacob Devlin et al. "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)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423/.

[4] Alec Radford et al. "Language Models are Unsupervised Multitask Learners". In: *OpenAI Blog* 1.8 (2019).

[5] Tom Brown et al. "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 1877–1901.

[6] Ashish Vaswani et al. "Attention is All You Need". In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017.

[7] Hugo Touvron et al. "LLaMA: Open and Efficient Foundation Language Models". In: *arXiv preprint arXiv:2302.13971* (2023).

[8] Albert Q Jiang et al. "Mistral 7B". In: *arXiv preprint arXiv:2310.06825* (2023).

[9] Team Gemma et al. "Gemma: Open models based on gemini research and technology". In: *arXiv preprint arXiv:2403.08295* (2024).

[10] Elias Frantar et al. "GPTQ: Accurate Post-Training Quantization for Generative Pre-Trained Transformers". In: *arXiv preprint arXiv:2210.17323* (2022).

[11] Ji Lin et al. "AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration". In: *Proceedings of Machine Learning and Systems*. Ed. by P. Gibbons, G. Pekhimenko, and C. De Sa. Vol. 6. 2024, pp. 87–100. URL: https://p

roceedings.mlsys.org/paper_files/paper/2024/file/42a452cbafa9dd64e9ba4aa95cc1ef21-Paper-Conference.pdf.

[12] Tim Dettmers et al. "GPT3.int8(): 8-bit Matrix Multiplication for Transformers at Scale". In: *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh et al. 2022. URL: https://openreview.net/forum?id=dXiGWqBoxaD.

[13] Zhenyu Zhang et al. "H2o: Heavy-hitter oracle for efficient generative inference of large language models". In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 34661–34710.

[14] Guangxuan Xiao et al. "Efficient Streaming Language Models with Attention Sinks". In: *The Twelfth International Conference on Learning Representations*. 2024.

[15] Yuhong Li et al. "SnapKV: LLM Knows What You are Looking for Before Generation". In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. 2024.

[16] Ting-Yun Chang, Jesse Thomason, and Robin Jia. "Do Localization Methods Actually Localize Memorized Data in LLMs? A Tale of Two Benchmarks". In: *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. Mexico City, Mexico: Association for Computational Linguistics, June 2024, pp. 3190–3211. DOI: 10.18653/v1/2024.naacl-long.176. URL: https://aclanthology.org/2024.naacl-long.176.

[17] Nicholas Carlini et al. "The Secret Sharer: Evaluating and Testing Unintended Memorization in Neural Networks". In: *28th USENIX Security Symposium*. 2019, pp. 267–284.

[18] Eric Lehman et al. "Does BERT Pretrained on Clinical Notes Reveal Sensitive Data?" In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Online: Association for Computational Linguistics, June 2021, pp. 946–959. DOI: 10.18653/v1/2021.naacl-main.73. URL: https://aclanthology.org/2021.naacl-main.73.

[19] Jooyoung Lee et al. "Do language models plagiarize?" In: *Proceedings of the ACM Web Conference 2023*. 2023, pp. 3637–3647. URL: https://dl.acm.org/doi/abs/10.1145/3543507.3583199.

[20] Nicholas Carlini et al. "Extracting Training Data from Large Language Models". In: *30th USENIX Security Symposium*. 2021, pp. 2633–2650.

[21] Yinzhi Cao and Junfeng Yang. "Towards Making Systems Forget with Machine Unlearning". In: *2015 IEEE Symposium on Security and Privacy*. IEEE. 2015, pp. 463–480.

[22] Lucas Bourtoule et al. "Machine Unlearning". In: *2021 IEEE Symposium on Security and Privacy*. IEEE. 2021, pp. 141–159.

[23] Damai Dai et al. "Knowledge Neurons in Pretrained Transformers". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2022, pp. 8493–8502.

[24] Kevin Meng et al. "Locating and Editing Factual Associations in GPT". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 17359–17372.

[25] Ting-Yun Chang, Jesse Thomason, and Robin Jia. "When Parts Are Greater Than Sums: Individual LLM Components Can Outperform Full Models". In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Ed. by Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen. Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 10280–10299. DOI: 10.18653/v1/2024.emnlp-main.574. URL: https://aclanthology.org/2024.emnlp-main.574/.

[26] Sewon Min et al. "Rethinking the Role of Demonstrations: What Makes In-Context Learning Work?" In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 11048–11064. DOI: 10.18653/v1/2022.emnlp-main.759. URL: https://aclanthology.org/2022.emnlp-main.759.

[27] Melanie Sclar et al. "Quantifying Language Models' Sensitivity to Spurious Features in Prompt Design or: How I learned to start worrying about prompt formatting". In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=RIu5lyNXjT.

[28] Anton Voronov, Lena Wolf, and Max Ryabinin. "Mind Your Format: Towards Consistent Evaluation of In-Context Learning Improvements". In: *Findings of the Association for Computational Linguistics ACL 2024*. Bangkok, Thailand and virtual meeting: Association for Computational Linguistics, Aug. 2024, pp. 6287–6310. DOI: 10.18653/v1/2024.findings-acl.375. URL: https://aclanthology.org/2024.findings-acl.375.

[29] Zihao Zhao et al. "Calibrate Before Use: Improving Few-Shot Performance of Language Models". In: *Proceedings of the 38th International Conference on Machine Learning*. PMLR. 2021, pp. 12697–12706.

[30] Ari Holtzman et al. "Surface Form Competition: Why the Highest Probability Answer Isn't Always Right". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 7038–7051. DOI: 10.18653/v1/2021.emnlp-main.564. URL: https://aclanthology.org/2021.emnlp-main.564.

[31] Yanda Chen et al. "On the Relation between Sensitivity and Accuracy in In-Context Learning". In: *Findings of the Association for Computational Linguistics: EMNLP 2023*. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 155–167. DOI: 10.18653/v1/2023.findings-emnlp.12. URL: https://aclanthology.org/2023.findings-emnlp.12.

[32] Ting-Yun Chang et al. "Why Do Some Inputs Break Low-Bit LLM Quantization?" In: *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. Ed. by Christos Christodoulopoulos et al. Suzhou, China: Association for Computational Linguistics, Nov. 2025, pp. 3410–3429. ISBN: 979-8-89176-332-6. DOI: 10.18653/v1/2025.emnlp-main.168. URL: https://aclanthology.org/2025.emnlp-main.168/.

[33] Eldar Kurtic et al. ""Give Me BF16 or Give Me Death"? Accuracy-Performance Trade-Offs in LLM Quantization". In: *arXiv preprint arXiv:2411.02355* (2024).

[34] Tim Dettmers and Luke Zettlemoyer. "The case for 4-bit precision: k-bit inference scaling laws". In: *International Conference on Machine Learning*. PMLR. 2023, pp. 7750–7774.

[35] Guangxuan Xiao et al. "SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models". In: *Proceedings of the 40th International Conference on Machine Learning*. PMLR. 2023, pp. 38087–38099.

[36] Saleh Ashkboos et al. "QuaRot: Outlier-Free 4-Bit Inference in Rotated LLMs". In: *arXiv preprint arXiv:2404.00456* (2024).

[37] Ting-Yun Chang et al. "Value-Aware Stochastic KV Cache Eviction for Reasoning Models". In: *arXiv preprint arXiv:2606.03928* (2026).

[38] OpenAI. *Learning to Reason with LLMs*. https://openai.com/index/learning-to-reason-with-llms. 2024.

[39] Qwen. *QwQ: Reflect Deeply on the Boundaries of the Unknown*. https://qwenlm.github.io/blog/qwq-32b-preview/. Nov. 2024.

[40] Daya Guo et al. "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning". In: *arXiv preprint arXiv:2501.12948* (2025).

[41] Charlie Victor Snell et al. "Scaling LLM Test-Time Compute Optimally Can be More Effective than Scaling Parameters for Reasoning". In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: https://openreview.net/forum?id=4FWAwZtd2n.

[42] Suyu Ge et al. "Model Tells You What to Discard: Adaptive KV Cache Compression for LLMs". In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=uNrFpDPMyo.

[43] Nelson Elhage et al. "A Mathematical Framework for Transformer Circuits". In: *Transformer Circuits Thread* (2021).

[44] Catherine Olsson et al. "In-Context Learning and Induction Heads". In: *arXiv preprint arXiv:2209.11895* (2022).

[45] Kevin Wang et al. "Interpretability in the Wild: a Circuit for Indirect Object Identification in GPT-2 Small". In: *International Conference on Learning Representations*. 2023.

[46] Mor Geva et al. "Transformer Feed-Forward Layers Are Key-Value Memories". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 2021, pp. 9484–9495.

[47] Mor Geva et al. "Transformer Feed-Forward Layers Build Predictions by Promoting Concepts in the Vocabulary Space". In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. 2022, pp. 30–45.

[48] Stella Biderman et al. "Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling". In: *Proceedings of the 40th International Conference on Machine Learning*. PMLR. 2023, pp. 2397–2430.

[49] Song Han, Huizi Mao, and William J Dally. "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding". In: *International Conference on Learning Representations*. 2016.

[50] Zhuang Liu et al. "Learning Efficient Convolutional Networks through Network Slimming". In: *ICCV*. 2017.

[51] Christos Louizos, Max Welling, and Diederik P Kingma. "Learning Sparse Neural Networks through L_0 Regularization". In: *International Conference on Learning Representations*. 2018.

[52] nostalgebraist. *Interpreting GPT: the Logit Lens*. https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens. 2020.

[53] Yao Lu et al. “Fantastically Ordered Prompts and Where to Find Them: Overcoming Few-Shot Prompt Order Sensitivity”. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2022, pp. 8086–8098.

[54] Peter Hase et al. “Does Localization Inform Editing? Surprising Differences in Causality-Based Localization vs. Knowledge Editing in Language Models”. In: *Thirty-seventh Conference on Neural Information Processing Systems*. 2023. URL: https://openreview.net/forum?id=EldbUlZtbd.

[55] Zhuocheng Gong et al. “Finding the Dominant Winning Ticket in Pre-Trained Language Models”. In: *Findings of the Association for Computational Linguistics: ACL 2022*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 1459–1472. DOI: 10.18653/v1/2022.findings-acl.115. URL: https://aclanthology.org/2022.findings-acl.115.

[56] Abhishek Panigrahi et al. “Task-Specific Skill Localization in Fine-Tuned Language Models”. In: *Proceedings of the 40th International Conference on Machine Learning*. PMLR. 2023, pp. 27011–27033.

[57] Babak Hassibi and David Stork. “Second order derivatives for network pruning: Optimal Brain Surgeon”. In: *Advances in Neural Information Processing Systems*. Vol. 5. Morgan-Kaufmann, 1992. URL: https://proceedings.neurips.cc/paper_files/paper/1992/file/303ed4c69846ab36c2904d3ba8573050-Paper.pdf.

[58] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958.

[59] Yasumasa Onoe et al. “Entity Cloze By Date: What LMs Know About Unseen Entities”. In: *Findings of the Association for Computational Linguistics: NAACL 2022*. Seattle, United States: Association for Computational Linguistics, July 2022. DOI: 10.18653/v1/2022.findings-naacl.52. URL: https://aclanthology.org/2022.findings-naacl.52/.

[60] Jiwei Li, Will Monroe, and Dan Jurafsky. “Understanding Neural Networks through Representation Erasure”. In: *arXiv preprint arXiv:1612.08220* (2016).

[61] Mor Geva et al. “Dissecting Recall of Factual Associations in Auto-Regressive Language Models”. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 2023, pp. 12216–12235.

[62] Leo Gao et al. “The Pile: An 800GB Dataset of Diverse Text for Language Modeling”. In: *arXiv preprint arXiv:2101.00027* (2021).

[63] Anton Sinitsin et al. "Editable Neural Networks". In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=HJedXaEtvS.

[64] Eric Mitchell et al. "Fast Model Editing at Scale". In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=0DcZxeWfOPt.

[65] Matthew D Zeiler and Rob Fergus. "Visualizing and Understanding Convolutional Networks". In: *Computer Vision – ECCV 2014*. Springer. 2014, pp. 818–833.

[66] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. "Axiomatic Attribution for Deep Networks". In: *Proceedings of the 34th International Conference on Machine Learning*. PMLR. 2017, pp. 3319–3328.

[67] Xinwei Wu et al. "DEPN: Detecting and Editing Privacy Neurons in Pretrained Language Models". In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 2023, pp. 2875–2886.

[68] Xiaohan Chen et al. "EarlyBERT: Efficient BERT Training via Early-bird Lottery Tickets". In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Online: Association for Computational Linguistics, Aug. 2021, pp. 2195–2207. DOI: 10.18653/v1/2021.acl-long.171. URL: https://aclanthology.org/2021.acl-long.171.

[69] Rui Zheng et al. "Robust Lottery Tickets for Pre-trained Language Models". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 2211–2224. DOI: 10.18653/v1/2022.acl-long.157. URL: https://aclanthology.org/2022.acl-long.157.

[70] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. "The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables". In: *International Conference on Learning Representations*. 2017. URL: https://openreview.net/forum?id=S1jE5L5gl.

[71] Eric Jang, Shixiang Gu, and Ben Poole. "Categorical Reparameterization with Gumbel-Softmax". In: *International Conference on Learning Representations*. 2017. URL: https://openreview.net/forum?id=rkE3y85ee.

[72] Shankar Padmanabhan et al. "Propagating Knowledge Updates to LMs Through Distillation". In: *Advances in Neural Information Processing Systems*. Vol. 36. 2023, pp. 47124–47142.

[73] Nelson F. Liu et al. “Do Question Answering Modeling Improvements Hold Across Benchmarks?” In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 13186–13218. DOI: 10.18653/v1/2023.acl-long.736. URL: https://aclanthology.org/2023.acl-long.736.

[74] Ian Tenney, Dipanjan Das, and Ellie Pavlick. “BERT Rediscovers the Classical NLP Pipeline”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 2019, pp. 4593–4601.

[75] Alec Radford, Rafal Jozefowicz, and Ilya Sutskever. “Learning to Generate Reviews and Discovering Sentiment”. In: *arXiv preprint arXiv:1704.01444* (2017).

[76] Wes Gurnee et al. “Finding Neurons in a Haystack: Case Studies with Sparse Probing”. In: *Transactions on Machine Learning Research* (2023).

[77] Anshita Gupta et al. “Editing Common Sense in Transformers”. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 2023, pp. 8214–8232.

[78] Róbert Csordás, Sjoerd van Steenkiste, and Jürgen Schmidhuber. “Are Neural Nets Modular? Inspecting Functional Modularity Through Differentiable Weight Masks”. In: *International Conference on Learning Representations*. 2021.

[79] Steven Cao, Victor Sanh, and Alexander Rush. “Low-Complexity Probing via Finding Subnetworks”. In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Online: Association for Computational Linguistics, June 2021, pp. 960–966. DOI: 10.18653/v1/2021.naacl-main.74. URL: https://aclanthology.org/2021.naacl-main.74.

[80] Negar Foroutan et al. “Discovering Language-neutral Sub-networks in Multilingual Language Models”. In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 7560–7575. DOI: 10.18653/v1/2022.emnlp-main.513. URL: https://aclanthology.org/2022.emnlp-main.513.

[81] Deniz Bayazit et al. “Discovering Knowledge-Critical Subnetworks in Pretrained Language Models”. In: *arXiv preprint arXiv:2310.03084* (2023).

[82] Bryan Klimt and Yiming Yang. “Introducing the Enron corpus”. In: *CEAS*. Vol. 45. 2004, pp. 92–96. URL: https://www.ceas.cc/papers-2004/168.pdf.

[83] Pratyush Maini et al. “Can Neural Network Memorization Be Localized?” In: *Proceedings of the 40th International Conference on Machine Learning*. PMLR. 2023.

[84] Long Ouyang et al. "Training Language Models to Follow Instructions with Human Feedback". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 27730–27744.

[85] Amanda Bertsch et al. "In-Context Learning with Long-Context Models: An In-Depth Exploration". In: *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. Albuquerque, New Mexico: Association for Computational Linguistics, Apr. 2025, pp. 12119–12149. ISBN: 979-8-89176-189-6. DOI: 10.18653/v1/2025.naacl-long.605. URL: https://aclanthology.org/2025.naacl-long.605/.

[86] Hugo Touvron et al. "Llama 2: Open foundation and fine-tuned chat models". In: *arXiv preprint arXiv:2307.09288* (2023).

[87] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, et al. "The llama 3 herd of models". In: *arXiv preprint arXiv:2407.21783* (2024).

[88] Jiachang Liu et al. "What Makes Good In-Context Examples for GPT-3?" In: *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*. Dublin, Ireland and Online: Association for Computational Linguistics, May 2022, pp. 100–114. DOI: 10.18653/v1/2022.deelio-1.10. URL: https://aclanthology.org/2022.deelio-1.10.

[89] nostalgebraist. "interpreting GPT: the logit lens". In: (2020). URL: https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens.

[90] Biao Zhang and Rico Sennrich. "Root Mean Square Layer Normalization". In: *Advances in Neural Information Processing Systems*. Vol. 32. 2019.

[91] Stephen H. Bach et al. "PromptSource: An Integrated Development Environment and Repository for Natural Language Prompts". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Ed. by Valerio Basile, Zornitsa Kozareva, and Sanja Stajner. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 93–104. DOI: 10.18653/v1/2022.acl-demo.9. URL: https://aclanthology.org/2022.acl-demo.9/.

[92] Callum McDougall et al. "Copy suppression: Comprehensively understanding an attention head". In: *arXiv preprint arXiv:2310.04625* (2023).

[93] Lean Wang et al. "Label Words are Anchors: An Information Flow Perspective for Understanding In-Context Learning". In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 9840–9855. DOI: 10.18653/v1/2023.emnlp-main.609. URL: https://aclanthology.org/2023.emnlp-main.609.

[94] Matt Gardner et al. "Evaluating Models' Local Decision Boundaries via Contrast Sets". In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, Nov. 2020, pp. 1307–1323. DOI: 10.18653/v1/2020.findings-emnlp.117. URL: https://aclanthology.org/2020.findings-emnlp.117.

[95] Yizhong Wang et al. "Super-NaturalInstructions: Generalization via Declarative Instructions on 1600+ NLP Tasks". In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 5085–5109. DOI: 10.18653/v1/2022.emnlp-main.340. URL: https://aclanthology.org/2022.emnlp-main.340.

[96] Chandra Bhagavatula et al. "Abductive Commonsense Reasoning". In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=Byg1v1HKDB.

[97] Ohad Rubin, Jonathan Herzig, and Jonathan Berant. "Learning To Retrieve Prompts for In-Context Learning". In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Seattle, United States: Association for Computational Linguistics, July 2022, pp. 2655–2671. DOI: 10.18653/v1/2022.naacl-main.191. URL: https://aclanthology.org/2022.naacl-main.191.

[98] Nils Reimers and Iryna Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 3982–3992. DOI: 10.18653/v1/D19-1410. URL: https://aclanthology.org/D19-1410.

[99] Yu Fei et al. "Mitigating Label Biases for In-context Learning". In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 14014–14031. DOI: 10.18653/v1/2023.acl-long.783. URL: https://aclanthology.org/2023.acl-long.783.

[100] Han Zhou et al. "Batch Calibration: Rethinking Calibration for In-Context Learning and Prompt Engineering". In: *arXiv preprint arXiv:2309.17249* (2023).

[101] Yao Lu et al. "Fantastically Ordered Prompts and Where to Find Them: Overcoming Few-Shot Prompt Order Sensitivity". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 8086–8098. DOI: 10.18653/v1/2022.acl-long.556. URL: https://aclanthology.org/2022.acl-long.556.

[102] Ting-Yun Chang and Robin Jia. "Data Curation Alone Can Stabilize In-context Learning". In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 8123–8144. DOI: 10.18653/v1/2023.acl-long.452. URL: https://aclanthology.org/2023.acl-long.452.

[103] Yao Fu et al. "Complexity-Based Prompting for Multi-step Reasoning". In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=yf1icZHC-l9.

[104] Sewon Min et al. "Noisy Channel Language Model Prompting for Few-Shot Text Classification". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 5316–5330. DOI: 10.18653/v1/2022.acl-long.365. URL: https://aclanthology.org/2022.acl-long.365.

[105] Simran Arora et al. "Ask Me Anything: A simple strategy for prompting language models". In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=bhUPJnS2g0X.

[106] Rishabh Agarwal et al. "Many-Shot In-Context Learning". In: *arXiv preprint arXiv:2404.11018* (2024).

[107] Ethan Perez, Douwe Kiela, and Kyunghyun Cho. "True Few-Shot Learning with Language Models". In: *Advances in Neural Information Processing Systems*. Ed. by A. Beygelzimer et al. 2021. URL: https://openreview.net/forum?id=ShnM-rRh4T.

[108] Harshay Shah, Andrew Ilyas, and Aleksander Madry. "Decomposing and Editing Predictions by Modeling Model Computation". In: *arXiv preprint arXiv:2404.11534* (2024).

[109] Xiaozhi Wang et al. "Finding Skill Neurons in Pre-trained Transformer-based Language Models". In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 11132–11152. DOI: 10.18653/v1/2022.emnlp-main.765. URL: https://aclanthology.org/2022.emnlp-main.765.

[110] Guillaume Alain and Yoshua Bengio. *Understanding intermediate layers using linear classifier probes*. 2017. URL: https://openreview.net/forum?id=ryF7rTqgl.

[111] Mor Geva et al. "Dissecting Recall of Factual Associations in Auto-Regressive Language Models". In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 12216–12235. DOI: 10.18653/v1/2023.emnlp-main.751. URL: https://aclanthology.org/2023.emnlp-main.751.

[112] Maximilian Li, Xander Davies, and Max Nadeau. "Circuit breaking: Removing model behaviors with targeted ablation". In: *arXiv preprint arXiv:2309.05973* (2023).

[113] Nicholas Goldowsky-Dill et al. "Localizing model behavior with path patching". In: *arXiv preprint arXiv:2304.05969* (2023).

[114] Kenneth Li et al. "Inference-Time Intervention: Eliciting Truthful Answers from a Language Model". In: *Advances in Neural Information Processing Systems* 36 (2024).

[115] Paul Michel, Omer Levy, and Graham Neubig. "Are Sixteen Heads Really Better than One?" In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/2c601ad9d2ff9bc8b282670cdd54f69f-Paper.pdf.

[116] Elena Voita et al. "Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 5797–5808. DOI: 10.18653/v1/P19-1580. URL: https://aclanthology.org/P19-1580.

[117] Roee Hendel, Mor Geva, and Amir Globerson. "In-Context Learning Creates Task Vectors". In: *Findings of the Association for Computational Linguistics: EMNLP 2023*. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 9318–9333. DOI: 10.18653/v1/2023.findings-emnlp.624. URL: https://aclanthology.org/2023.findings-emnlp.624.

[118] Sheng Liu et al. "In-context Vectors: Making In Context Learning More Effective and Controllable Through Latent Space Steering". In: *International Conference on Machine Learning*. PMLR. 2024, pp. 32287–32307.

[119] Jack Merullo, Carsten Eickhoff, and Ellie Pavlick. "Language Models Implement Simple Word2Vec-style Vector Arithmetic". In: *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. Mexico City, Mexico: Association for Computational Linguistics, June 2024, pp. 5030–5047. DOI: 10.18653/v1/2024.naacl-long.281. URL: https://aclanthology.org/2024.naacl-long.281.

[120] Eric Todd et al. "Function Vectors in Large Language Models". In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=AwyxtyMwaG.

[121] Qinan Yu, Jack Merullo, and Ellie Pavlick. "Characterizing Mechanisms for Factual Recall in Language Models". In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Singapore: Association for Computational

Linguistics, Dec. 2023, pp. 9924–9959. DOI: 10.18653/v1/2023.emnlp-main.615. URL: https://aclanthology.org/2023.emnlp-main.615.

[122] Tony Wang et al. "Forbidden Facts: An Investigation of Competing Objectives in Llama 2". In: *Socially Responsible Language Modelling Research*. 2023. URL: https://openreview.net/forum?id=1kavETZX3Y.

[123] AI@Meta. "Llama 3 Model Card". In: (2024). URL: https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

[124] Team OLMo et al. "2 OLMo 2 Furious". In: (2024). arXiv: 2501.00656 [cs.CL]. URL: https://arxiv.org/abs/2501.00656.

[125] Team Qwen. *Qwen3*. Apr. 2025. URL: https://qwenlm.github.io/blog/qwen3/.

[126] Ying Sheng et al. "FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU". In: *Proceedings of the 40th International Conference on Machine Learning*. Vol. 202. Proceedings of Machine Learning Research. PMLR, July 2023, pp. 31094–31116. URL: https://proceedings.mlr.press/v202/sheng23a.html.

[127] Gunho Park et al. "LUT-GEMM: Quantized Matrix Multiplication based on LUTs for Efficient Inference in Large-Scale Generative Language Models". In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=gLARhFLE0F.

[128] Changhun Lee et al. "Owq: Outlier-aware weight quantization for efficient fine-tuning and inference of large language models". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 12. 2024, pp. 13355–13364.

[129] Tanishq Kumar et al. "Scaling Laws for Precision". In: *The Thirteenth International Conference on Learning Representations*. 2025.

[130] Jerry Chee et al. "Quip: 2-bit quantization of large language models with guarantees". In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 4396–4429.

[131] Haozheng Luo et al. "Fast and Low-Cost Genomic Foundation Models via Outlier Removal". In: *Forty-second International Conference on Machine Learning*. 2025. URL: https://openreview.net/forum?id=oyUiJmkD7H.

[132] nostalgebraist. "interpreting GPT: the logit lens". In: (2020). URL: https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens.

[133] Nikhil Prakash et al. “Fine-Tuning Enhances Existing Mechanisms: A Case Study on Entity Tracking”. In: *Proceedings of the 2024 International Conference on Learning Representations*. arXiv:2402.14811. 2024.

[134] Noam Shazeer. “GLU variants improve transformer”. In: *arXiv preprint arXiv:2002.05202* (2020).

[135] Orevaoghene Ahia, Julia Kreutzer, and Sara Hooker. “The Low-Resource Double Bind: An Empirical Study of Pruning for Low-Resource Machine Translation”. In: *Findings of the Association for Computational Linguistics: EMNLP 2021*. Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 3316–3333. DOI: 10.18653/v1/2021.findings-emnlp.282. URL: https://aclanthology.org/2021.findings-emnlp.282/.

[136] Kelly Marchisio et al. “How Does Quantization Affect Multilingual LLMs?” In: *Findings of the Association for Computational Linguistics: EMNLP 2024*. Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 15928–15947. DOI: 10.18653/v1/2024.findings-emnlp.935. URL: https://aclanthology.org/2024.findings-emnlp.935/.

[137] Guilherme Penedo et al. “The FineWeb Datasets: Decanting the Web for the Finest Text Data at Scale”. In: *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. 2024. URL: https://openreview.net/forum?id=n6SCkn2QaG.

[138] Alex Mallen et al. “When Not to Trust Language Models: Investigating Effectiveness of Parametric and Non-Parametric Memories”. In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 9802–9822. DOI: 10.18653/v1/2023.acl-long.546. URL: https://aclanthology.org/2023.acl-long.546/.

[139] Xiuying Wei et al. “Outlier suppression+: Accurate quantization of large language models by equivalent and effective shifting and scaling”. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 2023, pp. 1648–1665.

[140] Colin Raffel et al. “Exploring the limits of transfer learning with a unified text-to-text transformer”. In: *Journal of machine learning research* 21.140 (2020), pp. 1–67.

[141] Tim Dettmers et al. “QLoRA: Efficient Finetuning of Quantized LLMs”. In: *Advances in Neural Information Processing Systems* 36 (2023).

[142] Han Guo et al. “Fast Matrix Multiplications for Lookup Table-Quantized LLMs”. In: *Findings of the Association for Computational Linguistics: EMNLP 2024*. Mi-

ami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 12419–12433. DOI: 10.18653/v1/2024.findings-emnlp.724. URL: https://aclanthology.org/2024.findings-emnlp.724/.

[143] Mengzhao Chen et al. "EfficientQAT: Efficient Quantization-Aware Training for Large Language Models". In: *Proceedings of the 41st International Conference on Machine Learning*. PMLR. 2024.

[144] Zechun Liu et al. "LLM-QAT: Data-Free Quantization Aware Training for Large Language Models". In: *Findings of the Association for Computational Linguistics: ACL 2024*. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 467–484. DOI: 10.18653/v1/2024.findings-acl.26. URL: https://aclanthology.org/2024.findings-acl.26/.

[145] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation". In: *arXiv preprint arXiv:1308.3432* (2013).

[146] Olga Kovaleva et al. "BERT Busters: Outlier Dimensions that Disrupt Transformers". In: *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. Online: Association for Computational Linguistics, Aug. 2021, pp. 3392–3405. DOI: 10.18653/v1/2021.findings-acl.300. URL: https://aclanthology.org/2021.findings-acl.300/.

[147] Xiuying Wei et al. "Outlier suppression: Pushing the limit of low-bit transformer language models". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 17402–17414.

[148] Jinze Bai et al. "Qwen Technical Report". In: *arXiv preprint arXiv:2309.16609* (2023).

[149] Team Qwen. *Qwen2.5: A Party of Foundation Models*. Sept. 2024. URL: https://qwenlm.github.io/blog/qwen2.5/.

[150] Jiale Chen, Torsten Hoefler, and Dan Alistarh. "The Geometry of LLM Quantization: GPTQ as Babai's Nearest Plane Algorithm". In: *arXiv preprint arXiv:2507.18553* (2025).

[151] Rewon Child et al. "Generating long sequences with sparse transformers". In: *arXiv preprint arXiv:1904.10509* (2019).

[152] Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. "Quantizable transformers: Removing outliers by helping attention heads do nothing". In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 75067–75096.

[153] Ruihao Gong et al. "Llmc: Benchmarking large language model quantization with a versatile compression toolkit". In: *arXiv preprint arXiv:2405.06001* (2024).

[154] Zechun Liu et al. "SpinQuant: LLM Quantization with Learned Rotations". In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: https://openreview.net/forum?id=ogO6DGE6FZ.

[155] Yann N Dauphin et al. "Language modeling with gated convolutional networks". In: *International conference on machine learning*. PMLR. 2017, pp. 933–941.

[156] Alexander Wettig et al. "Organize the Web: Constructing Domains Enhances Pre-Training Data Curation". In: *Forty-second International Conference on Machine Learning*. 2025. URL: https://openreview.net/forum?id=boSqwdvJVC.

[157] Kelechi Ogueji et al. "Intriguing Properties of Compression on Multilingual Models". In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 9092–9110. DOI: 10.18653/v1/2022.emnlp-main.619. URL: https://aclanthology.org/2022.emnlp-main.619/.

[158] Jiachen Zhu et al. "Transformers without Normalization". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2025.

[159] Xingyu Chen et al. "Do not think that much for 2+ 3=? on the overthinking of o1-like llms". In: *arXiv preprint arXiv:2412.21187* (2024).

[160] Guangxiang Zhao et al. "Explicit sparse transformer: Concentrated attention through explicit selection". In: *arXiv preprint arXiv:1912.11637* (2019).

[161] Iz Beltagy, Matthew E Peters, and Arman Cohan. "Longformer: The long-document transformer". In: *arXiv preprint arXiv:2004.05150* (2020).

[162] Manzil Zaheer et al. "Big bird: Transformers for longer sequences". In: *Advances in neural information processing systems* 33 (2020), pp. 17283–17297.

[163] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. "Reformer: The Efficient Transformer". In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=rkgNKkHtvB.

[164] Luka Ribar et al. "SparQ Attention: Bandwidth-Efficient LLM Inference". In: *Forty-first International Conference on Machine Learning*.

[165] Jiaming Tang et al. "QUEST: Query-Aware Sparsity for Efficient Long-Context LLM Inference". In: *Proceedings of the 41st International Conference on Machine Learning*. 2024.

[166] Lijie Yang et al. "TidalDecode: Fast and Accurate LLM Decoding with Position Persistent Sparse Attention". In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: https://openreview.net/forum?id=EkfLaCJ7bk.

[167] Yizhao Gao et al. "Sparse Attention Adaptation for Long Reasoning". In: *The Fourteenth International Conference on Learning Representations*. 2026. URL: https://openreview.net/forum?id=c5BOcHM6J8.

[168] Alessio Devoto et al. "A Simple and Effective L_2 Norm-Based Strategy for KV Cache Compression". In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 18476–18499. DOI: 10.18653/v1/2024.emnlp-main.1027. URL: https://aclanthology.org/2024.emnlp-main.1027/.

[169] Matanel Oren et al. "Transformers are Multi-State RNNs". In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Ed. by Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen. Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 18724–18741. DOI: 10.18653/v1/2024.emnlp-main.1043. URL: https://aclanthology.org/2024.emnlp-main.1043/.

[170] Zefan Cai et al. "R-KV: Redundancy-aware KV Cache Compression for Reasoning Models". In: *The Thirty-ninth Annual Conference on Neural Information Processing Systems*. 2025.

[171] An Yang et al. "Qwen3 technical report". In: *arXiv preprint arXiv:2505.09388* (2025).

[172] Ayan Sengupta, Siddhant Chaudhary, and Tanmoy Chakraborty. "Value-Guided KV Compression for LLMs via Approximated CUR Decomposition". In: *The Thirty-ninth Annual Conference on Neural Information Processing Systems*. 2025.

[173] Jiwon Song et al. "Reasoning Path Compression: Compressing Generation Trajectories for Efficient LLM Reasoning". In: *The Thirty-ninth Annual Conference on Neural Information Processing Systems*. 2025.

[174] Karl Cobbe et al. "Training Verifiers to Solve Math Word Problems". In: *arXiv preprint arXiv:2110.14168* (2021).

[175] Zhiyu Guo, Hidetaka Kamigaito, and Taro Watanabe. "Attention Score is not All You Need for Token Importance Indicator in KV Cache Reduction: Value Also Matters". In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 21158–21166. DOI: 10.18653/v1/2024.emnlp-main.1178. URL: https://aclanthology.org/2024.emnlp-main.1178/.

[176] Tianyu Guo et al. "Active-Dormant Attention Heads: Mechanistically Demystifying Extreme-Token Phenomena in LLMs". In: *The Second Conference on Parsimony and Learning (Recent Spotlight Track)*. 2025. URL: https://openreview.net/forum?id=Zx6WUbE9J7.

[177] Art of Problem Solving. *AIME Problems and Solutions*. https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions.

[178] HMMT. *HMMT 2025*. Accessed: 2025. 2025. URL: https://www.hmmt.org/.

[179] David Rein et al. "GPQA: A Graduate-Level Google-Proof Q&A Benchmark". In: *First Conference on Language Modeling*. 2024. URL: https://openreview.net/forum?id=Ti67584b98.

[180] Dan Hendrycks et al. "Measuring Mathematical Problem Solving With the MATH Dataset". In: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. 2021. URL: https://openreview.net/forum?id=7Bywt2mQsCe.

[181] Naman Jain et al. "LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code". In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: https://openreview.net/forum?id=chfJJYC3iL.

[182] Zirui Liu et al. "KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache". In: *International Conference on Machine Learning*. PMLR. 2024, pp. 32332–32344.

[183] Zunhai Su and Kehong Yuan. "KVSink: Understanding and Enhancing the Preservation of Attention Sinks in KV Cache Quantization for LLMs". In: *Second Conference on Language Modeling*. 2025. URL: https://openreview.net/forum?id=gIqb6zWZoO.

[184] Hicham Badri and Appu Shaji. *Half-Quadratic Quantization of Large Machine Learning Models*. 2023. URL: https://dropbox.github.io/hqq_blog/.

[185] Zichang Liu et al. "Scissorhands: Exploiting the Persistence of Importance Hypothesis for LLM KV Cache Compression at Test Time". In: *Thirty-seventh Conference on Neural Information Processing Systems*. 2023. URL: https://openreview.net/forum?id=JZfg6wGi6g.

[186] Huiqiang Jiang et al. "MInference 1.0: Accelerating Pre-filling for Long-Context LLMs via Dynamic Sparse Attention". In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. 2024.

[187] Prajwal Singhania et al. "Loki: Low-rank Keys for Efficient Sparse Attention". In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. 2024. URL: https://openreview.net/forum?id=raABeiV71j.

[188] Vivek Chari and Benjamin Van Durme. "Compactor: Calibrated Query-Agnostic KV Cache Compression with Approximate Leverage Scores". In: *arXiv preprint arXiv:2507.08143* (2025).

[189] Coleman Richard Charles Hooper et al. "KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization". In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. 2024. URL: https://openreview.net/forum?id=0LXotew9Du.

[190] Junhyuck Kim et al. "Not All Bits Are Equal: Scale-Dependent Memory Optimization Strategies for Reasoning Models". In: *arXiv preprint arXiv:2510.10964* (2025).

[191] Amir Zandieh et al. "TurboQuant: Online Vector Quantization with Near-optimal Distortion Rate". In: *The Fourteenth International Conference on Learning Representations*. 2026. URL: https://openreview.net/forum?id=tO3ASKZlok.

[192] Utkarsh Saxena et al. "Eigen attention: Attention in low-rank space for kv cache compression". In: *Findings of the Association for Computational Linguistics: EMNLP 2024*. 2024, pp. 15332–15344.

[193] Aixin Liu et al. "Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model". In: *arXiv preprint arXiv:2405.04434* (2024).

[194] Chi-Chih Chang et al. "Palu: KV-Cache Compression with Low-Rank Projection". In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: https://openreview.net/forum?id=LWMS4pk2vK.

[195] Zhenyuan Guo et al. "Dynamic Thinking-Token Selection for Efficient Reasoning in Large Reasoning Models". In: *arXiv preprint arXiv:2601.18383* (2026).

[196] Alessio Devoto, Maximilian Jeblick, and Simon Jégou. "Expected attention: Kv cache compression by estimating attention from future queries distribution". In: *arXiv preprint arXiv:2510.00636* (2025).

[197] Vladimir I Levenshtein. "Binary Codes Capable of Correcting Deletions, Insertions, and Reversals". In: *Soviet Physics Doklady* 10 (1965), pp. 707–710.

[198] Katherine Lee et al. "Deduplicating Training Data Makes Language Models Better". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2022.

[199] Chiyuan Zhang et al. "Counterfactual Memorization in Neural Language Models". In: *Advances in Neural Information Processing Systems*. Vol. 36. 2023.

[200] Richard Socher et al. "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1631–1642. URL: https://aclanthology.org/D13-1170.

[201] Xiang Zhang, Junbo Zhao, and Yann LeCun. "Character-level Convolutional Networks for Text Classification". In: *Advances in Neural Information Processing Systems*. Vol. 28. Curran Associates, Inc., 2015. URL: https://proceedings.neurips.cc/paper_files/paper/2015/file/250cf8b51c773f3f8dc8b4be867a9a02-Paper.pdf.

[202] Christopher Clark et al. "BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions". 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)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 2924–2936. DOI: 10.18653/v1/N19-1300. URL: https://aclanthology.org/N19-1300.

[203] Alex Wang et al. "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding". In: *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 353–355. DOI: 10.18653/v1/W18-5446. URL: https://aclanthology.org/W18-5446.

[204] Mohammad Taher Pilehvar and Jose Camacho-Collados. "WiC: the Word-in-Context Dataset for Evaluating Context-Sensitive Meaning Representations". 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)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 1267–1273. DOI: 10.18653/v1/N19-1128. URL: https://aclanthology.org/N19-1128.

[205] Adina Williams, Nikita Nangia, and Samuel Bowman. "A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 1112–1122. DOI: 10.18653/v1/N18-1101. URL: https://aclanthology.org/N18-1101.

[206] Alexey Romanov and Chaitanya Shivade. "Lessons from Natural Language Inference in the Clinical Domain". In: *Proceedings of the 2018 Conference on Empirical Methods*

in Natural Language Processing. Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 1586–1596. DOI: 10.18653/v1/D18-1187. URL: https://aclanthology.org/D18-1187.

[207] Peter Clark et al. "Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge". In: *arXiv:1803.05457v1* (2018).

[208] Swaroop Mishra et al. "Cross-Task Generalization via Natural Language Crowdsourcing Instructions". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 3470–3487. DOI: 10.18653/v1/2022.acl-long.244. URL: https://aclanthology.org/2022.acl-long.244.

[209] Vivek Ramanujan et al. "What's Hidden in a Randomly Weighted Neural Network?" In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 11893–11902.

[210] Kelly Zhang and Samuel Bowman. "Language Modeling Teaches You More than Translation Does: Lessons Learned Through Auxiliary Syntactic Task Analysis". In: *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 359–361. DOI: 10.18653/v1/W18-5448. URL: https://aclanthology.org/W18-5448.

[211] John Hewitt and Percy Liang. "Designing and Interpreting Probes with Control Tasks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 2733–2743. DOI: 10.18653/v1/D19-1275. URL: https://aclanthology.org/D19-1275.

[212] Together Computer. *RedPajama: An Open Source Recipe to Reproduce LLaMA training dataset*. Apr. 2023. URL: https://github.com/togethercomputer/RedPajama-Data.

[213] Zhangir Azerbayev et al. *Llemma: An Open Language Model For Mathematics*. 2023. arXiv: 2310.10631 [cs.CL].

[214] Keiran Paster et al. *OpenWebMath: An Open Dataset of High-Quality Mathematical Web Text*. 2023. arXiv: 2310.06786 [cs.AI].

[215] Paul Jaccard. "The distribution of the flora in the alpine zone. 1". In: *New phytologist* 11.2 (1912), pp. 37–50.

[216] Ruibin Xiong et al. “On layer normalization in the transformer architecture”. In: *International conference on machine learning*. PMLR. 2020, pp. 10524–10533.

A Appendix for Chapter 3

A.1 The Loss for Information Injection

In the INJ Benchmark, we use regular language modeling loss to train the LLM θ on a new sequence $x = [x_1, \ldots, x_{\mathcal{T}}]$ of $\mathcal{T}$ tokens. Formally,

$$\frac{1}{\mathcal{T}-1} \sum_{t=2}^{\mathcal{T}} -\log P_\theta(x_t|x_{<t}) \tag{1}$$

Here, the index t starts from 2, because all the LLMs we use (GPT2 and Pythia models) do not add ¡bos¿ tokens to data when doing language modeling in their pretraining. Therefore, there is no loss on the first token x_1 and the total loss is averaged across $\mathcal{T}-1$ token.

A.2 Details of IG

Recall that a sequence $x = (p, s)$ consists of a prefix p and a suffix $s = [s_1, \ldots, s_T]$. Denote $\mathrm{P}(\hat{h}_t^l)$ as the LLM output probability of token s_t if we replace the original hidden state at the l-th layer, $h_t^l \in \mathbb{R}^{d_2}$, with a new hidden state $\hat{h}_t^l \in \mathbb{R}^{d_2}$:

$$\mathrm{P}(\hat{h}_t^l) = P_\theta(s_t|p, s_{<t}, \hat{h}_t^l) \tag{2}$$

To calculate the integrated gradients along the i-th neuron dimension, we gradually change $\hat{h}_t^l$ from a zero vector[1] to the original hidden state h_t^l, and cumulating the gradients of $\mathrm{P}(\cdot)$ along the i-th dimension. Finally, we get the attribution score $\mathcal{A}^l(i)$ by averaging the integrated

[1]We follow Dai et al. [23] to set the *baseline* in integrated gradients to a zero vector that has the same shape as h_t^l.

gradients across the suffix length T:

$$\mathbf{IG}_i(z) := \ z_i \int_{\alpha=0}^{1} \frac{\partial \mathrm{P}(\alpha z)}{\partial z_i} d\alpha, \tag{3}$$

$$\mathcal{A}^l(i) = \frac{1}{T} \sum_{t=1}^{T} \mathbf{IG}_i(h_t^l) \tag{4}$$

where $\mathbf{IG}_i(h_t^l)$ is the integrated gradients along the i-th neuron dimension in the l-th layer at the t-th timestep, when the input is $[p, s_{<t}]$. Sundararajan, Taly, and Yan [66] compute Riemann sum to approximate Eq. 3, which uses a fixed number of intervals to approximate the integrals. We closely follow the implementation of `https://github.com/EleutherAI/knowledge-neurons`.

A.3 Details of Slimming

We initialize every mask value m_i^l as 1, which is equivalent to running the pretrained LLM without masking. When training the mask, we clip every m_i^l to $[0, 1]$. Note that for both Slimming and Hard Concrete, because we are learning a mask on each neuron, we do not apply any random dropout during training.

A.4 Details of Hard Concrete

Louizos, Welling, and Kingma [51] obtain the hard concrete r.v. $\bar{m}_i^l$ by first stretching the binary concrete r.v. $\hat{m}_i^l$ (Eq. 3.11) from the interval $(0, 1)$ to (γ, ζ), where $\gamma = -0.1, \zeta = 1.1$, and then clip the value to the $[0, 1]$ interval:

$$\bar{m}_i^l = \min\left(1, \max\left(0,\ \hat{m}_i^l\ \cdot (\zeta - \gamma) + \gamma\right)\right) \tag{5}$$

They then use L_0 regularization to encourage sparsity on the weights after applying the mask $\bar{m}^l$. After reparametrization, they have the regularization $\mathcal{R}(\bar{m}^l)$:

$$\mathcal{R}(\bar{m}^l) \ = \sum_{i=1}^{d_2} \sigma \left(\log m_i^l - C \right), \tag{6}$$

where $C = \beta \log \frac{-\gamma}{\zeta}$ is a constant.

A.5 Expanding the dataset of INJ Benchmark

We double the data size of the INJ Benchmark by including the newly released ECBD 2022, 2023 splits, having 328 distinct definition sentences from ECBD 2021-2023. We experiment with this expanded dataset on GPT2, injection ratio=0.1%, using the same hyperparameters as Table 3.2. Table A.1 shows the results on ECBD 2021-2023 are very close to the ones on ECBD-2021 only (Table 3.2), suggesting that our conclusions hold when we increase the dataset size.

	GPT2 124M		
ratio = 0.1%	**R@0.1%**	**R@0.2%**	**R@0.5%**
HC	58.3	81.6	**94.6**
Slim	**59.2**	**84.3**	94.5
IG	53.3	73.0	84.1
Activation	11.1	26.5	52.7

Table A.1: The INJ Benchmark results of GPT2 on the expanded dataset, ECBD 2021-2023.

A.6 Do random seeds affect the results of the INJ Benchmark?

In INJ Benchmark, we sample different sets of weights for different examples (see Algorithm 1); thus, the results reported in Table 3.2 are averaged over many different choices of weights. To further show that random seeds do not affect our results, we run an additional experiment on GPT2, with the injection ratio=0.1%. Specifically, for each example, we choose a different

random seed and thus choose a different set of weights to inject the example. Comparing the new results in Table A.2 with the original ones in Table 3.2, we find that the recall scores barely change for all localization methods. Also, for each method, we run paired two-tailed t-tests comparing the recalls between the original and new seeds and observe that all pairs have p-values > 0.05, suggesting that differences between random seeds are not significant.

	GPT2 124M		
ratio = 0.1%	**R@0.1%**	**R@0.2%**	**R@0.5%**
HC	57.9	81.0	94.6
Slim	**59.6**	**84.4**	**94.8**
IG	54.0	73.7	83.9
Activation	11.4	26.4	52.7

Table A.2: The INJ Benchmark results with a new set of random seeds. The Recall@k% scores are very similar to the original ones in Table 3.2, showing the INJ Benchmark is not sensitive to the choice of random seed.

A.7 Computation costs of different methods

Among all five localization methods, Activations is the most computationally efficient, because Eq. 3.8 only requires one forward pass. Both the pruning-based methods Slimming and Hard Concrete perform fast, as only the masks are trainable. Calculating integrated gradients (IG) is time-consuming, while Zero-Out is the worst, because it leaves out every

	Time
Activations	$\sim$ 0.3 sec
Slimming	$\sim$ 12 sec
Hard Concrete	$\sim$ 1 min
IG	$\sim$ 43 min
Zero-Out	$\sim$ 8.5 hr

Table A.3: The elapsed time of different methods to do localization (i.e., assign attribution scores to every neuron) on one sequence memorized by Pythia-6.9B. We time all methods on a single RTX A6000 GPU.

neuron one by one. We compare the computational cost of different methods on one sequence memorized by Pythia-deduped-6.9B, where each sequence in the collected set $\mathcal{X}$ consists of a 32-token prefix and a 48-token suffix. We follow the common implementation that sets the number of intervals to 20 for Riemann sum in IG. Table A.3 shows the elapsed time calculated on an RTX A6000 48G GPU. When running IG and Zero-Out we patch and batch the activations to reach 99% GPU utilities. Still, applying Zero-Out to do localization on one sequence costs 8.5 hours, and $\mathcal{X}$ contains 500 sequences in total. Due to the extremely heavy computation cost, we do not have the results of Zero-Out on Pythia-6.9B in the DEL Benchmark.

A.8 Details of Data Collection

We show some collected examples in Tables A.9&A.10. Table A.4 reports how well the pretrained LLMs memorize sequences in the collected datasets.

	Acc	**Dist**	**PPL**	Len
GPT2-XL	99.3%	0.48	10.18	150
Pythia-deduped-2.8B	98.8%	1.07	5.58	160
Pythia-deduped-6.9B	99.7%	0.20	5.24	167

Table A.4: Quantifying memorization of the collected datasets. The high Accuracy (Acc) and low Levenshtein distance (Dist) show our collected sequences ($\mathcal{X}$) are indeed well memorized by LLMs. The last column (Len) reports the average suffix length of each dataset at the character level. We also measure the perplexity (PPL) on sequences sampled from the Pile-dedupe ($\mathcal{D}$).

The pretrained sequences of Pythia models. EleutherAI releases the exact batches used by Pythia models during pretraining, where each sequence in a batch consists of 2049 tokens [2]. We first randomly downsample the pretraining batches to a subset $\mathcal{Z}$ of 102400 sequences. Then, we use our criteria in 3.3.2 to determine whether Pythia memorizes a sequence in the subset. After filtering, there remain $500 \sim 1000$ sequences in the subsets

[2] `https://github.com/EleutherAI/pythia#exploring-the-dataset`

for both Pythia-deduped-2.8B and Pythia-deduped-6.9B; we simply sample 505 of them respectively as our collected datasets.

We also randomly sample a subset of 2048 sequences ($\mathcal{D}$), each consisting of 128 tokens, to measure the perplexity of all LLMs we evaluate. We ensure that $\mathcal{Z} \cap \mathcal{D} = \varnothing$, so there is no overlap between the collected memorized sequences and sequences for perplexity.

Filtering with greedy decoding. Given the prefix p as the prompt to the LLM, we generate the suffix $\bar{s} = [\bar{s_1}, \ldots, \bar{s_{48}}]$ with greedy decoding, where

$$\bar{s_t} = \underset{w \in \text{Voc}}{\text{argmax}}\ P_\theta(w|p, \bar{s}_{<t}). \tag{7}$$

We then calculate the Levenshtein distance [197] between the true suffix s and the generated one $\bar{s}$, filtering out sequences with a distance greater than 20 characters. Note $\bar{s}$ is different from $\hat{s}$ in Eq 3.4, which is generated by teacher-forcing and is used to calculate memorization scores.

Deduplication. Although we use the deduplicated version of the dataset and models, the Pile-dedupe and Pythia-deduped models, we still find lots of near-duplicated sequences. Thus, we further deduplicate the collected memorized sequences. In particular, we follow Lee et al. [198] to represent each sequence with a set of 5-grams when calculating the Jaccard index. Among a set of duplicates, we select the one that is best memorized, i.e., having the lowest Levenshtein distance on the generated suffix $\bar{s_t}$ (Eq. 7), and discard the others.

Manually searched data. With our searching criteria in 3.3.2, we can only identify less than 10 memorized sequences from subsets of the Pile-dedupe, Common Crawl, and Wikipedia, probably because OpenAI carefully preprocesses the data before training GPT2-XL. Thus, we actively search for potentially memorized data, such as famous poems and common lists of sorted items. We collect 105 sequences memorized by GPT2-XL and man-

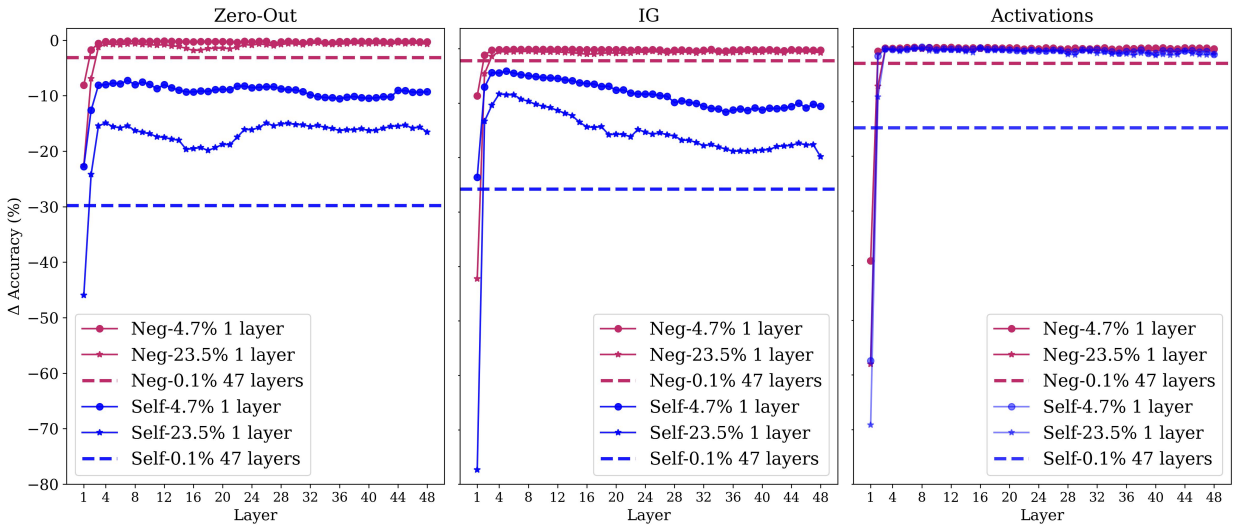


Figure A.1: The DEL Benchmark results of ZERO-OUT, IG, and ACTIVATIONS methods when dropping out the same number of neurons in a single layer, where the blue lines show Δ Self-Acc and the red lines show Δ Neg-Acc. Under the same "neuron budget", dropping out neurons in multiple layers (blue dashed lines) substantially outperforms dropout in a single layer, implying that memorized information is stored in a distributed fashion over multiple layers. Besides, dropping out neurons in the bottom layer greatly hurts the memorization of negative examples (red lines), suggesting that the bottom layer encodes general information.

ually categorize them (see Tables 3.1 & A.9), including 31 examples from Carlini et al. [20]. We set the prefix and suffix of a sequence by trial and error to ensure high memorization Accuracy. Unlike automatic searches that tend to find templated texts or uninteresting data with repetitive tokens [199], our manual search ensures better data quality and enables us to analyze memorization within and across categories.

In particular, Figures A.2 & A.3 show that different localization methods constantly confuse sequences of related categories. For example, they are unable to disentangle neurons of different quotes and identify a small set of neurons responsible for both the order of Zodiac Signs and the order of Planets.

Responsible checklist. Note the `Contact Info` category of our manually collected dataset only contains public data, such as mailing addresses of corporate headquarters and famous buildings; thus, it does not have any potential risk of revealing private information. Sim-

ilarly, our memorized datasets for Pythia models are a subset of the Pile, a public corpus under the MIT License.

A.9 Hyperparameters

In the INJ Benchmark, the ECBD-2021 set contains 156 definition sequences. For the DEL Benchmark, we collect a set of 505, 505, and 105 sequences memorized by Pythia-deduped-6.9B, Pythia-deduped-2.8B, and GPT2-XL, respectively. For each set, we sample 5 sequences as the dev set, using the dev set performance to determine the hyperparameters for each LLM. The hyperparameters include the integrated gradient steps, i.e., the number of intervals in Riemann sum for integral approximation in IG; the temperature β and the initialization value of parameters m in Hard Concrete; the learning rate, the number of training epochs, and λ, which balances the memorization loss and the sparsity loss, in Slimming and Hard Concrete. We observe that both Slimming and Hard Concrete are sensitive to the choice of hyperparameters. On the other hand, we find the performance of IG does not improve when using more integrated gradient steps, where we experiment with different steps ranging from 20 to 300. Thus, we set the step to 20 for all examples to reduce the heavy computation costs.

A.10 More experiments comparing dropout in one layer with multiple layers.

In 3.5.4, neurons are localized by IG. In this section, we conduct the same experiment using Zero-Out and Activations methods. Figure A.1 shows that dropping out N neurons in multiple layers (Self-0.1% 47 layers) even outperforms dropping out $5 \times N$ neurons in a single layer (Self-23.5% 1 layer), except for the bottom layers where the memorization of both target and negative examples are greatly hurt. Hence, we believe the memory of a piece of information is distributed across layers; meanwhile, only a few weights in each layer

are mainly responsible for the memorization (3.5.2).

We do not have the single-layer results of SLIMMING and HARD CONCRETE, because both methods train the masks of all neurons jointly, which requires us to retrain the masks only on a single layer to obtain their attribution scores. In comparison, the other three methods consider each neuron individually, allowing us to use the same attribution scores to select neurons in a single layer and make direct comparisons with the results in Table 3.3 (the dashed lines in Figure A.1).

A.11 Predicting top neurons across layers

In the INJ Benchmark, we randomly sample weights across layers to inject the data, instead of sampling a fixed percentage of weights per layer (see Algorithm 1). Hence, it may seem more natural to predict top-k% of neurons across layers; we experiment with this alternative in Table A.5.

Comparing the results of Table 3.2 and Table A.5, we find that predicting top neurons per layer outperforms predicting top neurons across layers. This is because all localization methods assign larger attribution scores to neurons in the bottom layers, barely predicting neurons in the upper layers if we rank neurons globally. On the other hand, Table 3.2 and Table A.5 show consistent results. Our findings and the ranking of different methods are coherent whether we rank neurons per layer or globally.

A.12 Implementation Details

Table A.6 summarizes the architectures of LLMs we use. We run most experiments on RTX3090 24G GPUs; experiments involving Pythia-6.9B are run on RTXA6000 48G GPUs. We use `transformers 4.31.0` and `pytorch 1.13`.

	GPT2 124M			GPT2-XL 1.5B			Pythia-deduped 2.8B			Pythia-deduped 6.9B		
	R@1%	R@2%	R@5%	R@1%	R@2%	R@5%	R@1%	R@2%	R@5%	R@1%	R@2%	R@5%
ratio = 1%												
Hard Concrete	**46.6**	**66.8**	**88.0**	**21.8**	**25.1**	**32.8**	33.3	**48.4**	**70.7**	31.5	47.5	**69.4**
Slimming	43.1	64.6	79.9	5.2	11.5	27.0	**33.6**	47.3	59.8	**35.0**	**49.6**	63.4
Zero-Out	24.0	36.8	52.7	4.2	7.3	13.5	10.1	14.3	20.5	-	-	-
IG	10.3	18.1	36.3	1.4	4.8	12.2	6.1	10.8	21.1	8.9	13.9	24.1
Activations	2.5	4.4	9.8	1.5	2.8	6.8	3.2	5.1	21.6	4.1	6.3	17.4
Random	1.0	2.0	5.0	1.0	2.0	5.0	1.0	2.0	5.0	1.0	2.0	5.0
ratio = 0.1%	@0.1%	@0.2%	@0.5%	@0.1%	@0.2%	@0.5%	@0.1%	@0.2%	@0.5%	@0.1%	@0.2%	@0.5%
Hard Concrete	51.2	77.4	**96.4**	**49.8**	**57.5**	**63.6**	**45.6**	**65.5**	**85.9**	28.7	40.7	55.8
Slimming	**62.7**	**87.0**	95.4	18.1	35.1	54.0	45.0	62.6	73.6	**39.1**	**52.1**	**64.3**
Zero-Out	57.4	81.7	91.9	14.7	20.9	31.1	16.4	20.6	25.8	-	-	-
IG	36.0	55.0	75.5	2.5	3.5	6.0	12.6	16.4	21.9	19.7	23.6	28.9
Activations	9.0	12.9	23.4	3.5	4.6	6.7	8.0	16.8	40.5	21.2	31.4	50.2
Random	0.1	0.2	0.5	0.1	0.2	0.5	0.1	0.2	0.5	0.1	0.2	0.5

Table A.5: The INJ Benchmark. The average Recall@k% of different methods when predicting top-k% of neurons *across* layers. The results are consistent with Table 3.2, where methods predict a fixed k% of neurons in each layer.

	# Layers	# Neurons
GPT2 124M	12	3072
GPT2-XL 1.5B	48	6400
Pythia-deduped-2.8B	32	10240
Pythia-deduped-6.9B	32	16384

Table A.6: The number of layers and the number of FFN neurons in each layer of different LLMs.

	GPT2 124M			GPT2-XL 1.5B			Pythia-deduped 2.8B			Pythia-deduped 6.9B		
	R@1%	R@2%	R@5%	R@1%	R@2%	R@5%	R@1%	R@2%	R@5%	R@1%	R@2%	R@5%
injection ratio = 1%												
HARD CONCRETE	$9E-85$	$7E-86$	$2E-77$	$2E-100$	$8E-108$	$8E-111$	$3E-114$	$3E-121$	$6E-138$	$5E-111$	$6E-129$	$3E-155$
SLIMMING	$1E-72$	$8E-73$	$4E-62$	$1E-66$	$5E-77$	$2E-84$	$3E-111$	$5E-119$	$4E-119$	$1E-121$	$3E-134$	$2E-132$
	@0.1%	@0.2%	@0.5%	@0.1%	@0.2%	@0.5%	@0.1%	@0.2%	@0.5%	@0.1%	@0.2%	@0.5%
injection ratio = 0.1%												
HARD CONCRETE	$6E-03$	$9E-06$	$1E-18$	$2E-96$	$3E-87$	$6E-73$	$7E-91$	$7E-110$	$1E-127$	$2E-42$	$7E-77$	$3E-83$
SLIMMING	$3E-10$	$4E-17$	$2E-20$	$1E-62$	$2E-73$	$4E-71$	$6E-96$	$7E-101$	$4E-99$	$1E-52$	$1E-54$	$1E-52$

Table A.7: The p-values of the INJ Benchmark. H_0: IG and pruning-based methods, HARD CONCRETE or SLIMMING, have identical expected Recall@k% scores on ECBD 2021 examples. As we have 24 settings in total, we run 24 one-tailed paired t-tests with Bonferroni correction, setting the significance level $\alpha = \frac{0.05}{24}$. We color the results that have p-values $> \alpha$.

	GPT2-XL 1.5B		Pythia-deduped 2.8B		Pythia-deduped 6.9B	
dropout ratio =	0.1%	0.5%	0.1%	0.5%	0.1%	0.5%
HARD CONCRETE	$1.6E-10$	$3.8E-17$	$5.3E-61$	$4.1E-78$	$2.2E-61$	$3.2E-90$
SLIMMING	$1.6E-04$	$1.8E-15$	$1.2E-01$	$8.9E-55$	$2.4E-24$	$2.0E-60$

Table A.8: The p-values of the DEL Benchmark, where we focus on the memorization accuracy of the target examples. H_0: IG and pruning-based methods, HARD CONCRETE or SLIMMING, have identical expected Δ Self-Acc scores on the memorized sequences. As we have 6 settings in total, we run 6 one-tailed paired t-tests with Bonferroni correction, setting the significance level $\alpha = \frac{0.05}{6}$. We color the results that have p-values $> \alpha$.

Email	100%	Write to Jon Hilsenrath at jon.hilsenrath@wsj.com
Zodiac Signs	100%	Aries Taurus Gemini Cancer Leo Virgo Libra Scorpio Sagittarius Capricorn Aquarius Pisces
Patreon	100%	Thank you to our Patreon Supporters: Saintsofwar, Anon, Lord_Of_Fapping, Dryzak, Chabalbac, ioNz, LaX, VNT
Declaration of Independence	100%	We hold these truths to be self-evident, that all men are created equal, that they are endowed by their Creator with certain unalienable Rights, that among these are Life, Liberty and the pursuit of Happiness.
Trump	100%	Sorry losers and haters, but my I.Q. is one of the highest -and you all know it! Please don't feel so stupid or insecure, it's not your fault.
Newton	100%	I do not know what I may appear to the world, but to myself I seem to have been only like a boy playing on the sea-shore, and diverting myself in now and then finding a smoother pebble or a prettier shell than ordinary, whilst the great ocean of truth lay all undiscovered before me.
Dr. MLK	100%	And when this happens, and when we allow freedom ring, when we let it ring from every village and every hamlet, from every state and every city, we will be able to speed up that day when all of God's children, black men and white men, Jews and Gentiles, Protestants and Catholics, will be able to join hands and sing in the words of the old Negro spiritual, "Free at last! Free at last! Thank God Almighty, we are free at last"
Genesis	100%	In the beginning God created the heaven and the earth. And the earth was without form, and void; and darkness was upon the face of the deep. And the Spirit of God moved upon the face of the waters. And God said, Let there be light: and there was light.
The Road Not Taken	100%	Two roads diverged in a yellow wood,\n\nAnd sorry I could not travel both\n\nAnd be one traveler, long I stood\n\nAnd looked down one as far as I could\n\nTo where it bent in the undergrowth;\n\nThen took the other, as just as fair,\n\nAnd having perhaps the better claim,\n\nBecause it was grassy and wanted wear

Table A.9: Examples of our manually collected data. The prompt (prefix) is colored in brown. The numbers are the Accuracy (Eq. 3.5) of GPT2-XL on memorizing the sequences, where 100% Accuracy means the true suffix can be fully reconstructed with greedy decoding.

Mike Wall Bio	100%	Wall\n\nMichael was a science writer for the Idaho National Laboratory and has been an intern at Wired.com, The Salinas Californian newspaper, and the SLAC National Accelerator Laboratory. He has also worked as a herpetologist and wildlife biologist. He has a Ph.D. in evolutionary biology from the University of Sydney, Australia, a bachelor's degree from the
Hardware	100%	PCs) may be defined as a desktop, floor standing, or portable microcomputer that includes a system unit having a central processing unit (CPU) and associated volatile and non-volatile memory, including random access memory (RAM) and basic input/output system read only memory (BIOS ROM), a system monitor, a keyboard, one or more flexible diskette drives, a CD-ROM drive,
Contact Info of Skyhorse Publishing	100%	, or educational purposes. Special editions can also be created to specifications. For details, contact the Special Sales Department, Arcade Publishing, 307 West 36th Street, 11th Floor, New York, NY 10018 or arcade@skyhorsepublishing.com.\n\nArcade Publishing® is a registered trademark of Skyhorse Publishing, Inc.®, a Delaware corporation.\n\nVisit
Meme	98%	a lot; that's great! It's a little awkward to ask, but we need your help. If you have already donated, we sincerely thank you. We're not salespeople, but we depend on donations averaging $14.76 and fewer than 1% of readers give. If you donate just $5.00, the price of your coffee, Catholic Online School could keep thriving. Thank
Malik Report	100%	check that allowed Dvorak to flick the puck over his shoulder...\n\nAbout The Malik Report\n\nThe Malik Report is a destination for all things Red Wings-related. I offer biased, perhaps unprofessional-at-times and verbose coverage of my favorite team, their prospects and developmental affiliates. I've joined the Kukla's Korner family with five years of blogging under
Porn	100%	make love to her. She returned the favor with an amazing*
Pokémon Fans	100%	We're a group of Pokémon fans dedicated to providing the best place on the Internet for discussing ideas and sharing fan-made content. Welcome! We're glad you're here.\n\nIn order to join our community we need you to create an account with us. Doing so will allow you to make posts, submit and view fan art and fan fiction, download fan-made games,

Table A.10: Examples of memorized sequences we collect from the Pile-dedupe. The prompt (prefix) is colored in brown. The numbers are the Accuracy (Eq. 3.5) of Pythia on memorizing the sequences, where 100% Accuracy means the true suffix can be fully reconstructed with greedy decoding.

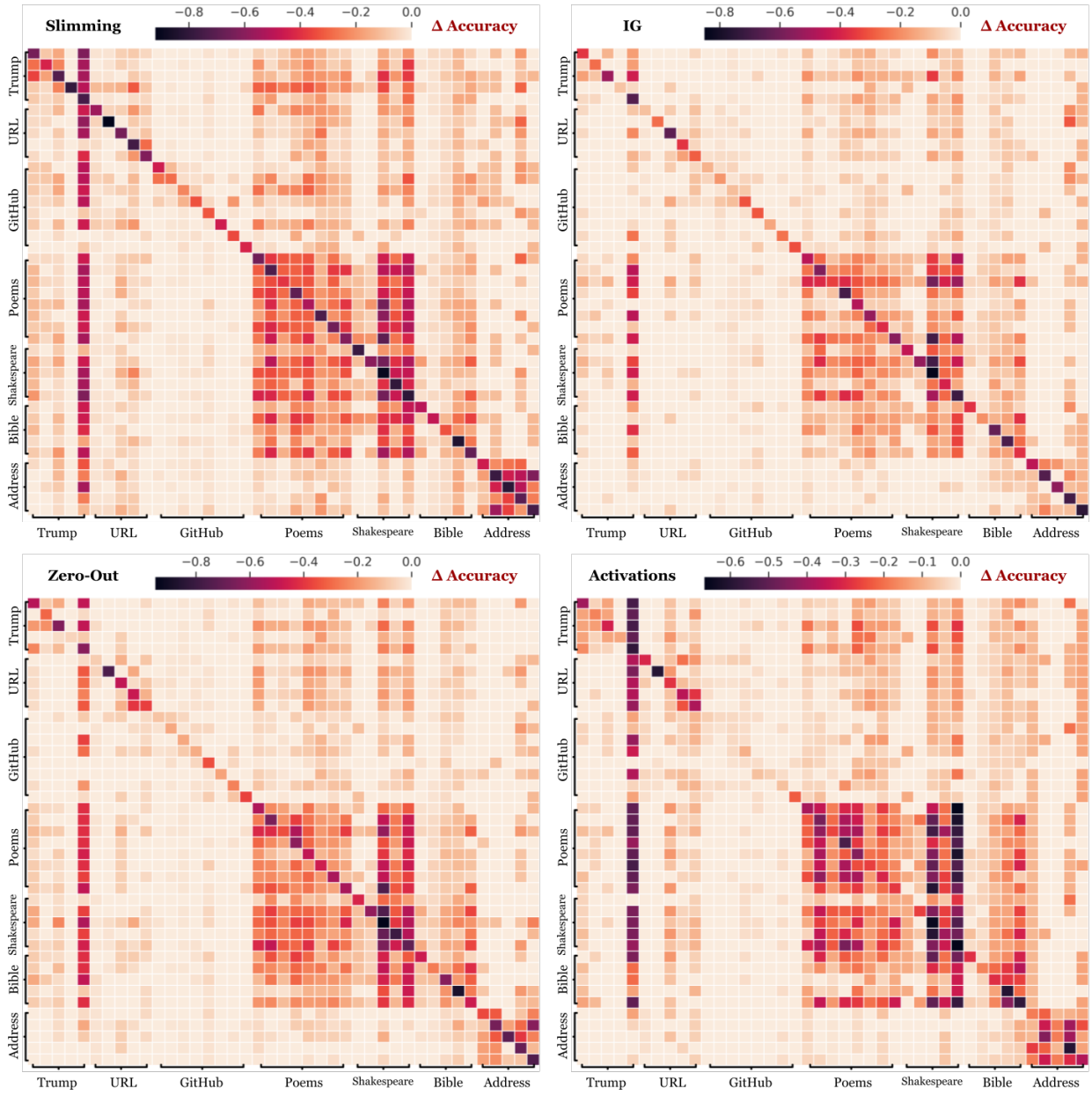


Figure A.2: Confusion matrices of localization methods on a subset of sequences memorized by GPT2-XL, where each row/column represents a sequence. Different methods show similar patterns of confusion.

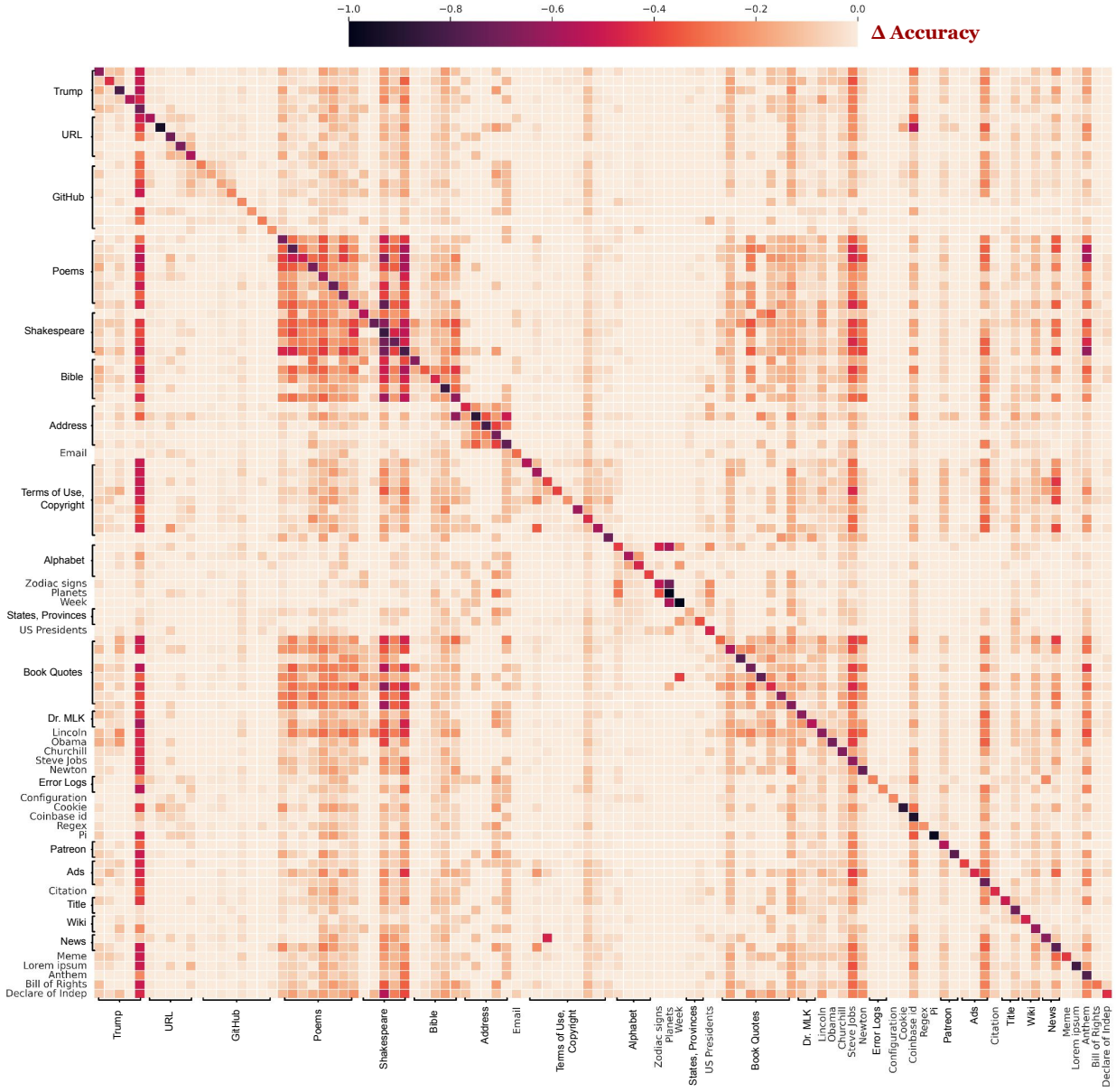


Figure A.3: Confusion matrix of HARD CONCRETE on the entire test set memorized by GPT2-XL. Each row shows how dropping out the predicted neurons (0.5%) on a target sequence changes the Accuracy of all sequences. HARD CONCRETE is unable to disentangle neurons of different quotes, including poems, Bible, books, and some famous people quotes. Also, it finds a small set of neurons responsible for memorizing both the order of Zodiac Signs and the order of Planets.

B Appendix for Chapter 4

B.1 Tasks and Templates

Table B.11 summarizes the 13 datasets we use in the paper, where we construct balanced test sets by randomly sampling 2000 examples in each task. We form the prompts by concatenating demonstrations in a randomly shuffled order. To avoid the recency bias [29], we keep shuffling the demonstrations until the last two have different labels. For minimally conservative templates (Section 4.4.1), Table B.12 compares the contrast sets we construct on Llama-2-7B. For our case study on Task069 and Task070, we sample 3 templates from Sclar et al. [27]. Figure B.4 compare the prompts of Task069 and Task070, which consist of an instruction followed by K templated demonstrations. Originally, the two tasks have $\sim 4\%$ of parallel examples. To make our task transfer challenging, we discard these overlapped examples.

Dataset	Task	# Classes
SST-2 [200]	Sentiment Analysis	2
Yelp-polarity [201]	Sentiment Analysis	2
BoolQ [202]	Yes/No QA	2
BoolQ Contrast Set [94]	Yes/No QA	2
QQP [203]	Paraphrase Identification	2
WiC [204]	Word Sense Disambiguation	2
RTE [203]	Natural Language Inference	2
MNLI [205]	Natural Language Inference	3
MedNLI [206]	NLI in Medical Domain	3
AGNews [201]	Topic Classification	4
ARC-Easy [207]	Multiple-Choice QA	4
Task069 [208, 95]	Abductive NLI	2
Task070 [208, 95]	Abductive NLI	2

Table B.11: Summary of all the datasets.

Task	Templates	Labels	Accuracy
SST-2	**T1** Review: {text}\nDo you think the review is positive or negative? {label}	negative/positive	50.6 ± 0.7
	T2 Review: {text}{space}\nDo you think the review is positive or negative? {label}	negative/positive	72.7 ± 6.1
BoolQ	**T1** Based on the following passage, {question}? {passage}\nAnswer: {label}	No/Yes	52.5 ± 2.0
	T2 Based on the following passage, {question}? {passage} Answer: {label}	No/Yes	66.7 ± 2.1
QQP	**T1** Are the questions "{sent1}" and "{sent2}" asking the same thing? {label}	no/yes	54.3 ± 1.1
	T2 Are the questions "{sent1}" and "{sent2}" asking the same thing? {label}	No/Yes	68.7 ± 4.1
AGNews	**T1** {text}\nIs this a piece of news regarding World, Sports, Business, or Technology? {label}	World/Sports/Business/Technology	43.9 ± 8.7
	T2 {text} Is this a piece of news regarding World, Sports, Business, or Technology? {label}	World/Sports/Business/Technology	88.5 ± 0.8

Table B.12: We construct minimally contrastive templates that only differ slightly (colored in red) but yield large differences in 4-shot ICL accuracy on Llama-2-7B. We report the average accuracy and standard deviation across 5 ICL runs with different demonstrations under the same template.

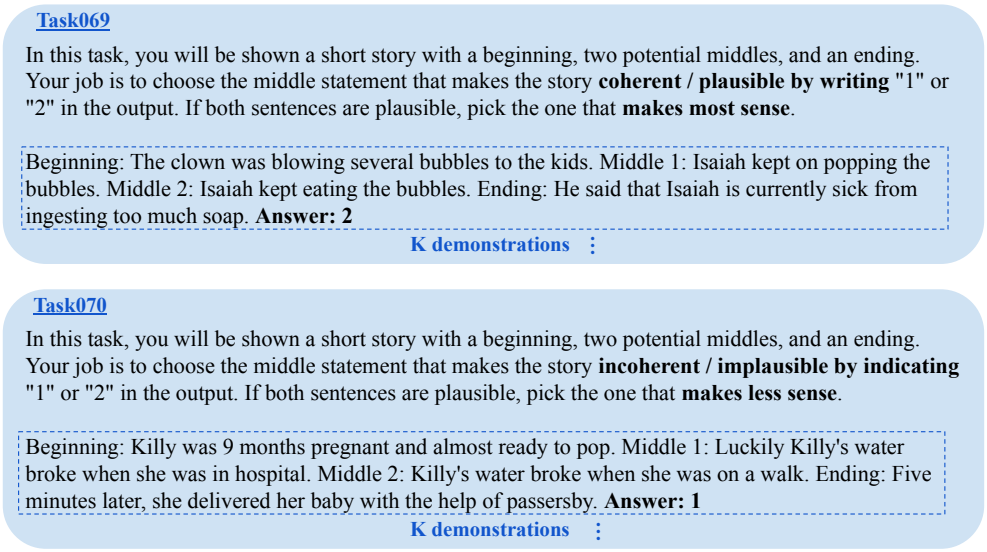
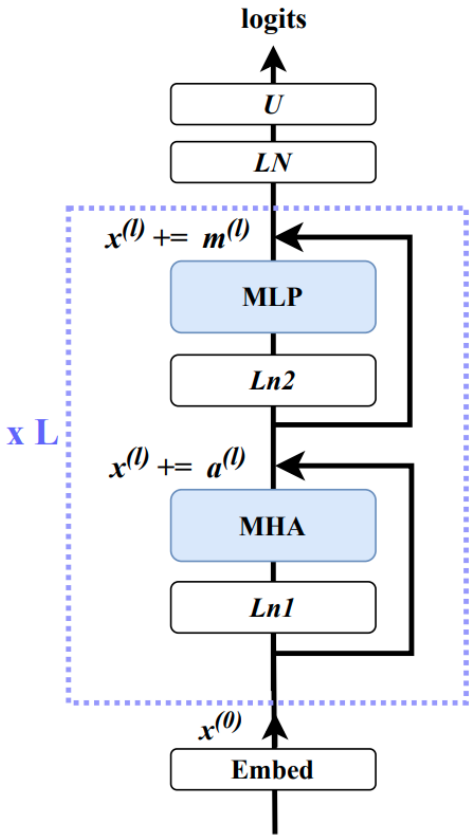


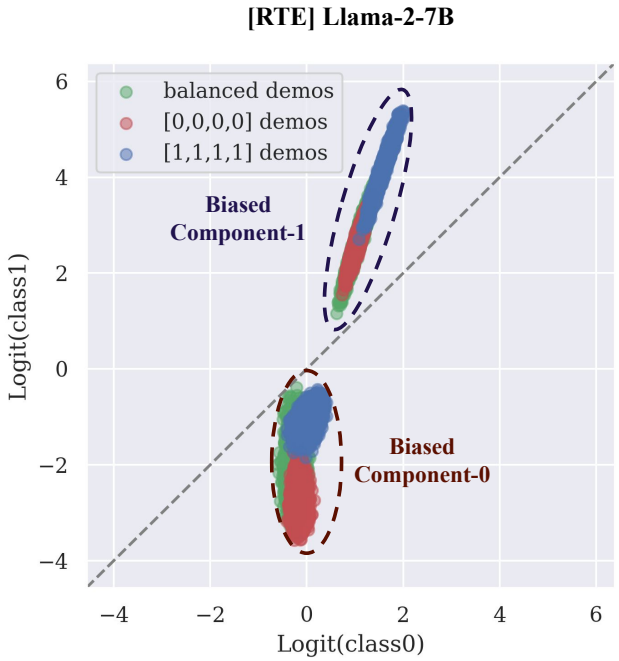
Figure B.4: Comparing the prompts of Task069 and Task070. We apply the templates of Sclar et al. [27] and prepend the task instructions before K demonstrations. We ensure that the two tasks do not have parallel examples to make the transfer experiment (Section 4.4.3) challenging.

B.2 Label-Biased Components

We say a component is label-biased when it always predicts a certain label on the entire test set (Section 4.3.2). In this section, we focus on the most biased components in binary classification tasks, i.e., the two components that have the largest value of $(\text{logit}_0 - \text{logit}_1)$ and $(\text{logit}_1 - \text{logit}_0)$, respectively, where $\text{logit}_0 \in \mathbb{R}$ and $\text{logit}_1 \in \mathbb{R}$ are the LLM output logits on the two classes. We name these two components as Biased Component-0 and Biased



(a) Transformer architecture in GPT2.



(b) Each dot represents an example in the test set. The two most biased components still insist on predicting the same label on the entire test set regardless of the labels of the demonstrations.

Figure B.5: Left: transformer architecture in GPT2-like models. Right: label-biased components on SST2.

Component-1, respectively. To understand how biased these two components are, we alter the choices of demonstrations and observe their behavior. Specifically, we consider three settings: demonstrations balanced in labels (green in Figure B.5b), demonstrations of all negative labels ($[0, 0, 0, 0]$; red), and demonstrations of all positive labels ($[1, 1, 1, 1]$; blue). We fix the template and sample 5 disjoint sets of demonstrations for each setting. Each dot in Figure B.5b shows the components' prediction on an example, and the x-axis and y-axis correspond to logit_0 and logit_1, respectively. A dot below the dashed diagonal line means the prediction on the example is class 0. We find that both Biased Component-0 and Biased Component-1 still insist on predicting a certain label on all examples, regardless of the labels in the prompts.

	SST2	BoolQ	QQP	WiC	RTE	MNLI	AGNews	ARC-Easy
Llama-2-7B	37.8	18.9	43.4	44.2	35.4	28.2	13.4	11.5
Llama-2-13B	32.6	18.7	37.2	39.3	32.1	26.0	12.4	13.1
Mistral-Instruct-7B	31.9	14.0	32.3	32.4	27.6	20.9	14.5	8.4
Llama-3-8B	38.3	21.4	42.4	40.0	36.9	28.1	15.8	15.0

Table B.13: We report the average percentage of label-biased components across 15 prompts for each task. A label-biased component always predicts the same label on the entire test set.

B.3 LayerNorms

Figure B.5a shows the transformer architecture in GPT2-like models. Because the layernorms inside each block are before MHA and MLP, known as Pre-LN, Eq. 4.1 has already taken $Ln1$ and $Ln2$ into account, and Eq. 4.6 only has the term for the final layernorm, $\text{LN}(\cdot)$.

Both Llama-2 and Mistral model families use RMSNorm [90], a layer normalization variant without centering and adding bias terms. Formally, let $x \in \mathbb{R}^d$ be the input, the root mean square norm $\text{LN}(x)$ is:

$$\text{LN}(x) = \frac{x}{\text{RMS}(x)} \odot \gamma, \tag{8}$$

$$\text{RMS}(x) = \sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}, \tag{9}$$

where $\gamma \in \mathbb{R}^d$ is the affine transform parameters and $\odot$ denotes element-wise multiplication.

B.4 Tests of Significance

We run one-tailed paired t-tests to test whether CompRW outperforms Calib+ significantly. In Table 4.4, we have the results of 15 prompts for each task and 8 tasks in total. For each model, we aggregate the 120 accuracy scores of CompRW and Calib+, respectively, and then calculate the p-values. Table B.14 shows that p-values < 0.05 in 8/8 setups, suggesting that CompRW performs significantly better than Calib+.

	Llama2-7B	Llama2-13B	Mistral-Ins-7B	Llama3-8B
$K = 12$	0.0010	0.0002	0.0470	0.0198
$K = 24$	0.0003	0.0001	0.0027	0.0245

Table B.14: The p-values < 0.05 in all 8 setups (4 LLMs, with $K = \{12, 24\}$ labeled examples), showing that CompRW performs significantly better than Calib+.

B.5 Do Good-Performing Components Exist in Randomly Initialized Models?

Ramanujan et al. [209] find that untrained subnetworks can perform on par with a ResNet-34 trained on ImageNet. Similarly, Zhang and Bowman [210] and Hewitt and Liang [211] show that representations of randomly initialized language models yield a strong baseline for probing tasks. In this section, we investigate (1) whether good-performing components still exist in a randomly initialized LLM, and (2) how CompRW method performs using component activations extracted from the randomly initialized LLM.

We run 4-shot ICL with 15 prompts and report the average accuracy and standard deviation. For CompRW, we use the same 4 demonstrations and 20 more examples for reweighting. Table B.15 shows that the best-performing component (Oracle-T1) in a randomly initialized Llama-2-7B still performs poorly on SST2 and ARC-Easy. While CompRW has substantial improvement over Full on the pretrained model, it has no effect on the randomly initialized model. We conclude that good-performing components do not exist in a randomly initialized LLM and our CompRW method relies on the pretrained component activations to perform well.

B.6 More Results on Transferability

In Section 4.4, we study the transferability of components across different choices of demonstrations and templates. Here, Table B.16 shows the full results on all LLMs and tasks. We

	SST2	ARC-Easy
Trained		
FULL	$75.8_{18.1}$	$57.5_{14.4}$
ORACLE-T1	$91.7_{0.9}$	$54.5_{10.1}$
COMPRW	$90.8_{1.8}$	$79.0_{1.6}$
Random		
FULL	$49.7_{0.7}$	$25.0_{0.5}$
ORACLE-T1	$55.2_{0.5}$	$26.7_{0.2}$
COMPRW	$51.4_{1.7}$	$25.0_{0.7}$

Table B.15: Comparing the ICL accuracy between pretrained (**Up**) and randomly initialized (**Down**) Llama-2-7B. The top-1 component (ORACLE-T1) and COMPRW perform near random on the untrained model.

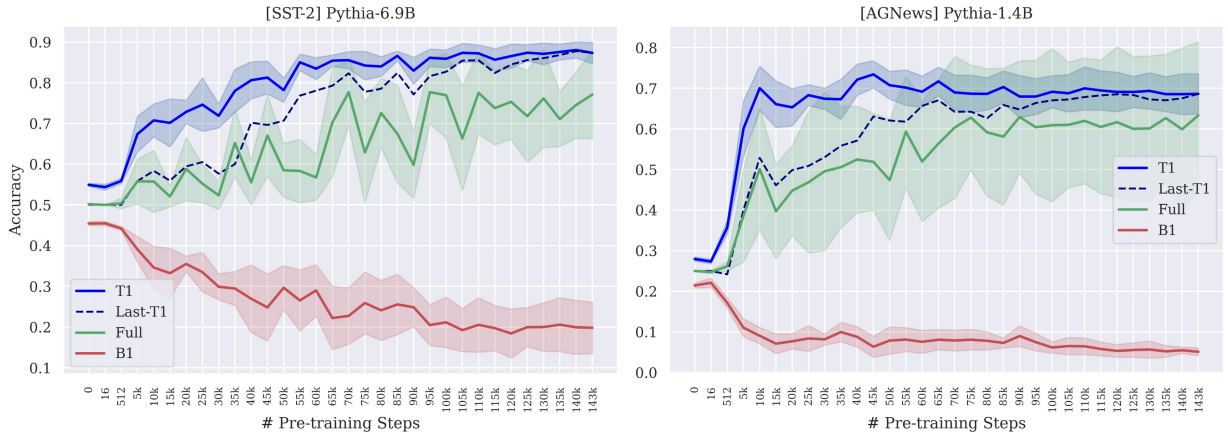


Figure B.6: 4-shot ICL accuracy on different pretraining checkpoints. We compare the full model (green) with the top-1 (solid blue) and bottom-1 (red) components. The dashed blue line tracks how the top-1 components of the last checkpoint (Last-T1) perform across time.

observe the same findings as Table 4.2: component accuracies agree well across randomly sampled demonstrations, but have much weaker agreements across randomly sampled templates. Because constructing minimally-contrastive templates requires non-trivial manual efforts, we only build contrast sets for 5 tasks on Llama-2-7B (shown in Table 4.2), where these tasks have the largest variances across templates.

	SST2	BoolQ	QQP	WiC	RTE	MNLI	AGNews	ARC
Correlation				***Llama-2-7B***				
(1) Demo	0.81	0.84	0.60	0.65	0.75	0.65	0.89	0.88
(2) Temp	0.40	0.16	0.03	0.15	0.19	0.09	0.68	0.44
IoU								
(1) Demo	0.36	0.74	0.27	0.21	0.53	0.24	0.63	0.70
(2) Temp	0.12	0.01	0.01	0.03	0.05	0.01	0.20	0.20
Correlation				***Llama-2-13B***				
(1) Demo	0.83	0.84	0.63	0.67	0.78	0.73	0.91	0.91
(2) Temp	0.57	0.30	0.09	0.19	0.28	0.16	0.76	0.55
IoU								
(1) Demo	0.26	0.71	0.31	0.18	0.46	0.39	0.55	0.65
(2) Temp	0.21	0.11	0.07	0.01	0.21	0.07	0.25	0.30
Correlation				***Mistral-Instruct-7B***				
(1) Demo	0.88	0.91	0.72	0.75	0.87	0.82	0.92	0.97
(2) Temp	0.58	0.44	0.19	0.26	0.40	0.30	0.77	0.60
IoU								
(1) Demo	0.39	0.59	0.27	0.29	0.50	0.45	0.68	0.80
(2) Temp	0.10	0.17	0.06	0.05	0.17	0.09	0.29	0.22
Correlation				***Llama-3-8B***				
(1) Demo	0.85	0.88	0.70	0.73	0.80	0.81	0.89	0.95
(2) Temp	0.55	0.39	0.26	0.25	0.31	0.23	0.67	0.52
IoU								
(1) Demo	0.42	0.56	0.28	0.25	0.46	0.52	0.65	0.68
(2) Temp	0.15	0.12	0.09	0.07	0.08	0.05	0.34	0.27

Table B.16: Full results of the average correlation and IoU between (1) two random sets of demonstrations and (2) two randomly sampled templates.

B.7 Pruning Good and Bad Components

Our method studies a component using its cached direct contribution to the output, whereas Michel, Levy, and Neubig [115] (*pruning*) zeroes out the activations of a component in the forward pass and thus indirectly changes the activations of other components in the upper layers. They consider a component important if pruning it causes large drops in task performance. In this section, we investigate the intersection between our method and

pruning.

First, we apply our decomposition to identify good and bad-performing components based on their ICL accuracy (3-shot for MNLI, 4-shot for other tasks). Second, we run ICL with pruning on Llama-2-7B, using the same 15 prompts in our main experiments for every task. We prune the top-50 components[3] and the bottom-50 components, respectively. Table B.17 compares the results with the full model without pruning. We find that pruning the top components (T50) greatly hurts the accuracy. On the contrary, pruning the bottom components (B50) only decreases the average accuracy on SST2 and RTE by 3.5%, and even slightly improves the ones on MNLI and AGNews. These findings may imply that our method and pruning interpret components in similar fashion.

	SST2	**RTE**	**MNLI**	**AGNews**
Full Model	**75.8** $_{18.1}$	**68.9** $_{3.2}$	34.4 $_{1.7}$	70.0 $_{19.9}$
Prune-T50	53.4 $_{8.8}$	57.8 $_{6.4}$	34.3 $_{3.2}$	26.8 $_{2.5}$
Prune-B50	72.3 $_{14.8}$	65.4 $_{5.8}$	**35.7** $_{4.0}$	**72.6** $_{15.7}$

Table B.17: Comparing the accuracies of the full Llama-2-7B model and pruning the top/bottom 50 components. We run 15 prompts for each task and report the average accuracy and standard deviation. We color the numbers red when there is a large drop in accuracy.

B.8 Training Details and Hyperparameters

For both CompRW and Calib+ methods, we train a linear layer on $\mathcal{D}_{\text{train}}$ with stochastic gradient descent. Because we do not have an additional dev set to tune the hyperparameters, we use the same hyperparameters on all the tasks and models and do early stopping based on the loss and accuracy on $\mathcal{D}_{\text{train}}$. Specifically, we set learning rate $= 0.05$ for both methods and $\lambda = 0.1$ for the L1 regularization term in CompRW. We run all our ICL experiments on

[3] $\sim 5\%$ of the total components

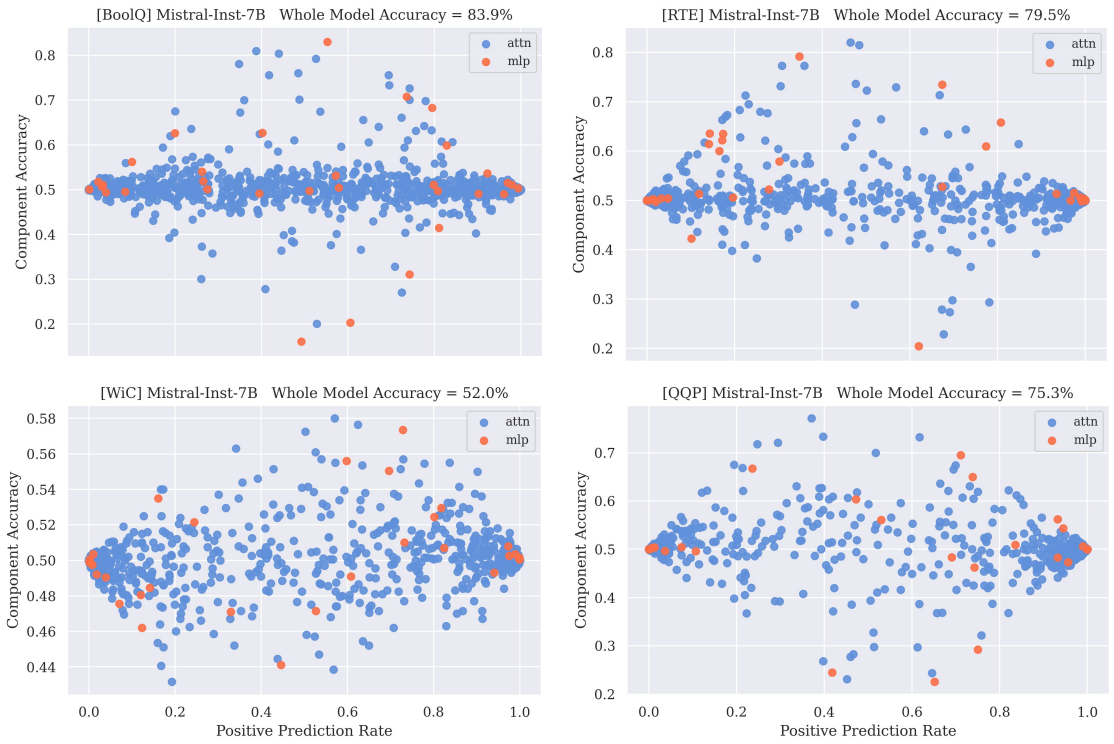


Figure B.7: Each dot represents a component (attention head: blue; MLP: orange) under 4-shot ICL on Mistral-Instruct-7B. The x-axis shows how often a component predicts label 1 across the test data of a binary task.

a single RTX A6000 GPU (48G). Both the component reweighting and calibration training processes can be run on a single i7 CPU within a minute.

B.9 Models

We use the model checkpoints on `Hugging Face`, `meta-llama/Llama-2-7b-hf`, `Llama-2-13b-hf`, `mistralai/Mistral-7B-Instruct-v0.1`, and `meta-llama/Meta-Llama-3-8B`.

C Appendix for Chapter 5

C.1 Implementation Details

We aim to answer whether different quantization methods make similar errors; thus, we follow the original implementations and do not unify their quantization configurations.

GPTQ. We use the `GPTQModel`[4] package for implementation. The calibration set consists of samples from English C4.[5] The quantization group size is 64 for 3-bit models and 128 for 4-bit models. We do not include the results of GPTQ3 with Llama-3-8B because we observe severe perplexity degradation with the latest `GPTQModel` package.

AWQ. We follow the original implementation[6] and use the released model checkpoints when available. The calibration set consists of samples from the Pile [62]. The quantization group size is 128 for both 3- and 4-bit models. Note that because AWQ applies scaling transformation, in our patching experiments Section 5.6.2, we also apply the same scaling factors on the *full-precision* models, which does not change their outputs but aligns their activations with the AWQ-quantized models.

NF. We use the official packages: `bitsandbytes`[7] for 4-bit models and `flute`[8] for 3-bit model. The quantization group size is 64 for both 3- and 4-bit models. We do not have the results of NF3 on Llama-3-70B due to kernel incompatibility.

EQAT. EQAT train on thousands of samples from RedPajama [212]. We use the officially released 3-bit model checkpoints,[9] which only cover the Llama3 models. The quantization

[4] `https://github.com/ModelCloud/GPTQModel`
[5] `en/c4-train.00001-of-01024.json.gz`
[6] `https://github.com/mit-han-lab/llm-awq`
[7] `https://pypi.org/project/bitsandbytes`
[8] `https://github.com/HanGuo97/flute`
[9] `https://github.com/OpenGVLab/EfficientQAT`

Model	Method	Fineweb	Wiki
Qwen2.5-7B	16 bits	11.06	6.85
	AWQ4	11.35	7.09
	NF4	11.37	7.10
	GPTQ4	11.33	7.16
	AWQ3	12.77	8.20
	NF3	13.47	8.51
	GPTQ3	12.26	8.11
Llama3-8B	16 bits	9.57	6.14
	AWQ4	10.13	6.56
	NF4	10.11	6.56
	GPTQ4	10.12	6.86
	AWQ3	12.13	8.19
	NF3	12.45	8.44
	EQAT3	10.77	7.10
Mistral-12B	16 bits	8.67	5.75
	AWQ4	9.09	6.01
	NF4	9.13	6.06
	GPTQ4	9.02	6.04
	AWQ3	10.32	7.03
	NF3	12.48	8.75
	GPTQ3	10.44	7.20
Llama3-70B	16 bits	7.15	2.86
	AWQ4	7.42	3.27
	NF4	7.88	3.22
	GPTQ4	7.52	3.58
	GPTQ3	9.64	6.51
	AWQ3	8.40	4.74

Table C.18: The perplexity of different quantization methods on the 10k FineWeb dataset and WikiText. Our FineWeb examples have shorter sequence lengths than WikiText (512 vs. 2048), leading to higher perplexity.

group size is 128.

Perplexity. We report the perplexity of different 3- and 4-bit quantization methods in Table C.18. We follow the standard implementation of WikiText perplexity that sets the sequence length to 2048. The sequence length of FineWeb is 512.

Data. We use FineWeb for our main experiments because it is a diverse, cleaned dataset, and none of the quantization methods considered above sample calibration data from it, making FineWeb an unbiased corpus for our study. We also explore more technical domains, Arxiv and OpenWebMath [213, 214]. Our findings that the quantization errors of different methods are strongly correlated (Section 5.4) also hold on these domains. Specifically, the average correlation across 8 pairs of methods on Llama3-70B is 0.69 on ArXiv and 0.89 on OpenWebMath.

Definition of Error. We also explore an alternative definition of quantization errors based on KL divergence. Let p_θ, the probability distribution of the full-precision model, be the true distribution. We can define errors as the KL divergence between the quantized and full-precision model, $\mathrm{KL}\left(p_\theta \,\|\, p_{\tilde{\theta}}\right)$. The results of these two definitions are highly correlated ($\rho \geq 0.97$). Thus, we adopt Eq. 5.4 for its lower computation costs.

C.2 The Construction of the $\mathcal{D}_{\text{large}}$

To ensure the diversity of $\mathcal{D}_{\text{large}}$ examples, we apply aggressive data deduplication based on Jaccard similarity [215], filtering out examples with 5-gram Jaccard similarity > 0.1. We create a separate $\mathcal{D}_{\text{large}}$ for each quantization method and observe similar findings across them. Besides, the $\mathcal{D}_{\text{large}}$ sets of different methods under the same LLMs often have overlapping examples, with the average Jaccard similarity $\frac{|A \cap B|}{|A \cup B|} = 0.30, 0.49, 0.66, 0.66$ on Qwen2.5-7B, Meta-Llama-3-8B, Mistral-12B, and Meta-Llama-3-70B, respectively. Therefore, in the main content, we always use the one derived from AWQ3 as $\mathcal{D}_{\text{large}}$ for simplicity.

C.3 Correlations Between Residual Magnitudes and Quantization Errors

We present the correlations between the quantization errors $\mathcal{E}(\mathcal{D}; \tilde{\theta})$ and the residual magnitudes $\mathbb{S}(\mathcal{D}; \theta; l)$ across layers of Qwen2.5-7B, Llama3-8B, Mistral-12B, and Llama3-70B in

Fig. C.10, C.11, C.12, C.13, respectively. We find that different models and methods all show that the correlations become increasingly negative in deeper layers.

C.4 RMSNorm and Outliers

First, we select the serverest outlier dimensions in $r^{(l)}$, namely, the most-activated dimensions in $r^{(l)}$ that have the highest average absolute activation over the 10k FineWeb dataset. Next, we rank the entries in RMSNorm weights γ by their absolute values. We compare the median of γ with those entries corresponding to the outlier dimensions. Table C.21 shows that in the upper half layers of Llama3-70B, all outlier dimensions have much smaller weights than the median. On the other hand, in Mistral-12B, the most severe outliers have small weights (rank 1 or 2) across layers, but other outlier dimensions sometimes have much larger weights than the median. These results show that RMSNorm weights suppress the largest outlier dimensions but sometimes also promote other outlier dimensions. In Fig. C.8 and Fig. C.9, we show the kurtosis values before and after RMSNorm, respectively. Comparing the scale of the kurtosis values (y-axis) of the two figures, we find that the scale after RMSNorm is much smaller, implying that RMSNorm in the upper layers suppresses outliers overall.

Similar to Wei et al. [147], we find that the LayerNorm parameters modulate activation outliers; however, we observe mostly opposite effects, likely due to the different architectural designs (pre-LN vs. post-LN [216]).

C.5 More Early Exiting Results

We apply early exiting on $z^{(l)}$ and $r^{(l)}$, the hidden states after the residual operations (see Eq. 5.6), across layers. Figure C.14, C.15, C.16, and C.17 shows the results on Qwen2.5-7B, Llama3-8B, Mistral-12B, and Llama3-70B, respectively, where we use AWQ3 to produce the quantized models. All the LLMs show that the NLLs of $\mathcal{D}_{\text{large}}$ (purple) decrease dramatically in the later layers, suggesting that $\mathcal{D}_{\text{large}}$ relies more on the late layers to decode the representations than $\mathcal{D}_{\text{ctrl}}$ (green).

Model	(Method1, Method2)	$\rho(\mathcal{E}_1, \mathcal{E}_2)$
Qwen2.5-7B	(AWQ4 , NF4)	0.52
	(AWQ4 , GPTQ4)	0.66
	(AWQ4 , NF3)	0.61
	(AWQ4 , GPTQ3)	0.65
	(AWQ3 , NF3)	0.78
	(AWQ3 , GPTQ3)	0.84
	(NF4 , GPTQ4)	0.57
	(NF4 , AWQ3)	0.57
	(NF4 , GPTQ3)	0.55
	(NF3 , GPTQ3)	0.75
	(GPTQ4, AWQ3)	0.75
	(GPTQ4, NF3)	0.70
Llama3-8B	(AWQ4 , NF4)	0.95
	(AWQ4 , GPTQ4)	0.96
	(AWQ4 , NF3)	0.88
	(AWQ4 , EQAT3)	0.92
	(AWQ3 , NF3)	0.98
	(AWQ3 , EQAT3)	0.97
	(NF4 , GPTQ4)	0.95
	(NF4 , AWQ3)	0.87
	(NF4 , EQAT3)	0.90
	(NF3 , EQAT3)	0.96
	(GPTQ4, AWQ3)	0.90
	(GPTQ4, NF3)	0.90
	(GPTQ4, EQAT3)	0.93

Table C.19: The quantization errors of different methods are strongly correlated on the FineWeb. The numbers in the method names denote 3 or 4-bit precision.

Variant Pairs	$\boldsymbol{\rho}$
(damp-0.005-sort, damp-0.01-sort)	0.95
(damp-0.005-sort, damp-0.02-sort)	0.95
(damp-0.01 -sort, damp-0.02-sort)	0.95
(damp-0.005-sort, damp-0.01-no_sort)	0.92
(damp-0.01 -sort, damp-0.01-no_sort)	0.92
(damp-0.02 -sort, damp-0.01-no_sort)	0.92

Table C.20: Quantization error correlations of 3-bit Qwen2.5-7B GPTQ variants with different damping coefficients and activation-sorting settings. All pairs are strongly correlated.

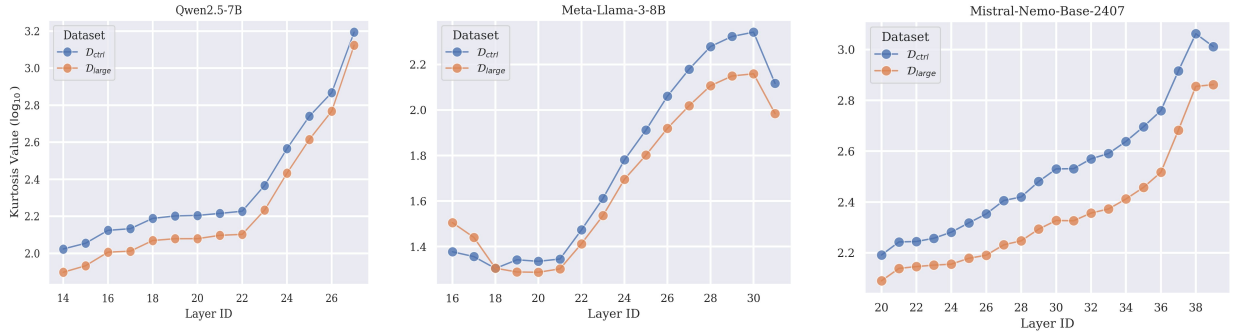


Figure C.8: Pearson's kurtosis values of the residual states $r^{(l)}$ from different full-precision LLMs (log scale). The large-error set $\mathcal{D}_{\text{large}}$ shows lower kurtosis values than the control set $\mathcal{D}_{\text{ctrl}}$ across layers, indicating that $\mathcal{D}_{\text{large}}$ contains fewer or less extreme outliers in the residual states.

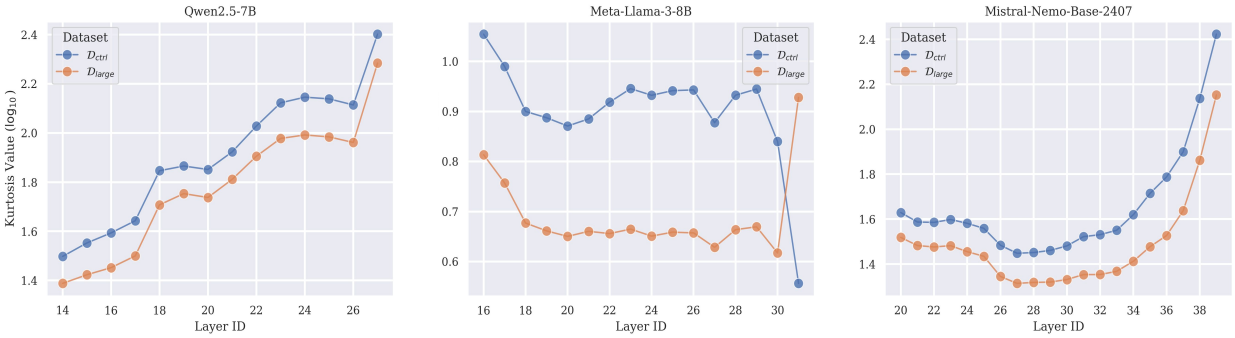


Figure C.9: Pearson's kurtosis values of the post-RMSNorm hidden states $h^{(l)}$ from different full-precision LLMs (log scale). The large-error set $\mathcal{D}_{\text{large}}$ shows lower kurtosis values than the control set $\mathcal{D}_{\text{ctrl}}$, except for the last layer of Llama3-8B, indicating that overall $\mathcal{D}_{\text{large}}$ contains fewer or less extreme outliers in the hidden states after RMSNorm. In addition, compared with Fig. C.8 (before RMSNorm), the scale of the kurtosis values in this figure is much smaller (y-axis), implying that RMSNorm in the upper layers tends to suppress outliers.

C.6 Full Results on PopQA

Fig. C.19 and C.20 show the accuracy degradation and absolute accuracy values of different models, respectively. Fig. C.21 shows the popularity histogram.

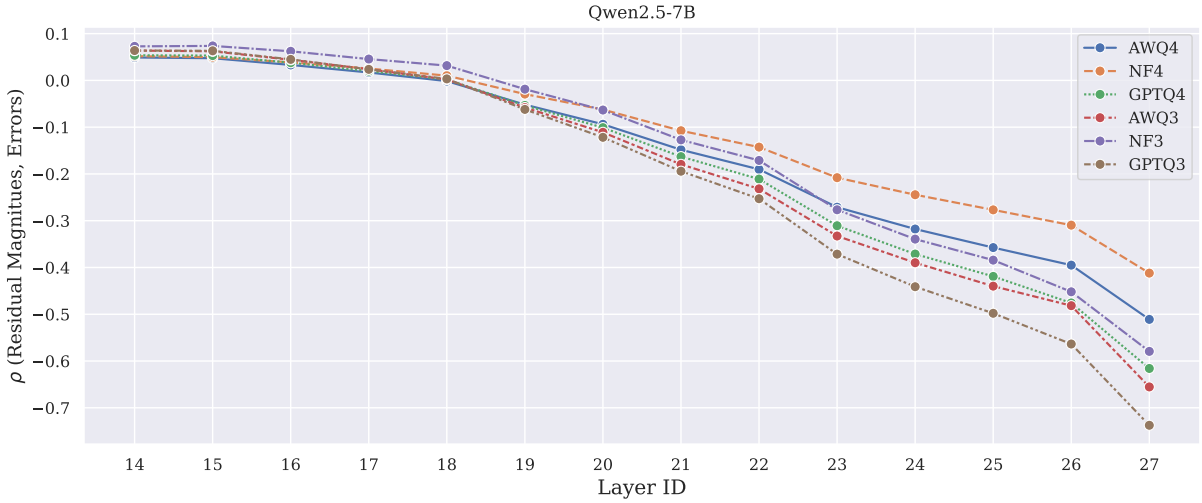


Figure C.10: Correlations between the quantization errors $\mathcal{E}(\mathcal{D};\tilde{\theta})$ and the residual magnitudes $\mathbb{S}(\mathcal{D};\theta;l)$ across the upper half layers of Qwen2.5-7B. All methods show negative correlations in the last few layers.

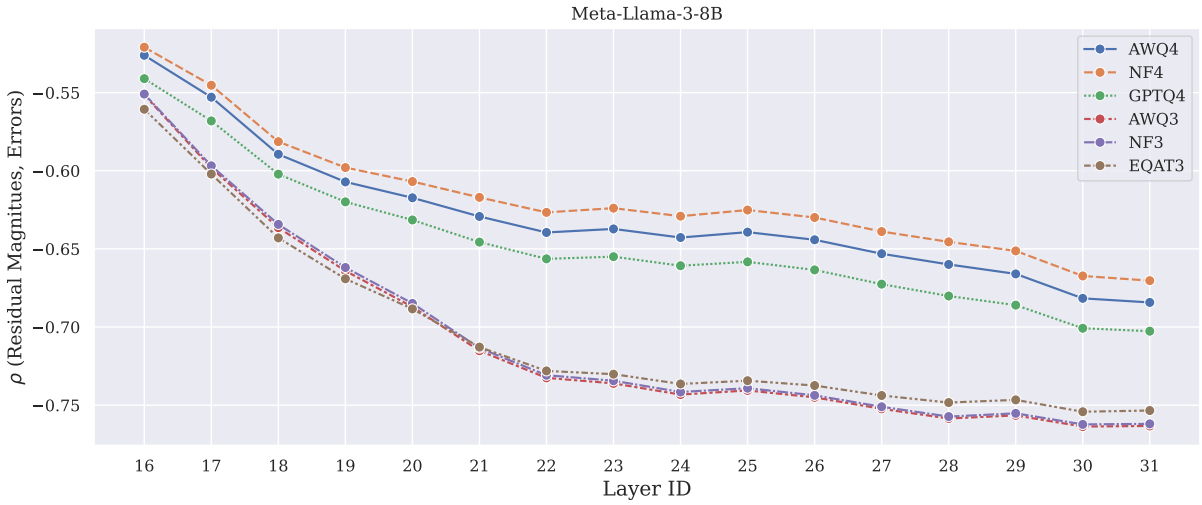


Figure C.11: Correlations between the quantization errors $\mathcal{E}(\mathcal{D};\tilde{\theta})$ and the residual magnitudes $\mathbb{S}(\mathcal{D};\theta;l)$ across the upper half layers of Llama3-8B. All methods show negative correlations in the last few layers.

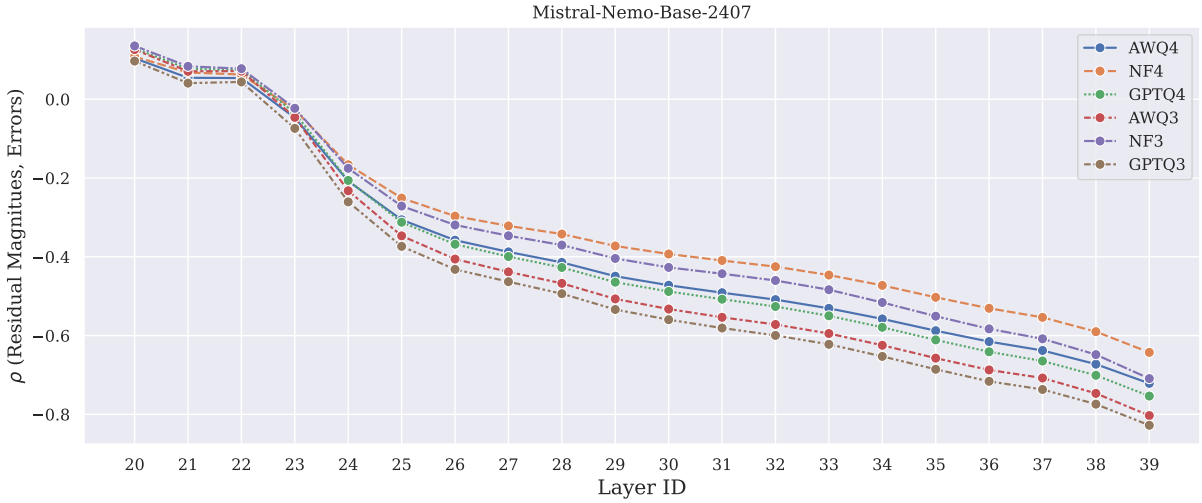


Figure C.12: Correlations between the quantization errors $\mathcal{E}(\mathcal{D};\tilde{\theta})$ and the residual magnitudes $\mathbb{S}(\mathcal{D};\theta;l)$ across the upper half layers of Mistral-12B. All methods show negative correlations in the last few layers.

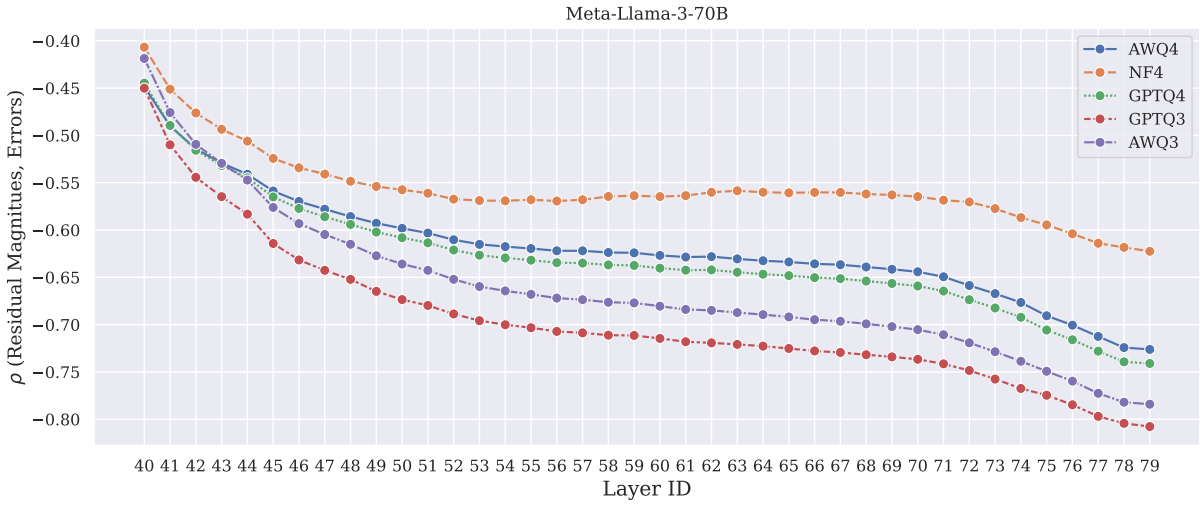


Figure C.13: Correlations between the quantization errors $\mathcal{E}(\mathcal{D};\tilde{\theta})$ and the residual magnitudes $\mathbb{S}(\mathcal{D};\theta;l)$ across the upper half layers of Llama3-70B. All methods show negative correlations in the last few layers.

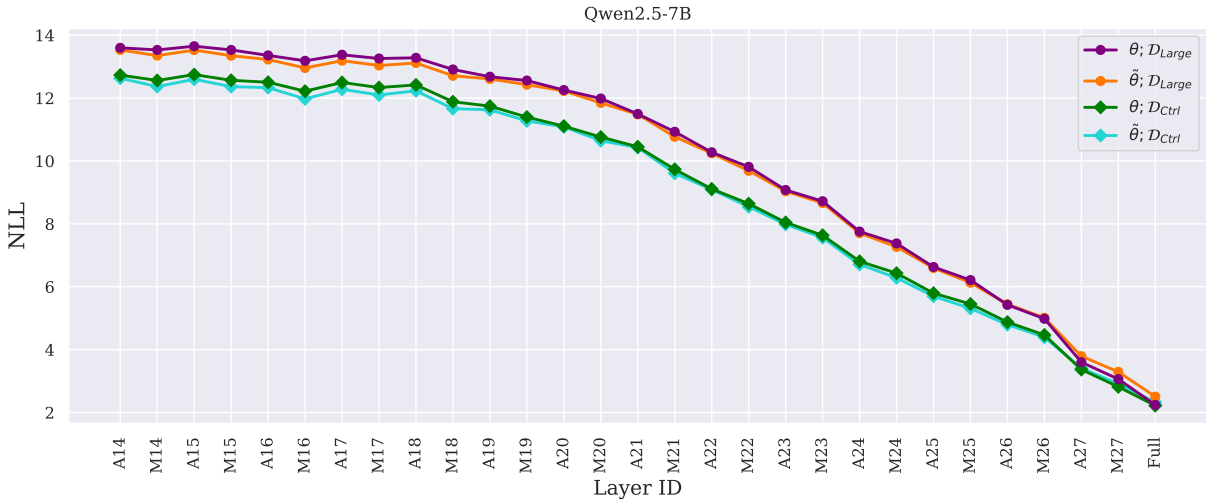


Figure C.14: Early Exiting results on Qwen2.5-7B. We decode the $z^{(l)}$ (A#) and $r^{(l)}$ (M#) across layers.

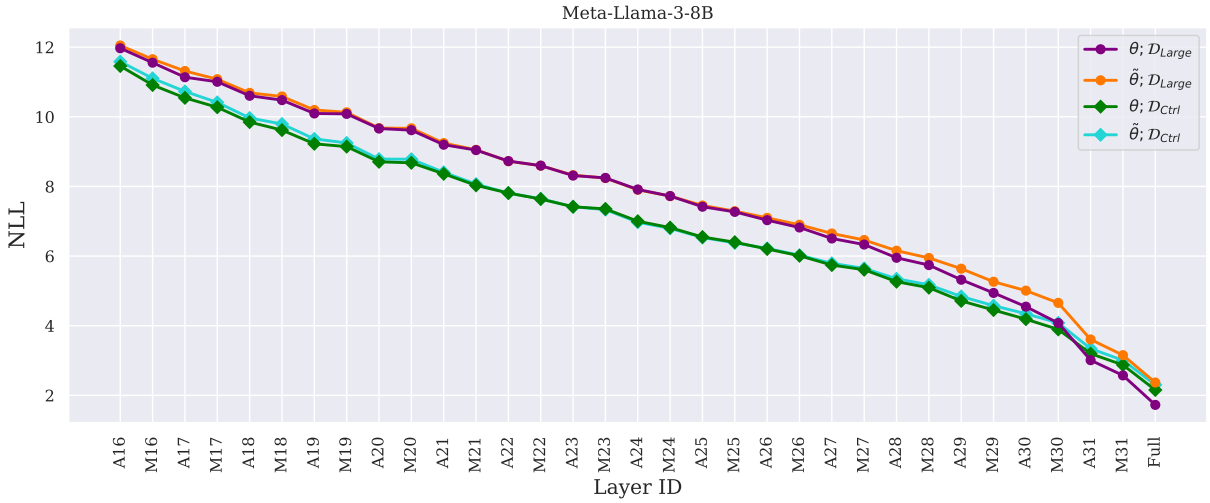


Figure C.15: Early Exiting results on Llama3-8B. We decode the inputs to $z^{(l)}$ (A#) and $r^{(l)}$ (M#) across layers.

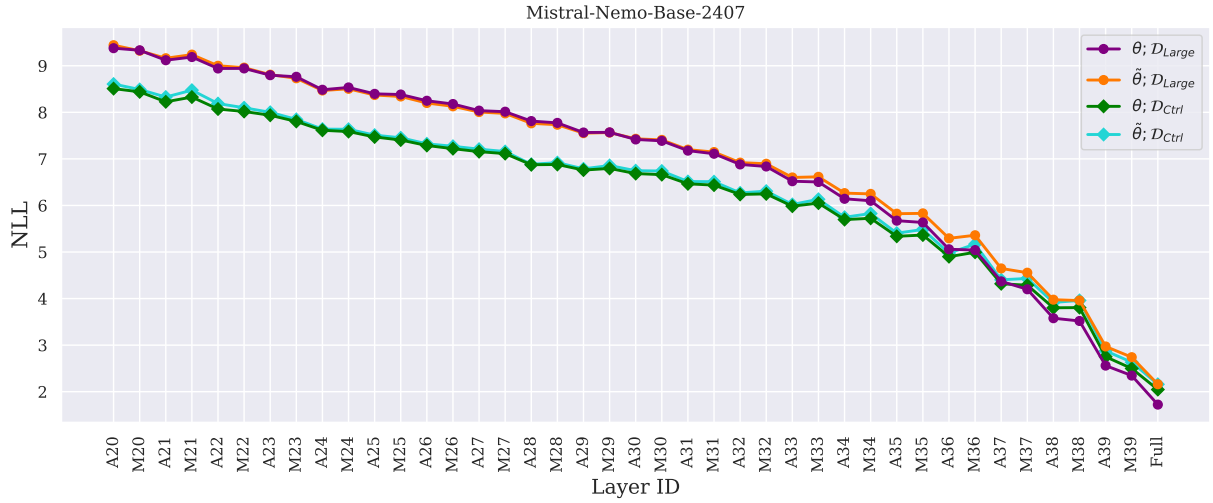


Figure C.16: Early Exiting results on Mistral-12B. We decode the $z^{(l)}$ (A#) and $r^{(l)}$ (M#) across layers.

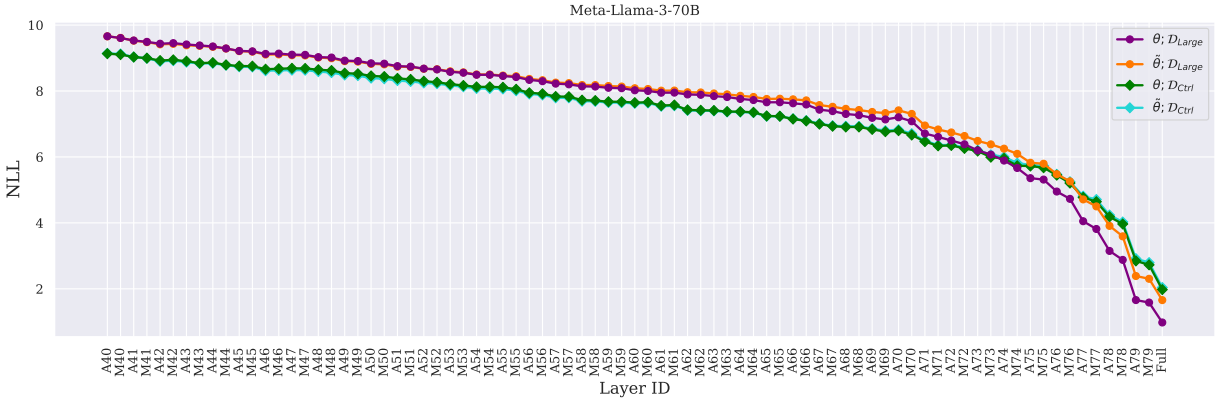


Figure C.17: Early Exiting results on Llama3-70B. We decode $z^{(l)}$ (A#) and $r^{(l)}$ (M#) across layers.

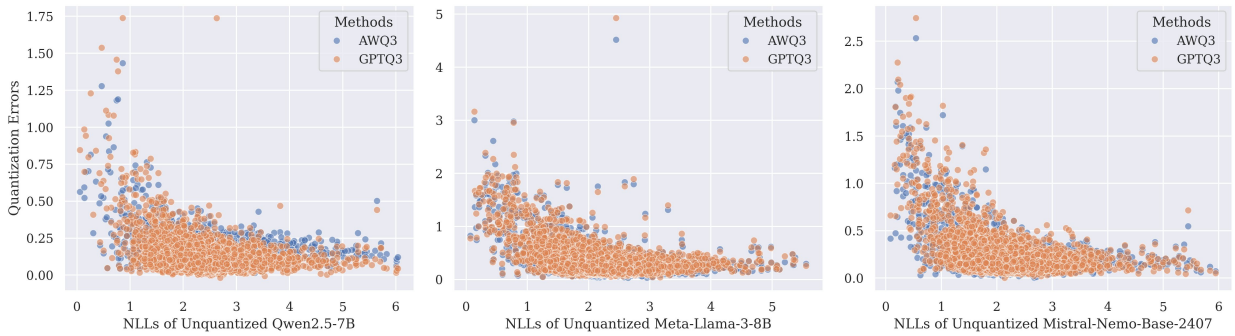


Figure C.18: NLLs before quantization vs. quantization errors on different LLMs. Each dot represents a sample from FineWeb. In general, the low-frequency data (x-axis values ≥ 4) have smaller errors. On the other hand, many samples that suffer from large quantization errors have small NLLs before quantization.

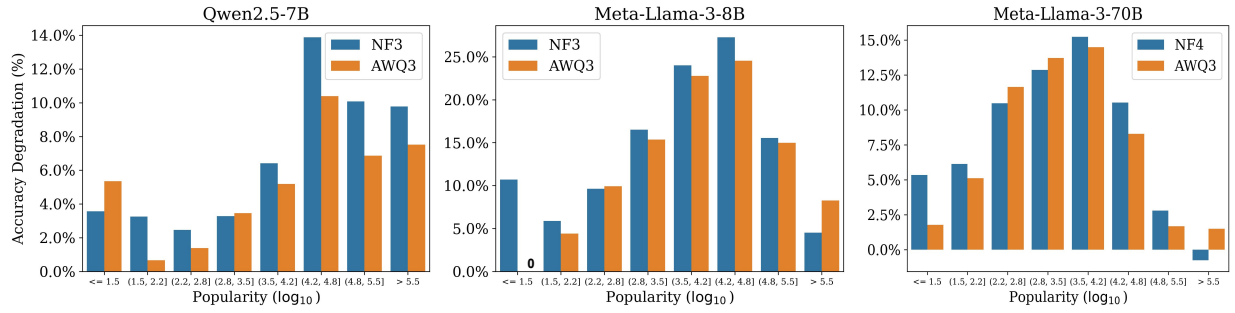


Figure C.19: Accuracy degradation of different models on PopQA.

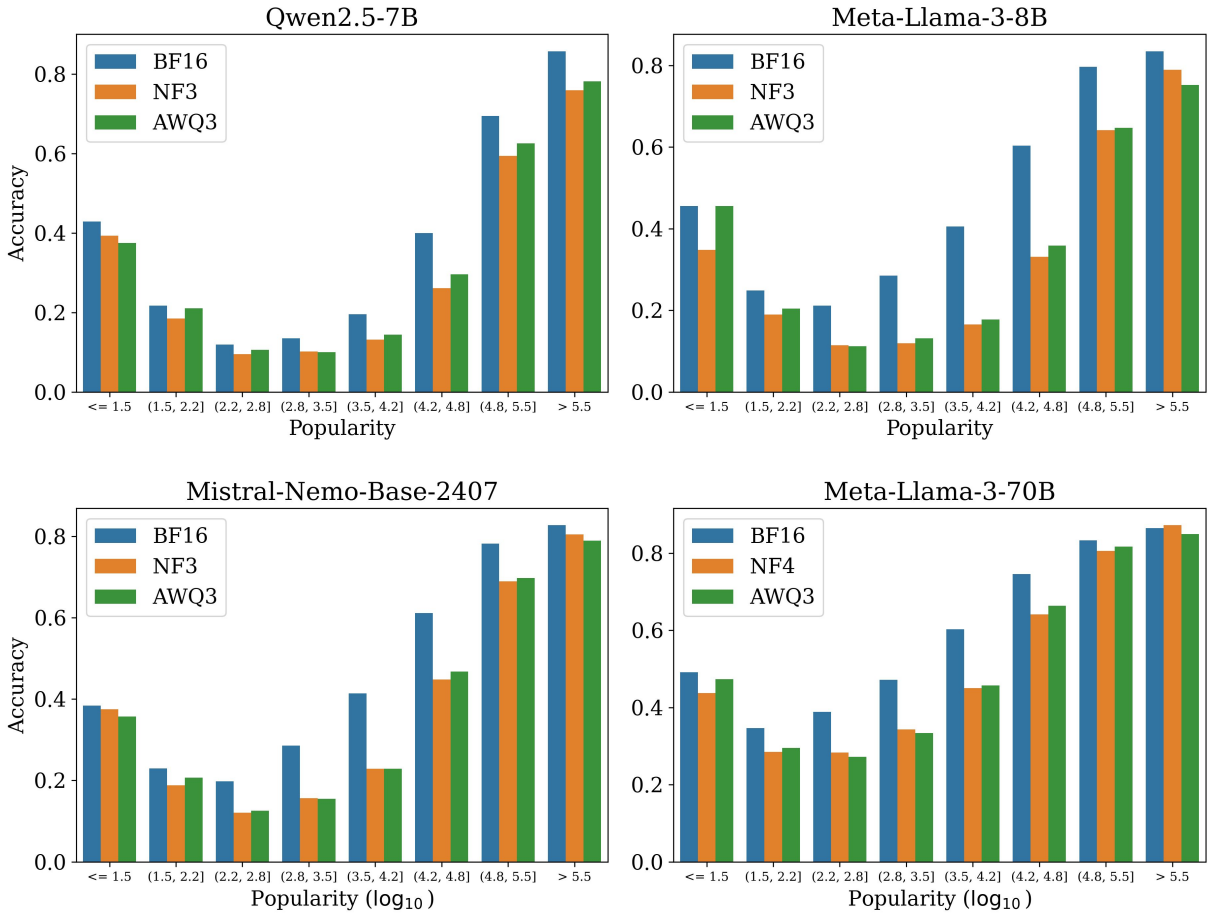


Figure C.20: Absolute accuracies of different models on PopQA.

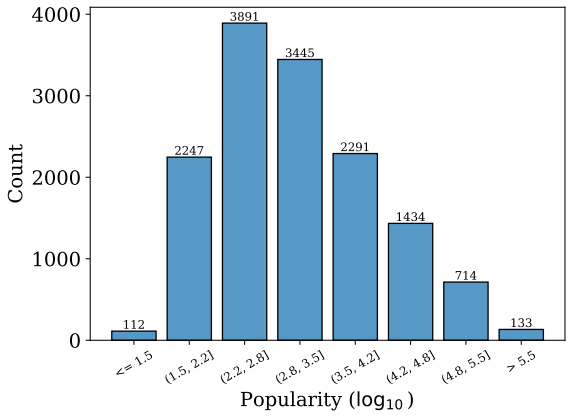


Figure C.21: The histogram of the PopQA dataset. We partition the dataset into 8 buckets based on the sample popularity and ensure that each partition contains at least 100 samples.

	Layer	Rank of Outliers' $\lvert\gamma\rvert$	Median $\lvert\gamma\rvert$	Outliers' $\lvert\gamma\rvert$
Llama3-70B	40	[2, 9, 1, 10, 13]	0.1553	[0.0195, 0.0376, 0.0186, 0.0386, 0.0413]
	41	[2, 8, 1, 9, 10]	0.1582	[0.0170, 0.0312, 0.0142, 0.0320, 0.0330]
	42	[5, 1, 7, 10, 8]	0.1602	[0.0215, 0.0128, 0.0271, 0.0312, 0.0292]
	43	[1, 6, 2, 8, 9]	0.1621	[0.0117, 0.0275, 0.0154, 0.0315, 0.0317]
	44	[1, 7, 2, 9, 8]	0.1631	[0.0171, 0.0369, 0.0209, 0.0396, 0.0374]
	45	[1, 7, 10, 2, 7]	0.1660	[0.0130, 0.0383, 0.0420, 0.0209, 0.0383]
	46	[1, 7, 14, 12, 2]	0.1689	[0.0088, 0.0342, 0.0405, 0.0396, 0.0223]
	47	[1, 7, 8, 3, 14]	0.1699	[0.0090, 0.0320, 0.0364, 0.0206, 0.0396]
	48	[1, 6, 8, 2, 9]	0.1709	[0.0098, 0.0359, 0.0420, 0.0247, 0.0427]
	49	[1, 5, 12, 3, 12]	0.1738	[0.0128, 0.0354, 0.0476, 0.0297, 0.0476]
	50	[1, 6, 13, 3, 8]	0.1758	[0.0100, 0.0327, 0.0437, 0.0273, 0.0369]
	51	[1, 6, 11, 3, 8]	0.1777	[0.0117, 0.0369, 0.0471, 0.0312, 0.0427]
	52	[1, 7, 8, 3, 19]	0.1787	[0.0126, 0.0422, 0.0454, 0.0315, 0.0591]
	53	[1, 6, 11, 3, 14]	0.1816	[0.0136, 0.0337, 0.0422, 0.0255, 0.0461]
	54	[1, 6, 9, 2, 11]	0.1826	[0.0138, 0.0293, 0.0376, 0.0248, 0.0388]
	55	[1, 5, 9, 3, 11]	0.1846	[0.0110, 0.0330, 0.0427, 0.0277, 0.0454]
	56	[1, 6, 9, 3, 13]	0.1865	[0.0138, 0.0378, 0.0439, 0.0297, 0.0471]
	57	[1, 6, 9, 15, 3]	0.1885	[0.0124, 0.0344, 0.0408, 0.0459, 0.0282]
	58	[1, 6, 9, 15, 14]	0.1904	[0.0121, 0.0322, 0.0388, 0.0457, 0.0437]
	59	[1, 5, 12, 23, 14]	0.1934	[0.0206, 0.0420, 0.0559, 0.0684, 0.0583]
	60	[1, 6, 12, 27, 14]	0.1953	[0.0197, 0.0461, 0.0547, 0.0776, 0.0554]
	61	[1, 7, 12, 24, 14]	0.1982	[0.0177, 0.0459, 0.0532, 0.0693, 0.0562]
	62	[1, 6, 22, 14, 20]	0.2002	[0.0160, 0.0435, 0.0659, 0.0579, 0.0635]
	63	[1, 6, 12, 22, 15]	0.2031	[0.0199, 0.0530, 0.0640, 0.0781, 0.0679]
	64	[1, 8, 26, 18, 16]	0.2070	[0.0272, 0.0608, 0.0903, 0.0757, 0.0723]
	65	[1, 2, 9, 8, 26]	0.2109	[0.0225, 0.0403, 0.0635, 0.0603, 0.0903]
	66	[1, 9, 5, 6, 22]	0.2119	[0.0259, 0.0591, 0.0496, 0.0520, 0.0825]
	67	[1, 10, 6, 26, 7]	0.2158	[0.0266, 0.0625, 0.0537, 0.0918, 0.0559]
	68	[1, 9, 5, 27, 6]	0.2227	[0.0293, 0.0762, 0.0659, 0.1079, 0.0674]
	69	[1, 9, 2, 36, 7]	0.2266	[0.0294, 0.0732, 0.0425, 0.1201, 0.0688]
	70	[1, 12, 28, 6, 7]	0.2285	[0.0311, 0.0737, 0.1167, 0.0654, 0.0669]
	71	[1, 28, 16, 6, 7]	0.2344	[0.0312, 0.1328, 0.1006, 0.0771, 0.0776]
	72	[1, 33, 11, 9, 3]	0.2441	[0.0430, 0.1650, 0.0947, 0.0918, 0.0649]
	73	[1, 36, 9, 23, 3]	0.2520	[0.0483, 0.1729, 0.1089, 0.1455, 0.0723]
	74	[1, 36, 8, 18, 3]	0.2598	[0.0459, 0.1846, 0.1060, 0.1367, 0.0767]
	75	[1, 44, 11, 10, 3]	0.2676	[0.0796, 0.2139, 0.1250, 0.1230, 0.0923]
	76	[1, 43, 3, 15, 15]	0.3047	[0.0486, 0.2217, 0.0981, 0.1533, 0.1533]
	77	[1, 2, 52, 4, 5]	0.2891	[0.0554, 0.0820, 0.2080, 0.0869, 0.0913]
	78	[1, 4, 2, 20, 6]	0.2812	[0.0781, 0.0972, 0.0923, 0.1562, 0.1138]
	79	[9, 4, 1, 3, 13]	0.2236	[0.0688, 0.0500, 0.0309, 0.0454, 0.0747]

Table C.21: RMSNorm weights γ ranked in descending order by absolute value. We compare the median weight with those of the outlier dimensions, listing ranks and weight values for the 1st through 5th severest outliers. In Llama3-70B, all outlier dimensions have weights far below the median, suggesting that the model uses RMSNorm weights to suppress outliers. (Continued in Table C.22.)

	Layer	Rank of Outliers' $\lvert\gamma\rvert$	Median $\lvert\gamma\rvert$	Outliers' $\lvert\gamma\rvert$
Mistral-12B	20	[1, 6, 22, 376, 4000]	0.8594	[0.001640, 0.137695, 0.330078, 0.648438, 0.902344]
	21	[1, 5, 22, 289, 2463]	0.9062	[0.001648, 0.113770, 0.324219, 0.644531, 0.906250]
	22	[1, 2, 22, 283, 2402]	0.9453	[0.002487, 0.044434, 0.369141, 0.687500, 0.941406]
	23	[1, 2, 22, 308, 3740]	0.9570	[0.001511, 0.089355, 0.410156, 0.726562, 0.988281]
	24	[2, 1, 261, 21, 4599]	1.0078	[0.080078, 0.001457, 0.730469, 0.365234, 1.062500]
	25	[2, 1, 167, 20, 1615]	1.0469	[0.102539, 0.017334, 0.691406, 0.359375, 1.023438]
	26	[2, 1, 184, 21, 1145]	1.0859	[0.092285, 0.001183, 0.722656, 0.390625, 1.039062]
	27	[5, 1, 151, 21, 840]	1.1172	[0.140625, 0.025146, 0.722656, 0.410156, 1.046875]
	28	[2, 1, 150, 22, 792]	1.1484	[0.094727, 0.005585, 0.757812, 0.429688, 1.070312]
	29	[2, 1, 175, 21, 945]	1.1875	[0.121582, 0.016113, 0.785156, 0.453125, 1.132812]
	30	[2, 1, 181, 19, 1080]	1.2266	[0.116211, 0.009583, 0.855469, 0.458984, 1.187500]
	31	[2, 1, 211, 3486, 21]	1.2578	[0.129883, 0.000584, 0.933594, 1.273438, 0.488281]
	32	[2, 1, 212, 4982, 18]	1.2812	[0.153320, 0.000881, 1.000000, 1.335938, 0.458984]
	33	[2, 1, 200, 5098, 18]	1.2812	[0.154297, 0.001419, 1.054688, 1.421875, 0.542969]
	34	[1, 2, 5111, 666, 21]	1.3125	[0.000793, 0.210938, 1.640625, 1.273438, 0.636719]
	35	[1, 2, 5112, 5054, 5110]	1.3594	[0.110352, 0.213867, 1.773438, 1.484375, 1.742188]
	36	[1, 2, 5114, 5111, 5064]	1.3906	[0.001472, 0.277344, 1.984375, 1.906250, 1.578125]
	37	[1, 19, 5116, 5107, 5095]	1.3828	[0.001205, 0.808594, 2.328125, 2.109375, 1.867188]
	38	[1, 5119, 5115, 5118, 5103]	1.4219	[0.316406, 3.078125, 2.921875, 3.031250, 2.359375]
	39	[1, 28, 61, 40, 5120]	1.6172	[0.601562, 1.265625, 1.476562, 1.406250, 5.437500]

Table C.22: RMSNorm weights γ ranked in descending order by absolute value for Mistral-12B (continued from Table C.21). The severest outliers have small weights (rank 1 or 2) across layers, but the other outliers sometimes have large weights (rank > 5000).

D Appendix for Chapter 6

D.1 Variants for Capturing Value-State Magnitude and Variety

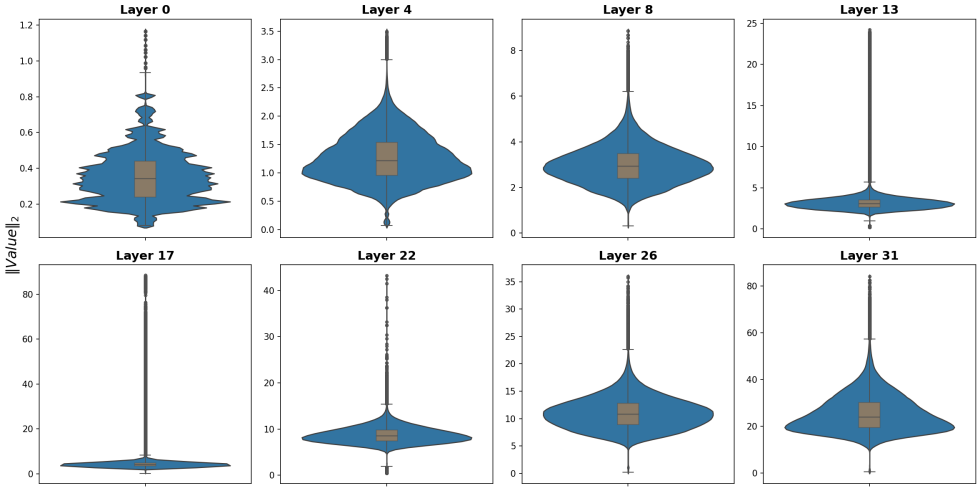


Figure D.22: Layer-wise violin plots of $L_2(\boldsymbol{v}_i) := \|\boldsymbol{v}_i\|_2$, where the value vectors $\boldsymbol{v}_i$ are from the full KV cache of Qwen3-4B during GSM8K generation. The distribution shows outliers that have large L_2 norm.

In this section, we explore different ways to compute the magnitude and variety of a value state $\boldsymbol{v} \in \mathbb{R}^d$ in KV cache: (1) $L_2(\boldsymbol{v}) = \|\boldsymbol{v}\|_2 = \sqrt{\sum_{j=1}^{d} |v_j|^2}$, (2) $\text{Range}(\boldsymbol{v}) := \max_{j\in[d]} v_j - \min_{j\in[d]} v_j$, and (3) $\text{Var}(\boldsymbol{v}) = \frac{1}{d}\sum_{j=1}^{d}(\boldsymbol{v}_j - \mu)^2$, where $\mu = \frac{1}{d}\sum_{j=1}^{d} v_j$.

First, we extract the value vectors $\boldsymbol{v}$ from the full KV cache during the generation of Qwen3-4B on GSM8K. Figures D.22 and D.23 present the layer-wise distributions of $L_2(\boldsymbol{v})$ and $\text{Range}(\boldsymbol{v})$, respectively. The violin plots show prominent outliers that have large $L_2(\boldsymbol{v})$ and $\text{Range}(\boldsymbol{v})$ in different layers. We also compute the Pearson correlation between $L_2(\boldsymbol{v})$ and $\text{Range}(\boldsymbol{v})$ and found that the two are highly correlated (> 0.8) over layers.

Next, we study whether preventing these outlier value states from eviction can maintain the accuracy under KV cache compression. We apply the framework introduced in Section 6.3 that pre-allocates dedicated value slots in the token budget. To decide which entries to keep in the slots, we experiment with the three value-scoring variants: keeping values with the

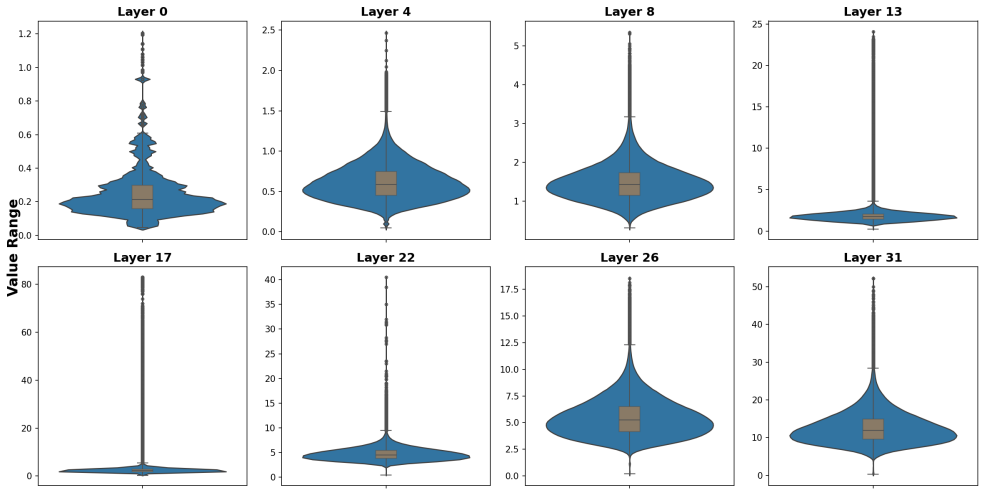


Figure D.23: Layer-wise violin plots of Range($\boldsymbol{v}_i$), where the values $\boldsymbol{v}_i$ are from the full KV cache of Qwen3-4B during GSM8K generation. The distribution shows outliers that have large Range.

largest $L_2(\boldsymbol{v})$, Range($\boldsymbol{v}$), or Var($\boldsymbol{v}$), respectively.[10] For the remaining budget, we apply stochastic sampling on top of the attention scores in SnapKV as described in Section 6.3. In Table D.23, we compare the value-scoring variants on GSM8K and AIME25. The three variants demonstrate comparable accuracy gains over the SnapKV (attention-only) baseline.

Method	**GSM8K**	**AIME25**
SnapKV	64.25	45.80
+ $L_2(\boldsymbol{v})$	84.55	59.17
+ Range($\boldsymbol{v}$)	84.45	59.19
+ Var($\boldsymbol{v}$)	84.00	57.71

Table D.23: Impact of value-state scoring variants on KV cache eviction accuracy. We report the results on Qwen3-4B at 4× compression. The three variants, which capture different aspects of value-state magnitude and variety, achieve comparable performance gains over the SnapKV baseline.

We choose Range($\boldsymbol{v}$) as the value scoring function for our main approach due to its connection with quantization; specifically, Range($\boldsymbol{v}$) captures the extreme values $\max \boldsymbol{v}$ and

[10] Here, Range is the same as VASE-ATTNV in the main content.

$\min \boldsymbol{v}$ in the vector, which is associated with the *outlier* challenge in LLM quantization [12]. We show the importance of large-Range value states in Section 6.3.1, where evicting them causes the model to enter a nonsensical reasoning loop:

"Wait, that doesn't make sense. Wait, no, the user might have had a different problem. Wait, the original problem is in the context..."

"So, the answer is the maximum profit, which is $125. But let me check my calculations again…So profit is 5125 - 5000 = 125.\n\nSecond option: $8000? Wait, the problem says \"the value of the item is $8000?\" Wait, the original problem says…"

Figure D.24: Examples of model outputs when evicting large-Range value states on GSM8K. The model falls into a repetitive loop and fails to reach the correct answer.

D.2 Token Statistics and Hyperparameters

	Prompt Tokens	**Gen Tokens**
AIME25	198	17,790
AIME26	148	16,642
HMMT25	127	18,189
MATH	94	5,281
LiveCodeBench-v6-medium	557	11,088

Table D.24: The average token counts of prompt (prefill) and generation (decode) of reasoning tasks. The **Gen Tokens** column represents the average number of tokens generated by the full Qwen3-4B model. The decode phase yields a significantly higher number of tokens than the prefill phase.

Table D.24 lists the average token counts of the prefill and decode phase of Qwen3-4B, respectively, showing that reasoning models may generate over 10,000 tokens at decode steps for a math question that has fewer than 200 prefill tokens. This intensifies the memory-bound constraints of the decode phase, motivating our approach to bound KV cache memory usage to a fixed cost.

Based on the statistics of Table D.24, we set the token budget $K = 4096$ for AIME25, AIME26, and HMMT25, $K = 1024$ for MATH, and $K = 2048$ for GPQA-Diamond, which roughly translates to a $4\times$ KV cache compression. For LiveCodeBench, we set $K = 2048$, which is $\sim 5\times$ compression.

All the eviction methods are training-free but have hyperparameters. We choose the best hyperparameters for each method on GSM8K and then apply them to our main experiments (Section 6.4.2) without further tuning. For R-KV, we experiment with different redundancy hyperparameters, $\lambda = \{0.1, 0.5, 0.9\}$, and set $\lambda = 0.5$. We set the Gaussian matrix rank $r = 20$ for CurDKV and VaSE-DKV, and the value reservation budget $N_v = K/4$ for VaSE-AttnV.

D.3 Statistical Significance

Method	**AIME25**	**AIME26**	**HMMT25**	**GPQA-D**	**MATH**
R-KV	54.6 ± 7.4	60.2 ± 7.4	44.6 ± 5.7	59.1 ± 3.0	86.1 ± 1.3
CurDKV	53.8 ± 7.6	61.3 ± 7.3	39.6 ± 5.3	48.5 ± 3.1	75.4 ± 1.7
VaSE-DKV	63.3 ± 7.5	68.8 ± 6.9	49.4 ± 5.4	56.8 ± 3.0	86.5 ± 1.3
VaSE-AttnV	64.0 ± 7.3	72.3 ± 6.5	50.0 ± 5.6	57.4 ± 3.0	85.4 ± 1.4

Table D.25: Reasoning-task accuracy (%) with standard errors for KV cache eviction methods on Qwen3-14B under $\sim 4\times$ cache compression.

Table D.25 shows the average accuracy $\pm$ standard errors for KV cache eviction methods on Qwen3-14B under $\sim 4\times$ cache compression. For each problem indexed $s \in \{1, \ldots, S\}$, we sample R independent generations and compute the per-problem pass@1 accuracy:

$$p_s = \frac{1}{R} \sum_{r=1}^{R} \mathbf{1}[\text{sample } r \text{ of problem } s \text{ is correct}], \tag{10}$$

where $R = 16$ for datasets with fewer than 100 examples and $R = 8$ otherwise. The overall

Token Budget	**2048**	**4096**	**Full**
Compression	4×	2×	1×
R-KV	59.09	63.32	65.25
CurDKV	48.48	60.54	
VASE-DKV	56.76	63.38	
VASE-AttnV	57.39	63.07	

Table D.26: GPQA-Diamond accuracy (%) for KV cache eviction methods on Qwen3-14B under different token budgets.

pass@1 accuracy and its standard error SE are then:

$$\bar{p} = \frac{1}{S}\sum_{s=1}^{S} p_s, \qquad \mathrm{SD} = \sqrt{\frac{1}{S-1}\sum_{s=1}^{S}(p_s - \bar{p})^2}, \qquad \mathrm{SE} = \frac{\mathrm{SD}}{\sqrt{S}}. \tag{11}$$

The variability captured by the standard errors reflects both the stochasticity of the model's sampling procedure and, for VASE methods, the randomness introduced by stochastic eviction. On tasks with larger standard errors (e.g., AIME25, AIME26), the small number of test problems (30 each) is the primary cause. While individual confidence intervals overlap due to small test sets, VASE outperforms all baselines across every task–model combination.

D.4 Additional Results

Table D.26 shows the Qwen3-14B results of GPQA-Diamond under different token budgets. While R-KV outperforms our VASE methods at 2048 token budget, the accuracy gap closes at 4096 tokens ($\sim 2\times$ compression). In this setting, both R-KV and our methods nearly recover the performance of the full model.

In Figure D.25, we extract the full value cache during the generation of Qwen3-4B on GSM8K examples and plot the distributions of Range($\boldsymbol{v}$) over different chunks of consecutive tokens. We put the first four sink tokens [183] into an individual chunk, which shows distinct distributions compared to the other chunks. Except for the first layer (Layer 0), the sink tokens have a lower median Range($\boldsymbol{v}$), which corresponds to the value-state drain phenomena

[175]. Excluding the sink tokens, the distributions of Range($\boldsymbol{v}$) are consistent as the sequence length increases.

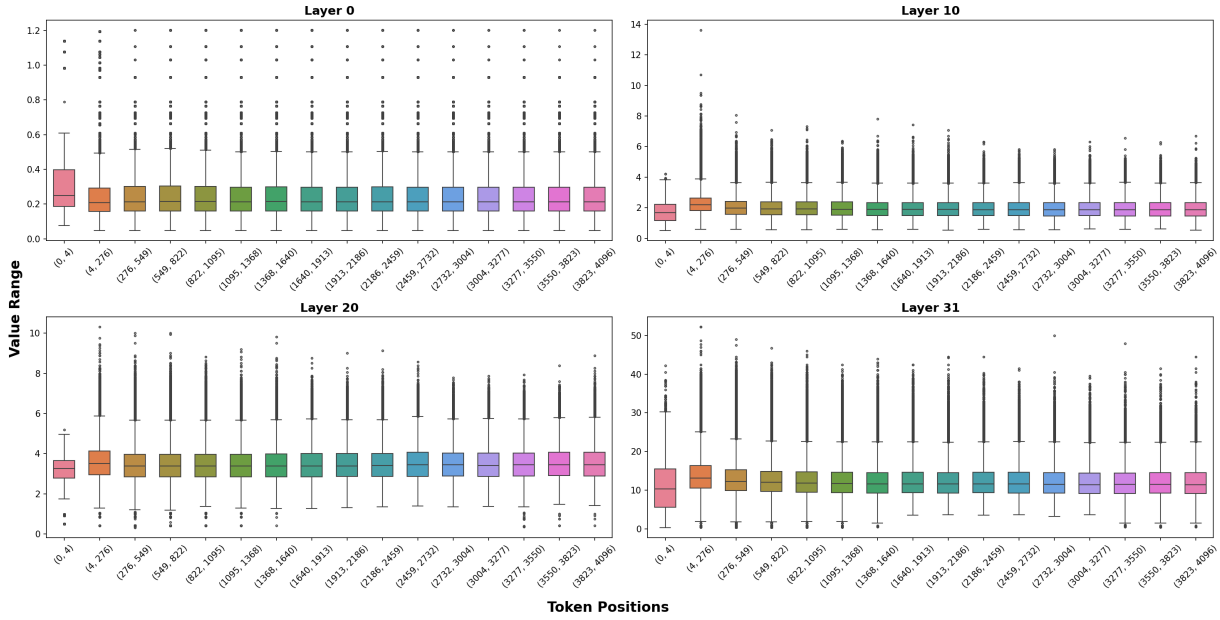


Figure D.25: Range($\boldsymbol{v}$) over token position chunks, where each chunk consists of the value states of consecutive tokens. Boxplots illustrate the dynamic range (y-axis) of value states at specific layers. Token positions (x-axis) are bucketed to show how the range distribution evolves through the sequence. The first chunk contains sink tokens at position $(0, 4)$. Excluding the sink tokens, the distribution of Range($\boldsymbol{v}$) does not drift as the context window grows.

D.5 Limitations

Our evaluation focuses on reasoning models and the decode phase of generation, where eviction methods offer the greatest benefit. As a result, we do not evaluate on prefill-phase compression benchmarks. However, because our method is built on fundamental observations of value-state distributions and stochastic diversity, we believe the core methodology can be extended to the prefill phase for long-prompt compression.

We only implement our methods on Qwen3 models because our selection-based baseline SeerAttention-R only releases checkpoints for Qwen models. Nevertheless, since our value-scoring function relies on simple min and max statistics of the value cache, our method is

model-agnostic and can be easily extended to other LLM architectures.

Lastly, while our findings link value-state magnitude to quantization errors, we focus on sparse attention, and quantization method development is beyond the scope of this chapter. Future research could explore an integrated framework that combines KV cache eviction with outlier-aware quantization to push the limits of compression ratio.

D.6 Compute Resources

Our experiments were conducted on NVIDIA A100 and H100 GPUs. Every experiment can be run on a single GPU with 80GB memory. Each task takes 15–72 GPU hours to finish under the Hugging Face backend, depending on the dataset size and the average number of generated tokens.

D.7 Licenses

As shown in Table D.27, we include licenses for any artifacts used in this chapter. Copyright © MAA indicates that AIME problems are copyrighted by the Mathematical Association of America and are used here for evaluation purposes only.

D.8 Broader Impacts

This work proposes a training-free KV cache eviction method that reduces the memory footprint of reasoning models during inference. The primary positive impact is enabling more efficient deployment of large language models, reducing hardware requirements and energy consumption. This can broaden access to reasoning-capable models for resource-constrained practitioners and organizations.

As a general-purpose inference optimization, VASE does not introduce new model capabilities or alter model outputs beyond the approximation inherent in cache compression. It therefore does not directly raise new ethical concerns beyond those already associated with

Asset	License	Source
Models		
Qwen3-4B	Apache 2.0	huggingface.co/Qwen/Qwen3-4B
Qwen3-14B	Apache 2.0	huggingface.co/Qwen/Qwen3-14B
Datasets & Benchmarks		
GSM8K	MIT	github.com/openai/grade-school-math
MATH	MIT	github.com/hendrycks/math
GPQA-Diamond	CC-BY-4.0	huggingface.co/datasets/Idavidrein/gpqa
LiveCodeBench-v6	MIT (code) / CC (data)	github.com/LiveCodeBench/LiveCodeBench
AIME 2025 / 2026	Copyright © MAA	artofproblemsolving.com/wiki
HMMT 2025	CC-BY-NC-SA 4.0	huggingface.co/datasets/MathArena/hmmt_feb_2025 huggingface.co/datasets/MathArena/hmmt_nov_2025
Code & Baselines		
SnapKV	CC-BY-4.0	openreview.net/forum?id=poE54GOq2l
R-KV	CC-BY-4.0	openreview.net/forum?id=2jwAjomEDB
CurDKV	CC-BY-4.0	openreview.net/forum?id=klmc4fwPLd
SeerAttention-R	CC-BY-4.0	openreview.net/forum?id=c5BOcHM6J8
HQQ	Apache 2.0	github.com/dropbox/hqq

Table D.27: Assets and their licenses.

the underlying language models. We do not foresee specific negative societal impacts arising from this work.