LEARNING WHEN TO ADAPT

Anonymous authors
Paper under double-blind review

ABSTRACT

Low-rank adaptation (LoRA) is a widely used parameter-efficient fine-tuning method, yet its learned correction is static: the same low-rank update is applied to every input. This input-agnostic approach creates an inevitable compromise between adapting to the fine-tuning distribution and preserving pre-trained behavior on inputs outside that distribution, contributing to *catastrophic forgetting*. We introduce DISeL (**D**ynamic **I**nput-**Se**nsitive **L**oRA), which augments LoRA modules with lightweight input-dependent gates over individual rank-one components. The gating mechanism is designed to preserve the pre-trained model's behavior by default, while training learns to activate selected components that reduce the fine-tuning loss. DISeL adds only a small number of parameters and preserves the low-rank structure. Across RoBERTa on GLUE, and LLAMA and MISTRAL models fine-tuned for mathematical reasoning and code generation, DISeL reduces forgetting relative to LoRA and related variants while maintaining competitive fine-tuning accuracy. In addition, the learned gate activations provide an interpretable diagnostic view of which layers and rank components are most activated during fine-tuning, giving insight into where task-specific adaptation is concentrated.

1 INTRODUCTION

Fine-tuning (FT) has become a central stage in the deployment of large language models (LLMs) where pre-trained models with general knowledge are specialized for instruction-following (Ouyang et al., 2022; Wei et al., 2022), mathematical reasoning (Yu et al., 2024; Yue et al., 2024), code generation (Roziere et al., 2023; Luo et al., 2024), and other downstream tasks. The most direct approach is *full fine-tuning*, which updates all parameters of the pre-trained model without constraint. Parameter-efficient fine-tuning (PEFT) methods instead restrict updates to a smaller, structured subspace. The most popular approach is Low-Rank Adaptation (LoRA) (Hu et al., 2022), which freezes the pre-trained weights and learns a low-rank correction using matrix factorization.

Despite achieving strong performance on their target tasks, fine-tuned models often exhibit degradation in their broader capabilities acquired during pre-training (Luo et al., 2025; Dong et al., 2024). This phenomenon, known as *catastrophic forgetting*, has been frequently studied in continual learning (Wang et al., 2024a; Liang et al., 2025; Wang et al., 2022a; 2023; McDonnell et al., 2023) and remains a key challenge in modern LLM fine-tuning. The extent of forgetting depends on the choice of fine-tuning method. Typically, full fine-tuning incurs substantial performance degradation outside the fine-tuning domain (Biderman et al., 2024; Luo et al., 2025), whereas LoRA mitigates this effect but does not eliminate it (Biderman et al., 2024; Shuttleworth et al., 2025).

The main reason for forgetting in these FT methods is that the learned modification is input-agnostic. Full fine-tuning, LoRA, and its variants such as DoRA (Liu et al., 2024) and AdaLoRA (Zhang et al., 2023) only differ in parameterization. Yet, they all apply a constant modification across the entire input space, regardless of whether correction is required. As a result, the same adaptation affects both fine-tuning-domain inputs, where correction is needed, and pre-training-domain inputs, where the original mapping should be preserved. This leads to an inherent compromise between adaptation and retention.

Existing input-dependent adaptations are primarily designed to mitigate interference across tasks or among multiple LoRA modules. For instance, mixture-of-LoRA approaches (Wu et al., 2024; Buehler & Buehler, 2024; Tian et al., 2024) learn routing or gating mechanisms that select or combine multiple LoRA modules based on the input. The closest to this work is GATED LoRA (Eom et al.,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

2025), which introduces for each rank-one-component, an input-dependent gate within a single adapter. However, its primary goal is to reduce inter-task interference in multi-task fine-tuning, rather than to preserve pre-trained behavior. None of the current methods explicitly target catastrophic forgetting, nor are they designed to make the adapter inactive on inputs outside the fine-tuning distribution, a key property for preserving pre-trained capabilities.

We propose DISeL (**D**ynamic **I**nput-**Se**nsitive **LoRA**), which preserves the low-rank parameterization of LoRA but makes the correction input-dependent and active only when necessary. The key idea is to multiply each rank-one component by an input-dependent learnable gate whose output lies in $[0, 1]$. The gates are effectively pre-set to zero in initialization, meaning that DISeL begins training with the pre-trained model. During training, gates are activated only when doing so reduces the fine-tuning loss, leading to updates that are applied selectively across inputs rather than uniformly.

Contributions. We identify the input-agnostic nature of full fine-tuning/LoRA modifications as a contributing factor in catastrophic forgetting, and illustrate this in a minimal linear regression setting (Sec. 2). We introduce DISeL, which augments each LoRA module with lightweight, input-dependent gates that activate only when needed (Sec. 3). Experiments on LLAMA and MISTRAL models fine-tuned for mathematical reasoning and code generation show that DISeL consistently reduces catastrophic forgetting compared with LoRA and its variants, while maintaining competitive in-domain accuracy (Sec. 4). The learned gate activations reveal which layers are most engaged during fine-tuning, providing direct insight into where task-specific knowledge is localized within the network (Sec. 5).

2 Background and Motivation

Input-agnostic fine-tuning. Consider a linear layer with pre-trained weights $\mathbf{W}_0 \in \mathbb{R}^{d_y \times d_x}$. Given an input representation $\mathbf{x} \in \mathbb{R}^{d_x}$, the layer produces $\mathbf{W}_0\mathbf{x} \in \mathbb{R}^{d_y}$. Fine-tuning learns a matrix $\mathbf{\Delta}$ and changes this output to

$$(\mathbf{W}_0 + \mathbf{\Delta})\mathbf{x} = \mathbf{W}_0\mathbf{x} + \mathbf{\Delta}\mathbf{x}. \tag{1}$$

Full FT (Ouyang et al., 2022; Devlin et al., 2019) learns an unconstrained update, whereas PEFT methods restrict its structure. In particular, LoRA (Hu et al., 2022) represents $\mathbf{\Delta}$ by a low-rank factorization. In either case, $\mathbf{\Delta}$ is fixed after training and the same update rule $\mathbf{\Delta}\mathbf{x}$ is used for every input. This does not mean that $\mathbf{\Delta}\mathbf{x}$ has the same value for all $\mathbf{x}$; it means that the update has no mechanism for deciding that it should be inactive on a particular input. Ideally, it should modify inputs from the fine-tuning distribution p_{ft}, while leaving inputs from the pre-training distribution p_{pt} close to $\mathbf{W}_0\mathbf{x}$. A single fixed update must serve both goals, producing an adaptation–retention tradeoff.

A regression framework for selective adaptation. We introduce a simple regression model of this tradeoff for one adapted linear layer. Here, $\mathbf{x}$ is the layer input, $\mathbf{W}_0\mathbf{x}$ is its pre-trained output, and $\mathbf{y}$ is the output we would like after adaptation. Suppose that the required change can contain any subset of r correction components $\mathbf{M}_1\mathbf{x}, \ldots, \mathbf{M}_r\mathbf{x}$. For each input, a hidden binary variable Z_i records whether the ith component is required: $Z_i = 1$ includes $\mathbf{M}_i\mathbf{x}$, while $Z_i = 0$ leaves it out. We assume that these switches are independent and that $Z_i \sim \text{Bernoulli}(\alpha_i)$, where $\alpha_i \in (0, 1)$ is the prior probability that component i is needed. The total required correction, denoted by $T(\mathbf{x}, Z)$, and the corresponding regression target are

$$T(\mathbf{x}, Z) := \sum_{i=1}^{r} Z_i \mathbf{M}_i \mathbf{x}, \qquad \mathbf{y} = \mathbf{W}_0\mathbf{x} + T(\mathbf{x}, Z). \tag{2}$$

Thus, $T(\mathbf{x}, Z)$ is simply the change relative to the pre-trained output. When every switch is zero, no change is required and the target is exactly $\mathbf{W}_0\mathbf{x}$; nonzero switch patterns represent inputs that require one or more parts of the task update.

The switches are not observed at prediction time, so they must be inferred from $\mathbf{x}$. We model the relationship between the input and the required switch pattern with a Gaussian mixture model (GMM). Gaussian mixtures provide a flexible way to model heterogeneous input distributions. We study a structured shared-covariance mixture in which different input regimes require different combinations of correction components. The shared covariance is $\mathbf{\Sigma} \succ 0$. Each switch has a vector $\boldsymbol{\mu}_i$ that describes

how the input mean changes when component i is needed. We center the two choices symmetrically, so changing a switch from off to on shifts the mean by exactly $\boldsymbol{\mu}_i$. Write $Z = (Z_1, \ldots, Z_r)$ and let $\mathbf{z} = (z_1, \ldots, z_r) \in \{0,1\}^r$ denote one possible switch pattern. Conditional on this pattern, we draw the input as

$$\mathbf{x} \mid Z = \mathbf{z} \sim \mathcal{N}\left(\sum_{i=1}^{r}(z_i - \tfrac{1}{2})\boldsymbol{\mu}_i, \boldsymbol{\Sigma}\right), \qquad \Pr(Z = \mathbf{z}) = \prod_{i=1}^{r}\alpha_i^{z_i}(1-\alpha_i)^{1-z_i}, \tag{3}$$

Each pattern selects one Gaussian component, and the product on the right is its probability under the independent Bernoulli switches. Marginalizing the 2^r possible patterns gives a GMM with 2^r Gaussian components.

Given only $\mathbf{x}$, the learner predicts the required correction with a function $f(\mathbf{x})$, so its adapted output is $\mathbf{W}_0\mathbf{x} + f(\mathbf{x})$. We first ask what the best possible predictor is in this setting, without imposing any parametric restriction on f. Under squared loss, this is the Bayes-optimal predictor, defined as the function that minimizes the population risk. Because the pre-trained term is shared with the target in (2), prediction error depends only on the correction, and the relevant risk is

$$\mathcal{R}(f) \coloneqq \tfrac{1}{2}\mathbb{E}\left[\|f(\mathbf{x}) - T(\mathbf{x}, Z)\|_2^2\right]. \tag{4}$$

A conventional fixed update has $f(\mathbf{x}) = \boldsymbol{\Delta}\mathbf{x}$ and cannot choose its correction components separately for each input. The Bayes-optimal predictor can instead average over the unknown switches according to how likely they are after observing $\mathbf{x}$.

Proposition 1 (Bayes-optimal predictor)**.** *For each component, define the residual interaction term*

$$\delta_i(\mathbf{x}) \coloneqq \log\frac{p(\mathbf{x} \mid Z_i = 1)}{p(\mathbf{x} \mid Z_i = 0)} - \boldsymbol{\mu}_i^\top \boldsymbol{\Sigma}^{-1}\mathbf{x},$$

where $p(\mathbf{x} \mid Z_i = s)$ is the conditional density of the input when switch i is fixed to s. Then the minimizer of (4) is

$$f^\star(\mathbf{x}) = \sum_{i=1}^{r}\pi_i(\mathbf{x})\mathbf{M}_i\mathbf{x}, \quad \pi_i(\mathbf{x}) = \Pr(Z_i = 1 \mid \mathbf{x}) = \sigma\left(\boldsymbol{\mu}_i^\top \boldsymbol{\Sigma}^{-1}\mathbf{x} + \log\tfrac{\alpha_i}{1-\alpha_i} + \delta_i(\mathbf{x})\right), \tag{5}$$

where $\sigma(t) \coloneqq (1 + e^{-t})^{-1}$.

A complete proof is given in Appendix I. The result has a direct interpretation: $\pi_i(\mathbf{x})$ is the probability, after seeing the input, that component $\mathbf{M}_i\mathbf{x}$ is required. The optimal correction softly includes each component in proportion to this probability. The term $\boldsymbol{\mu}_i^\top \boldsymbol{\Sigma}^{-1}\mathbf{x}$ is linear in the input, $\log(\alpha_i/(1-\alpha_i))$ is a constant bias determined by how often component i is needed, and $\delta_i(\mathbf{x})$ captures the remaining interactions with the other switches. If $\delta_i(\mathbf{x}) = 0$, as it does when $r = 1$, then $\pi_i(\mathbf{x})$ is exactly a sigmoid applied to a linear function of $\mathbf{x}$ plus a bias. For $r > 1$, each $p(\mathbf{x} \mid Z_i = s)$ is a mixture of 2^{r-1} Gaussians, so $\delta_i(\mathbf{x})$ is generally nonlinear. This Bayes-optimal form motivates softly selecting correction components according to the input. Inspired by this structure, in the next section we introduce a practical adapter with a similar input-dependent mechanism based on learned linear scores and biases.

3 DISEL: DYNAMIC INPUT-SENSITIVE LORA

In Section 2, we saw that different inputs may require different subsets of correction components. A fixed update matrix $\boldsymbol{\Delta}$ cannot select these components separately for each input, whereas the Bayes-optimal predictor weights each component by its input-dependent posterior probability. Motivated by this structure, let us consider a general adapter of the form

$$\mathbf{y} = \mathbf{W}_0\mathbf{x} + f(\mathbf{x}; \boldsymbol{\theta}), \tag{6}$$

where $f : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$ is a trainable mapping. The desired behavior for f is to provide the necessary adaptation on inputs from p_{ft}, while vanishing on inputs from p_{pt}. Meanwhile, the method should remain memory and computationally efficient to ensure practical applicability.

Method. To this end, we build our approach based on LORA, a widely adopted PEFT method, which freezes $\mathbf{W}_0$ and learns a low-rank correction as a product $\mathbf{ABx}$ with $\mathbf{A} \in \mathbb{R}^{d_y \times r}$ and $\mathbf{B} \in \mathbb{R}^{r \times d_x}$.

The key idea behind DISeL is to multiply each rank-one component by an input-dependent scalar gate $g_i(\mathbf{x}) \in [0, 1]$. Collecting these into a gate vector $g(\mathbf{x}) = (g_1(\mathbf{x}), \ldots, g_r(\mathbf{x}))^\top$ and a diagonal matrix $\mathbf{G}(\mathbf{x}) = \mathrm{diag}(g(\mathbf{x}))$, the DISeL adaptation is defined as

$$f(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{A}\mathbf{G}(\mathbf{x})\mathbf{B}\mathbf{x} = \sum_{i=1}^{r} g_i(\mathbf{x})\mathbf{a}_i\mathbf{b}_i^\top\mathbf{x}, \tag{DISeL}$$

where $\mathbf{a}_i$ and $\mathbf{b}_i^\top$ are $\mathbf{A}$'s columns and $\mathbf{B}$'s rows, respectively. The gate vector is parameterized as

$$g(\mathbf{x}) = \sigma\big(\mathbf{W}_g\mathbf{x} + \mathbf{b}_g\big) \in [0, 1]^r, \tag{7}$$

with $\mathbf{W}_g \in \mathbb{R}^{r\times d_x}$, $\mathbf{b}_g \in \mathbb{R}^r$, and $\sigma(z) = 1/(1 + e^{-z})$ applied element-wise. Each $g_i(\mathbf{x})$ plays the same structural role as the posterior probability $\pi_i(\mathbf{x})$ in Proposition 1. There, the posterior logit is an affine function of $\mathbf{x}$ plus the generally nonlinear interaction term $\delta_i(\mathbf{x})$. Equation (7) uses a learned affine logit, which is exact in the single-component case and provides an efficient linear approximation when multiple components interact. We use a sigmoid rather than a softmax so that gates act as independent soft switches, allowing multiple rank-one components to be active at the same time, or all inactive simultaneously, for the same input.

Parameter and computational overhead. DISeL introduces additional $rd_x + r$ parameters per adapted block, with a total of $rd_y + 2rd_x + r$ trainable parameters, compared with $rd_y + rd_x$ for LoRA. The gate computation requires $\mathcal{O}(d_x r)$ arithmetic operations.

Near-closed initialization. During fine-tuning, we typically observe only data from p_{ft}, whereas the Bayes-optimal component probabilities depend on the full input distribution, including inputs for which a component is unnecessary. Without access to p_{pt}, we cannot directly enforce $f(\mathbf{x}; \boldsymbol{\theta}) \approx \mathbf{0}$ on pre-training-domain inputs; instead, we encode this as the *default* behavior at initialization. Specifically, we initialize $\mathbf{b}_g$ to a negative value, such that $\sigma(\mathbf{b}_g)$ is approximately zero, and initialize $\mathbf{W}_g$ using a standard scheme, *e.g.*, He et al. (2015). Combined with the standard LoRA initialization, this makes DISeL start equal to the pre-trained model on all inputs. During training, each gate opens when doing so reduces the FT loss. This does not recover the exact posterior without pre-training data, but provides an inductive bias toward suppressing unnecessary corrections. The initialization of $\mathbf{b}_g$ is a hyperparameter that may depend on the task and dataset; in our LLAMA and MISTRAL experiments, we set $\mathbf{b}_g = -3 \cdot \mathbf{1}$, where $\sigma(-3) \approx 0.05$.

Gates learning vs. $\mathbf{A}, \mathbf{B}$ fine-tuning. The factors $\mathbf{A}$ and $\mathbf{B}$ play the same role as in standard LoRA: they parameterize the adaptation itself and thus are tuned with a small learning rate. In contrast, the gate parameters $(\mathbf{W}_g, \mathbf{b}_g)$ learn when each rank-one correction component is useful for the current input. Accordingly, we treat $(\mathbf{A}, \mathbf{B})$ and $(\mathbf{W}_g, \mathbf{b}_g)$ as two distinct optimization problems and assign them *separate learning rates*, with the gate learning rate substantially larger. In our LLAMA and MISTRAL experiments, we use a learning rate $5\times$ larger for the gate parameters than for $\mathbf{A}$ and $\mathbf{B}$.

4 EXPERIMENTS

In this section, we evaluate the downstream performance of DISeL in two settings: (i) fine-tuning ROBERTa-base Liu et al. (2019) on the GLUE benchmark (Wang et al., 2018) (Section 4.1), and (ii) instruction tuning of LLAMA 2-7B Touvron et al. (2023) and MISTRAL-7B Jiang et al. (2023) on mathematical reasoning and code generation tasks (Section 4.2). For each setting, we report two quantities: fine-tuning task performance and forgetting on benchmarks outside the fine-tuning domain. We compare DISeL against full FT and four widely used PEFT baselines: LoRA (Hu et al., 2022), DoRA (Liu et al., 2024), AdaLoRA (Zhang et al., 2023), and Gated LoRA (Eom et al., 2025).

4.1 ROBERTA ON GLUE

Setup. We fine-tune ROBERTa-base on five GLUE tasks: MNLI, SST-2, QNLI, CoLA, and MRPC. For each task, we consider ranks $r \in \{4, 8, 16, 32, 64\}$. Fine-tuning performance is measured as the average test accuracy across tasks. The forgetting is quantified via masked language modeling (MLM) perplexity of the fine-tuned model on BookCorpusOpen (Bandy & Vincent, 2021), CC-News (Mackenzie et al., 2020), and WikiText-2 (Merity et al., 2017), where the pre-trained model achieves an averaged perplexity of ≈ 6.4 on these datasets. Each configuration (method, task, rank)

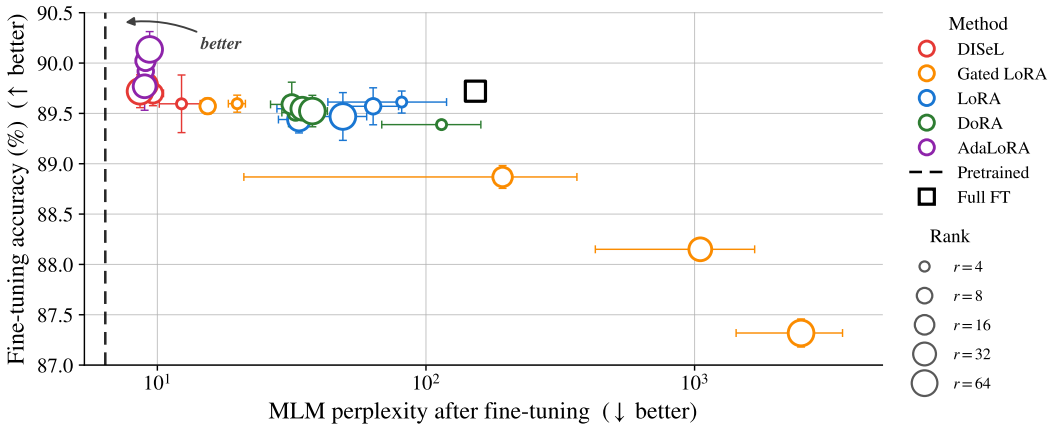


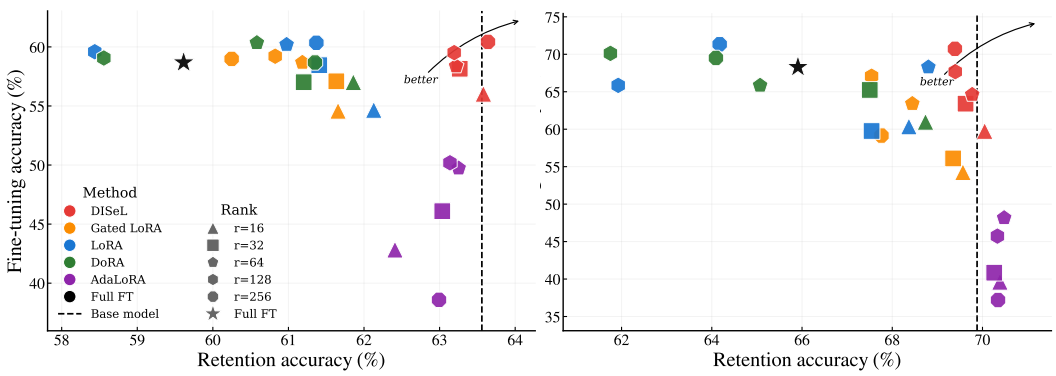
Figure 1: **RoBERTa on GLUE.** We fine-tune RoBERTa-base on five GLUE tasks: MNLI, SST-2, QNLI, CoLA, and MRPC, using various methods and ranks. The vertical axis measures the average test accuracy for the corresponding tasks, while the horizontal axis indicates the average masked-LM perplexity, evaluated on BookCorpusOpen, CC-News, and WikiText-2. Each point represents a method-rank pair averaged across three seeds, with error bars indicating the standard deviation. The dashed line marks the pre-trained baseline. DISeL and AdaLoRA remain close to this baseline while matching the FT accuracy of full FT. Gated LoRA attains strong FT accuracy, but its perplexity grows sharply with rank, indicating severe forgetting rather than failed task optimization.

is evaluated over three random seeds, and we report the mean $\pm$ standard deviation. Full details are provided in Appendix IV.

Figure 1 plots the FT accuracy against retention. While most methods achieve comparable FT accuracy, they differ substantially in retention performance. Full fine-tuning attains high accuracy but deviates strongly from the pre-trained baseline, yielding a mean perplexity of roughly 150. Fixed-rank adapters such as LoRA and DoRA mitigate this effect at the cost of slightly reduced FT accuracy, but still incur substantial forgetting, with perplexity ranging from 30 to 100. Gated LoRA also attains strong FT accuracy, especially at lower ranks, confirming that training succeeds and the model learns the downstream tasks. Its MLM perplexity, however, increases steeply with rank and exceeds 10^3 for the largest adapters. Thus, its poor retention is not caused by failed optimization: Gated LoRA successfully adapts to GLUE, but the resulting update severely damages its pre-trained language-modeling ability. Proposition 2 in Appendix II formalizes one possible structural source of this behavior. The top-k and ReLU variants of Gated LoRA are positively homogeneous, which prevents them from suppressing a correction on one input distribution while activating it on a scaled version of the same inputs; in this setting, the proposition proves an unavoidable lower bound on the joint adaptation and retention losses. By contrast, DISeL and AdaLoRA remain close to the pre-trained baseline, with perplexity around 9, while matching or surpassing the accuracy of full fine-tuning. DISeL thus achieves a similarly favorable tradeoff between accuracy and forgetting as pruning-based AdaLoRA, without requiring a pruning schedule. Although AdaLoRA performs competitively on this benchmark, the next section shows that its FT accuracy degrades at larger scales and on more challenging tasks, such as mathematical reasoning and code generation.

4.2 Llama and Mistral on mathematical reasoning and code generation

Setup. We extend our evaluation to large-scale instruction fine-tuning tasks. Specifically, we fine-tune Llama 2-7B on MetaMath (Yu et al., 2024) for math reasoning, and Mistral-7B on MagiCoder (Wei et al., 2024) for code generation. We evaluate FT performance on GSM8K (Cobbe et al., 2021) for math, and HumanEval (Chen et al., 2021) for code. For each model, we use ranks $r \in \{16, 32, 64, 128, 256\}$. To measure forgetting, we evaluate retention on 14 out-of-domain benchmarks, spanning commonsense reasoning, world knowledge, reading comprehension, and medical question answering, and report their unweighted mean accuracy. Appendix V lists the full benchmark suite and experimental details.



(a) LLAMA math fine-tuning

(b) MISTRAL code fine-tuning

Figure 2: **LLAMA and MISTRAL instruction fine-tuning.** We fine-tune LLAMA and MISTRAL on math and code instructions, respectively, with various methods using different ranks. Panel (a) shows the results for LLAMA, whereas Panel (b) depicts them for MISTRAL. Here, the vertical axis shows test accuracy for the corresponding fine-tuning task, and the horizontal axis shows the average accuracy across several pre-training tasks. The dashed vertical line marks the accuracy of the pre-trained model. DISeL maintains retention close to this baseline while achieving strong fine-tuning accuracy. Although GATED LORA also learns both target tasks, every one of its plotted configurations is Pareto-dominated by a DISeL configuration.

Figure 2 visualizes the tradeoff between fine-tuning and retention. Here, DISeL achieves strong task accuracy while maintaining retention close to the base model. On LLAMA, it reaches around 60% FT accuracy with retention near 63.3%, only slightly below the base model. On MISTRAL, our approach exceeds 70% FT accuracy while maintaining retention within a fraction of a percentage point of the base model ($\approx 70\%$). The occasional improvement over base-model retention suggests that fine-tuning can also improve the general reasoning abilities probed by some retention benchmarks.

In contrast, full FT attains reasonable task accuracy but incurs substantial forgetting, shifting far left of the base-model retention line and failing to dominate the adapter methods. Its relatively small forgetting compared with the worst adapter runs is partly explained by the smaller learning rate used for fine-tuning. GATED LORA successfully learns both target tasks, reaching up to roughly 59% accuracy on GSM8K. On HUMANEVAL, most ranks achieve only 54–64%, and only the rank-128 configuration reaches 67%. Its strongest task-performing configurations nevertheless remain about 2–3 retention points below the corresponding base models. Indeed, every plotted GATED LORA configuration in both panels is Pareto-dominated by at least one DISeL configuration. This shows that, in these settings, input-dependent rank gating alone is insufficient to obtain the adaptation–retention tradeoff achieved by our separate sigmoid router. Among the remaining PEFT methods, ADALORA matches DISeL in retention, as in the ROBERTa experiments, but its FT accuracy drops to roughly 40–50% on both models. LORA and DORA perform competitively at low ranks, but their retention deteriorates as rank increases, losing 4–8% at the largest ranks. By comparison, DISeL achieves consistent performance across ranks, models, and datasets, preserving both task performance and pre-training capability.

This robustness is practically useful beyond the final accuracy–retention tradeoff itself. Recent work has shown that LORA's performance can be highly sensitive to learning rate, rank, initialization, and scaling choices, often requiring careful tuning to achieve strong performance (Chen et al., 2026). In contrast, the gating mechanism of DISeL provides implicit regularization by limiting the effective modification of inactive or weakly activated components, thereby stabilizing training at larger or suboptimal adapter learning rates. Moreover, the separate learning rate for the gates also provides a direct control knob for the plasticity–retention tradeoff: larger gate learning rates encourage more gates to open and favor adaptation to the new task, whereas smaller gate learning rates keep more directions closed and favor retention of pre-training knowledge.

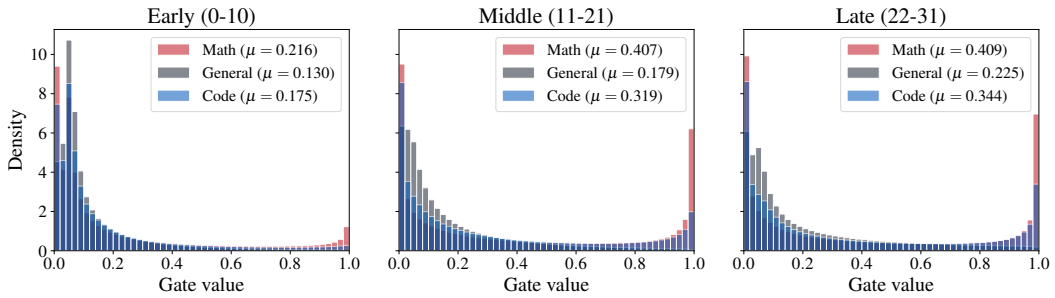


Figure 3: **Gate activation histograms across domains.** We analyze gate activations of a rank-32 LLAMA math adapter on held-out samples from math, general text, and code. Each panel corresponds to a layer-depth band: early, middle, or late layers. Here, the gates are substantially more open for math inputs, the fine-tuning domain, than for general text from the pre-training distribution. The largest differences appear in the middle and later layers. This demonstrates that DISeL selectively applies adaptation on inputs from the fine-tuned distribution.

5 INTERPRETABILITY OF DISEL

Setup. The main idea behind DISeL is that its gates are *input-dependent*. Specifically, the adapter correction should be active for inputs from the fine-tuning task and suppressed for others. We examine this behavior at two levels: whether gate patterns are stable for individual samples within the same domain, and how their aggregate usage varies across domains, layers, and modules. We use held-out math samples from GSM8K, code samples from HUMANEVAL, and general-text passages from WIKITEXT-2. For math and code, we analyze response tokens; for general text, we use all tokens. Unless stated otherwise, the domain and module analyses use the rank-32 LLAMA 2-7B math adapter from Section 4.2. Additional results are given in Appendix VII.

Sample-level fingerprints. Using the rank-32 LLAMA math and MISTRAL code adapters, we analyze 50 held-out GSM8K samples and 50 held-out HUMANEVAL samples. We concatenate the post-sigmoid gates from every adapted module and rank component at each response token. Averaging these vectors separately over odd- and even-position tokens gives two non-overlapping gate fingerprints for each sample. We match each odd-token fingerprint to the most similar of the 50 even-token fingerprints using cosine similarity. As shown in Table 1, matching accuracy reaches 100% on GSM8K and 80% on HUMANEVAL, compared with 2% chance. Although samples from the same domain have similar fingerprints overall, the correct pairs remain distinguishable, showing that the gates encode stable sample-specific structure in addition to the pattern shared across a domain.

Domain analysis. Figure 3 shows gate activations across different input domains. In early layers, the gates are nearly closed across all domains, suggesting that adaptation in these layers might not be needed. Yet, in the middle and late layers, there is a clear separation: math samples have substantially larger gate values than general text, while code lies between the two. This ordering makes sense in this setup: math is the fine-tuning domain, general text is outside that domain, and code shares some structured-symbolic and logical reasoning with math. Overall, the gates are not static; they open more to inputs from the fine-tuning domain than to those from the pre-trained domain.

6 RETENTION OVER TIME

The tradeoff plots in Section 4 report retention at the best target-task checkpoint, but do not reveal how retention *evolves* during training. A method may end close to the base model while passing through degraded intermediate states, or it may lose capability steadily from the first few optimization steps. To distinguish these regimes, we track retention at every saved checkpoint of the LLAMA 2-7B, for ranks $r \in \{16, 32, 64, 128, 256\}$. As before, retention is measured as the mean accuracy over the 14-benchmark suite from Section 4.2. Additional results are provided in Appendix VI.

Figure 4 shows the resulting trajectories for DISeL (left) and LORA (right). For LORA, retention degrades progressively with rank: at r=16, it remains within about one percentage point of the base

Table 1: **Sample-level gate fingerprints.** Similarity columns report mean cosine similarity between odd- and even-token fingerprints; matching selects the most similar even-token fingerprint among 50 samples.

Model and task	Same sample	Other samples	Match accuracy
LLAMA/GSM8K	0.997	0.975	50/50 (100%)
MISTRAL/HUMANEVAL	0.973	0.929	40/50 (80%)

model, but at higher ranks it exhibits a clear downward drift, falling roughly a 5% at the largest ranks. This degradation is persistent rather than transient; once lost, performance does not recover within the training horizon. In contrast, DISeL maintains near-flat retention curves across all ranks, with deviations remaining within about half a percentage point of the base model from the first checkpoint to the last. This stability can be attributed to the fact that input-dependent gates keep the correction largely inactive on inputs outside the fine-tuning distribution, preventing increased capacity from inducing additional drift. As a result, DISeL effectively decouples adapter rank from forgetting, enabling the use of larger ranks without sacrificing retention.

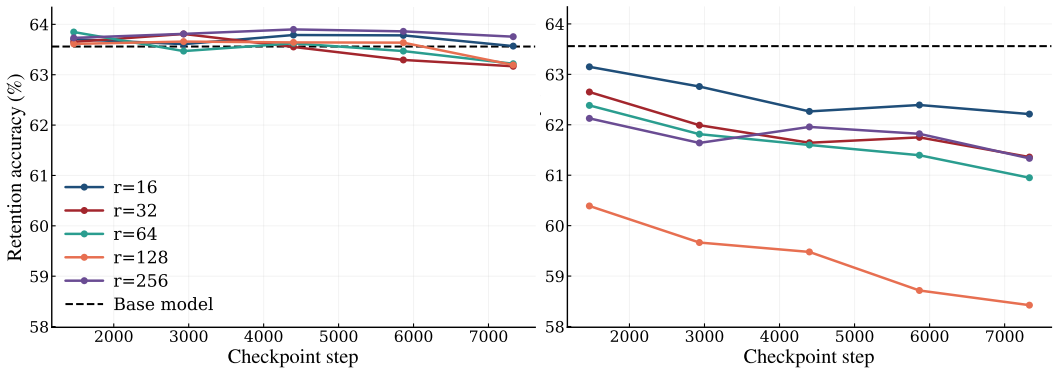
Finally, we observe that forgetting could also depend on the optimization setting. For LoRA, the r=128 run exhibits more forgetting than r=256, as the latter required a smaller learning rate (10^{-4}) for stability, whereas the former uses a twice larger step size. A similar effect appears in Figure 2, where Full FT does not incur the largest forgetting, partly due to its smaller learning rate used (10^{-5}).

7 Related work

We discuss three lines of work most related to DISeL: parameter-efficient low-rank fine-tuning, input-conditional and mixture-style adapters, and continual learning of pre-trained models.

Parameter-efficient and low-rank fine-tuning. LoRA (Hu et al., 2022) approximates the FT update using a fixed low-rank correction $\mathbf{\Delta} = \mathbf{AB}$ added to a frozen pre-trained weight, and has become the dominant approach to parameter-efficient adaptation. A large body of follow-up work refines the *parameterization* of $\mathbf{\Delta}$: AdaLoRA (Zhang et al., 2023) adapts the rank budget across layers, DoRA (Liu et al., 2024) decouples magnitude from direction, PiSSA (Meng et al., 2024) initializes $\mathbf{A}, \mathbf{B}$ from the principal singular subspace of $\mathbf{W}$, LoRA-GA (Wang et al., 2024b) chooses the initialization such that the first gradient step approximates full FT, VeRA (Kopiczko et al., 2024) freezes shared random $\mathbf{A}, \mathbf{B}$ and learns only a diagonal scaling between them, structurally close to the $\mathbf{G}(\mathbf{x})$ in DISeL but fixed across inputs, and AuroRA (Dong et al., 2025) introduces a nonlinear coupling between the factors. Other variants modify optimization dynamics: LoRA+ (Hayou et al., 2024) uses asymmetric learning rates for $\mathbf{A}$ and $\mathbf{B}$, LoRAM (Jiang et al., 2025) optimizes the low-rank adapter on Riemannian manifolds, and QLoRA (Dettmers et al., 2023) combines low-rank adaptation with weight quantization. Despite their differences, these methods produce a single correction matrix that is uniformly applied to every input during inference. DISeL preserves the low-rank parameterization of LoRA while making the correction input-dependent through lightweight gates. This selective adaptation mechanism directly addresses the structural conflict identified in Section 2.

Input-conditional and mixture-of-LoRA approaches. A separate line of work introduces input- or task-conditional adaptation by combining *multiple* LoRA modules. MoLE (Wu et al., 2024) routes between several LoRA adapters via a learned gating network, in the spirit of mixture-of-experts. LoRAHub (Huang et al., 2024) composes pre-trained task-specific LoRA modules into a single weighted combination selected for each downstream task. These approaches operate at the coarse-grained level of entire adapters, and scale both the parameter count and the routing cost with the number of modules. The closest architectural prior is Gated LoRA (Eom et al., 2025), which treats each rank-one component as a mini-expert, and selects the active components based on the input. Specifically, it uses a ReLU activation function on the same down-projection features used to compute the LoRA update, thereby avoiding a separate router. Its primary goal is to improve parameter utilization and reduce inter-task interference, especially in multi-task fine-tuning. Similar to Gated LoRA, DISeL also gates rank-one components inside a single adapter. However, DISeL uses separate learnable sigmoid gates initialized near zero so that the default behavior matches



(a) DISeL (b) LoRA

Figure 4: **Retention vs. training step.** We evaluate the retention performance during fine-tuning from the experiment in Sec. 4.2. Here we present the results for LLAMA 2- 7B model fine-tuned on METAMATHQA using LoRA and DISeL with various ranks. Panel (a) shows the result for DISeL, while Panel (b) plots them for LoRA. We see that DISeL maintains near-zero retention drop at every rank, including the largest, whereas LoRA's retention deteriorates with both rank and training time.

the pre-trained model. More importantly, our objective is fundamentally different: rather than improving multi-task routing, DISeL is designed to preserve pre-trained behavior on inputs outside the fine-tuning distribution.

Continual learning. Continual learning addresses forgetting across a sequence of tasks. GAIN-LoRA (Liang et al., 2025) adds a separate LoRA branch for each task and learns gates to limit interference. DUALPROMPT (Wang et al., 2022b), S-PROMPTS (Wang et al., 2022a), HIDE-PROMPT (Wang et al., 2023), and mixture-style prefix tuning (Le et al., 2024) use task- or domain-specific prompts, while RANPAC (McDonnell et al., 2023) learns class prototypes after a fixed random projection without updating the backbone. DISeL instead targets the retention–adaptation tradeoff within a *single* fine-tuning task, using input-dependent gates over rank-one components in one LoRA adapter, without task identities or routing labels.

8 CONCLUSION

We introduced DISeL, an FT method that makes low-rank adaptation input-dependent through gates on individual rank-one components. Motivated by the posterior-weighted corrections in our regression framework, this design learns to modulate adaptation across inputs while retaining the low-rank structure of LoRA. Across ROBERTa on GLUE and instruction tuning of LLAMA and MISTRAL for mathematical reasoning and code generation, DISeL achieves competitive FT accuracy while keeping retention close to the base model. Its learned gate patterns also provide an interpretable diagnostic of adapter usage across inputs, layers, and modules, helping characterize where and when adaptation is active.

Limitations and future work. DISeL does not guarantee preservation of all pre-trained capabilities, and its behavior depends on gate initialization, learning rates, and the adapted modules. Its input-dependent updates cannot generally be merged into fixed base weights, so efficient inference remains a challenge. Extending the method beyond single-task FT also requires further study. Future work could combine DISeL with explicit retention objectives, use gate statistics for adaptive rank selection, and explore multi-task and continual learning, alignment, multilingual transfer, and long-context adaptation.

AI USE STATEMENT

In this work, we used generative AI tools for literature review, polishing the text and implementing our method and experiments. The implementation is checked by the main authors and its correctness

is verified. We did not use AI tools for coming up with the main ideas and the method we proposed. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Jack Bandy and Nicholas Vincent. Addressing "documentation debt" in machine learning research: A retrospective datasheet for bookcorpus. *arXiv preprint arXiv:2105.05241*, 2021.

Dan Biderman, Jacob Portes, Jose Javier Gonzalez Ortiz, Mansheej Paul, Philip Greengard, Connor Jennings, Daniel King, Sam Havens, Vitaliy Chiley, Jonathan Frankle, et al. LoRA learns less and forgets less. *Transactions on Machine Learning Research*, 2024.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

Eric L Buehler and Markus J Buehler. X-LoRA: Mixture of low-rank adapter experts, a flexible framework for large language models with applications in protein mechanics and molecular design. *APL Machine Learning*, 2(2), 2024.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Nan Chen, Soledad Villar, and Soufiane Hayou. Learning rate scaling across LoRA ranks and transfer to full finetuning. *arXiv preprint arXiv:2602.06204*, 2026.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 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)*. Association for Computational Linguistics, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? Try ARC, the AI2 reasoning challenge. *arXiv:1803.05457*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2023.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 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)*. Association for Computational Linguistics, 2019.

William B Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005.

Guanting Dong, Hongyi Yuan, Keming Lu, Chengpeng Li, Mingfeng Xue, Dayiheng Liu, Wei Wang, Zheng Yuan, Chang Zhou, and Jingren Zhou. How abilities in large language models are affected by supervised fine-tuning data composition. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 177–198, 2024.

Haonan Dong, Wenhao Zhu, Guojie Song, and Liang Wang. AuroRA: Breaking low-rank bottleneck of LoRA with nonlinear mapping. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2025.

SooHwan Eom, Hee Suk Yoon, Eunseop Yoon, Mark A Hasegawa-Johnson, and Chang D Yoon. Gated LoRA: Dual-purpose projections for parameter-efficient mini-expert fine-tuning. In *Submitted to International Conference on Learning Representations*, 2025.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. LoRA+: Efficient low rank adaptation of large models. In *Proceedings of the 41st International Conference on Machine Learning*. PMLR, 2024.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1026–1034, 2015.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *The Ninth International Conference on Learning Representations*, 2021.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations*, 2022.

Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. LoraHub: Efficient cross-task generalization via dynamic LoRA composition. In *First Conference on Language Modeling*, 2024.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7B, 2023.

Xiaowen Jiang, Xun Wang, and Sebastian U. Stich. LoRAM: Low-rank adaptation of large language models on manifold. In *Sparsity in LLMs (SLLM): Deep Dive into Mixture of Experts, Quantization, Hardware, and Inference*, 2025.

Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What disease does this patient have? A large-scale open domain question answering dataset from medical exams. *arXiv preprint arXiv:2009.13081*, 2020.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pp. 1601–1611, 2017.

Dawid Kopiczko, Tijmen Blankevoort, and Yuki Asano. VeRA: Vector-based random matrix adaptation. In *The Twelfth International Conference on Learning Representations*, 2024.

Minh Le, An Nguyen, Huy Nguyen, Trang Nguyen, Trang Pham, Linh Van Ngo, and Nhat Ho. Mixture of experts meets prompt-based continual learning. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2024.

Yan-Shuo Liang, Jia-Rui Chen, and Wu-Jun Li. Gated integration of low-rank adaptation for continual learning of large language models. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2025.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. DoRA: Weight-decomposed low-rank adaptation. In *Proceedings of the 41st International Conference on Machine Learning*. PMLR, 2024.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *IEEE Transactions on Audio, Speech and Language Processing*, 2025.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. WizardCoder: Empowering code large language models with Evol-Instruct. In *The Twelfth International Conference on Learning Representations*, 2024.

Joel Mackenzie, Rodger Benham, Matthias Petri, Johanne R Trippas, J Shane Culpepper, and Alistair Moffat. CC-News-En: A large English news corpus. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020.

Mark D McDonnell, Dong Gong, Amin Parvaneh, Ehsan Abbasnejad, and Anton Van den Hengel. RanPAC: Random projections and pre-trained models for continual learning. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2023.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. PiSSA: Principal singular values and singular vectors adaptation of large language models. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2024.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *The Fifth International Conference on Learning Representations*, 2017.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? A new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2022.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners, 2019.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392, 2016.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code Llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 2021.

Reece Shuttleworth, Jacob Andreas, Antonio Torralba, and Pratyusha Sharma. LoRA vs full fine-tuning: An illusion of equivalence. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2025.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 2013.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. 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)*. Association for Computational Linguistics, 2019.

Chunlin Tian, Zhan Shi, Zhijiang Guo, Li Li, and Chengzhong Xu. HydraLoRA: An asymmetric LoRA architecture for efficient fine-tuning. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2024.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 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*, pp. 353–355, 2018.

Liyuan Wang, Jingyi Xie, Xingxing Zhang, Mingyi Huang, Hang Su, and Jun Zhu. Hierarchical decomposition of prompt-based continual learning: Rethinking obscured sub-optimality. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2023.

Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(8):5362–5383, 2024a.

Shaowen Wang, Linxi Yu, and Jian Li. LoRA-GA: Low-rank adaptation with gradient approximation. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2024b.

Yabin Wang, Zhiwu Huang, and Xiaopeng Hong. S-prompts learning with pre-trained transformers: An Occam's razor for domain incremental learning. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2022a.

Zifeng Wang, Zizhao Zhang, Sayna Ebrahimi, Ruoxi Sun, Han Zhang, Chen-Yu Lee, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, et al. DualPrompt: Complementary prompting for rehearsal-free continual learning. In *European Conference on Computer Vision*, pp. 631–648. Springer, 2022b.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641, 2019.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. Finetuned language models are zero-shot learners. In *The Tenth International Conference on Learning Representations*, 2022.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering code generation with OSS-Instruct. In *Proceedings of the 41st International Conference on Machine Learning*. PMLR, 2024.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pp. 94–106, 2017.

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)*. Association for Computational Linguistics, 2018.

Xun Wu, Shaohan Huang, and Furu Wei. Mixture of LoRA experts. In *The Twelfth International Conference on Learning Representations*, 2024.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng YU, Zhengying Liu, Yu Zhang, James Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. Mammoth: Building math generalist models through hybrid instruction tuning. In *The Twelfth International Conference on Learning Representations*, 2024.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. AdaLoRA: Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*, 2023.

Appendix

I Proof of Proposition 1

This appendix proves Proposition 1 and characterizes the residual interaction term $\delta_i(\mathbf{x})$. To distinguish a random input from a realized value, we write $\mathbf{X}$ for the random vector and $\mathbf{x}$ for a realization.

Proposition 1 (Bayes-optimal predictor)**.** *For each component, define the residual interaction term*

$$\delta_i(\mathbf{x}) \coloneqq \log \frac{p(\mathbf{x} \mid Z_i = 1)}{p(\mathbf{x} \mid Z_i = 0)} - \boldsymbol{\mu}_i^\top \boldsymbol{\Sigma}^{-1}\mathbf{x},$$

where $p(\mathbf{x} \mid Z_i = s)$ is the conditional density of the input when switch i is fixed to s. Then the minimizer of (4) is

$$f^\star(\mathbf{x}) = \sum_{i=1}^{r} \pi_i(\mathbf{x})\mathbf{M}_i\mathbf{x}, \quad \pi_i(\mathbf{x}) = \Pr(Z_i = 1 \mid \mathbf{x}) = \sigma\Big(\boldsymbol{\mu}_i^\top \boldsymbol{\Sigma}^{-1}\mathbf{x} + \log \tfrac{\alpha_i}{1-\alpha_i} + \delta_i(\mathbf{x})\Big), \quad (5)$$

where $\sigma(t) \coloneqq (1 + e^{-t})^{-1}$.

Proof. The proof has two parts. We first identify the best correction that can be predicted from $\mathbf{X}$, and then calculate the posterior probability that each correction component is required.

The learner observes $\mathbf{X}$ but not Z. Under squared loss, the best prediction of $T(\mathbf{X}, Z)$ from $\mathbf{X}$ is its conditional mean. Adding and subtracting $\mathbb{E}[T(\mathbf{X}, Z) \mid \mathbf{X}]$ inside the squared norm gives

$$\begin{aligned}\mathbb{E}\Big[\|T(\mathbf{X}, Z) - f(\mathbf{X})\|_2^2 \mid \mathbf{X}\Big] &= \mathbb{E}\Big[\|T(\mathbf{X}, Z) - \mathbb{E}[T(\mathbf{X}, Z) \mid \mathbf{X}]\|_2^2 \mid \mathbf{X}\Big] \\ &\quad + \|f(\mathbf{X}) - \mathbb{E}[T(\mathbf{X}, Z) \mid \mathbf{X}]\|_2^2 . \quad (8)\end{aligned}$$

The cross term vanishes because $T(\mathbf{X}, Z) - \mathbb{E}[T(\mathbf{X}, Z) \mid \mathbf{X}]$ has conditional mean zero. The first term in (8) does not depend on f, and the second is minimized at zero. Therefore

$$f^\star(\mathbf{X}) = \mathbb{E}[T(\mathbf{X}, Z) \mid \mathbf{X}]. \quad (9)$$

Using the definition of T in (2), and noting that $\mathbf{M}_i\mathbf{X}$ is fixed after conditioning on $\mathbf{X}$, we obtain

$$f^\star(\mathbf{X}) = \sum_{i=1}^{r} \mathbb{E}[Z_i \mid \mathbf{X}]\mathbf{M}_i\mathbf{X} = \sum_{i=1}^{r} \Pr(Z_i = 1 \mid \mathbf{X})\mathbf{M}_i\mathbf{X}. \quad (10)$$

The second equality follows from $Z_i \in \{0, 1\}$. It remains to calculate the posterior probability for one component.

Fix i and a realized input $\mathbf{X} = \mathbf{x}$. Bayes' rule gives

$$\Pr(Z_i = 1 \mid \mathbf{X} = \mathbf{x}) = \frac{\alpha_i p(\mathbf{x} \mid Z_i = 1)}{\alpha_i p(\mathbf{x} \mid Z_i = 1) + (1 - \alpha_i)p(\mathbf{x} \mid Z_i = 0)}. \quad (11)$$

Rearranging (11) yields

$$\Pr(Z_i = 1 \mid \mathbf{X} = \mathbf{x}) = \sigma\left(\log \frac{\alpha_i}{1 - \alpha_i} + \log \frac{p(\mathbf{x} \mid Z_i = 1)}{p(\mathbf{x} \mid Z_i = 0)}\right). \quad (12)$$

By the definition of $\delta_i(\mathbf{x})$, the likelihood ratio in (12) equals $\boldsymbol{\mu}_i^\top \boldsymbol{\Sigma}^{-1}\mathbf{x} + \delta_i(\mathbf{x})$. Combining this identity with (10) proves the formula in (5).

We next characterize the interaction term. The two likelihoods in (12) are generally not single Gaussian densities. The input is Gaussian only after the complete vector Z has been fixed; conditioning on Z_i alone leaves the other Bernoulli variables random.

Let Z_{-i} collect all switches except Z_i, and fix $\mathbf{z}_{-i} \in \{0, 1\}^{r-1}$. The conditional means associated with $Z_i = 1$ and $Z_i = 0$ are

$$m_{i,1}(\mathbf{z}_{-i}) = \sum_{j \neq i} (z_j - \tfrac{1}{2})\boldsymbol{\mu}_j + \tfrac{1}{2}\boldsymbol{\mu}_i, \qquad m_{i,0}(\mathbf{z}_{-i}) = \sum_{j \neq i} (z_j - \tfrac{1}{2})\boldsymbol{\mu}_j - \tfrac{1}{2}\boldsymbol{\mu}_i. \quad (13)$$

For a mean vector m, denote the Gaussian density with covariance $\mathbf{\Sigma}$ by

$$\phi_{\mathbf{\Sigma}}(\mathbf{x}; m) \coloneqq \frac{1}{(2\pi)^{d_x/2} \det(\mathbf{\Sigma})^{1/2}} \exp\!\left(-\tfrac{1}{2}(\mathbf{x}-m)^\top \mathbf{\Sigma}^{-1}(\mathbf{x}-m)\right). \tag{14}$$

By the law of total probability and independence of the Bernoulli switches,

$$\begin{aligned} p(\mathbf{x} \mid Z_i = 1) &= \sum_{\mathbf{z}_{-i}} \Pr(Z_{-i} = \mathbf{z}_{-i}) \phi_{\mathbf{\Sigma}}(\mathbf{x}; m_{i,1}(\mathbf{z}_{-i})), \\ p(\mathbf{x} \mid Z_i = 0) &= \sum_{\mathbf{z}_{-i}} \Pr(Z_{-i} = \mathbf{z}_{-i}) \phi_{\mathbf{\Sigma}}(\mathbf{x}; m_{i,0}(\mathbf{z}_{-i})), \end{aligned} \tag{15}$$

where each sum contains 2^{r-1} components and

$$\Pr(Z_{-i} = \mathbf{z}_{-i}) = \prod_{j \neq i} \alpha_j^{z_j} (1-\alpha_j)^{1-z_j}. \tag{16}$$

The weights are the same in the two likelihoods because fixing Z_i does not change the distribution of the remaining independent switches. For compactness, define

$$S_{i,s}(\mathbf{x}) \coloneqq \sum_{\mathbf{z}_{-i}} \Pr(Z_{-i} = \mathbf{z}_{-i}) \phi_{\mathbf{\Sigma}}(\mathbf{x}; m_{i,s}(\mathbf{z}_{-i})), \qquad s \in \{0,1\}. \tag{17}$$

Thus, $p(\mathbf{x} \mid Z_i = s) = S_{i,s}(\mathbf{x})$.

We now compare each pair of Gaussian components that shares the same $\mathbf{z}_{-i}$. Their normalization constants cancel, giving

$$\begin{aligned} \log \frac{\phi_{\mathbf{\Sigma}}(\mathbf{x}; m_{i,1}(\mathbf{z}_{-i}))}{\phi_{\mathbf{\Sigma}}(\mathbf{x}; m_{i,0}(\mathbf{z}_{-i}))} &= -\tfrac{1}{2}(\mathbf{x} - m_{i,1}(\mathbf{z}_{-i}))^\top \mathbf{\Sigma}^{-1}(\mathbf{x} - m_{i,1}(\mathbf{z}_{-i})) \\ &\quad + \tfrac{1}{2}(\mathbf{x} - m_{i,0}(\mathbf{z}_{-i}))^\top \mathbf{\Sigma}^{-1}(\mathbf{x} - m_{i,0}(\mathbf{z}_{-i})) \\ &= (m_{i,1}(\mathbf{z}_{-i}) - m_{i,0}(\mathbf{z}_{-i}))^\top \mathbf{\Sigma}^{-1}\mathbf{x} \\ &\quad - \tfrac{1}{2}\left(m_{i,1}(\mathbf{z}_{-i})^\top \mathbf{\Sigma}^{-1} m_{i,1}(\mathbf{z}_{-i}) - m_{i,0}(\mathbf{z}_{-i})^\top \mathbf{\Sigma}^{-1} m_{i,0}(\mathbf{z}_{-i})\right). \end{aligned} \tag{18}$$

Because $\mathbf{\Sigma}^{-1}$ is symmetric, the difference of quadratic forms factors as

$$\begin{aligned} &m_{i,1}(\mathbf{z}_{-i})^\top \mathbf{\Sigma}^{-1} m_{i,1}(\mathbf{z}_{-i}) - m_{i,0}(\mathbf{z}_{-i})^\top \mathbf{\Sigma}^{-1} m_{i,0}(\mathbf{z}_{-i}) \\ &\quad = (m_{i,1}(\mathbf{z}_{-i}) - m_{i,0}(\mathbf{z}_{-i}))^\top \mathbf{\Sigma}^{-1}(m_{i,1}(\mathbf{z}_{-i}) + m_{i,0}(\mathbf{z}_{-i})). \end{aligned} \tag{19}$$

Substituting (19) into (18) gives

$$\log \frac{\phi_{\mathbf{\Sigma}}(\mathbf{x}; m_{i,1}(\mathbf{z}_{-i}))}{\phi_{\mathbf{\Sigma}}(\mathbf{x}; m_{i,0}(\mathbf{z}_{-i}))} = (m_{i,1}(\mathbf{z}_{-i}) - m_{i,0}(\mathbf{z}_{-i}))^\top \mathbf{\Sigma}^{-1} \left(\mathbf{x} - \frac{m_{i,1}(\mathbf{z}_{-i}) + m_{i,0}(\mathbf{z}_{-i})}{2}\right). \tag{20}$$

From (13),

$$m_{i,1}(\mathbf{z}_{-i}) - m_{i,0}(\mathbf{z}_{-i}) = \boldsymbol{\mu}_i, \qquad \frac{m_{i,1}(\mathbf{z}_{-i}) + m_{i,0}(\mathbf{z}_{-i})}{2} = \sum_{j \neq i} (z_j - \tfrac{1}{2})\boldsymbol{\mu}_j. \tag{21}$$

Hence

$$\log \frac{\phi_{\mathbf{\Sigma}}(\mathbf{x}; m_{i,1}(\mathbf{z}_{-i}))}{\phi_{\mathbf{\Sigma}}(\mathbf{x}; m_{i,0}(\mathbf{z}_{-i}))} = \boldsymbol{\mu}_i^\top \mathbf{\Sigma}^{-1}\mathbf{x} - \sum_{j \neq i} (z_j - \tfrac{1}{2}) \boldsymbol{\mu}_i^\top \mathbf{\Sigma}^{-1} \boldsymbol{\mu}_j. \tag{22}$$

Exponentiating (22) gives

$$\begin{aligned} \phi_{\mathbf{\Sigma}}(\mathbf{x}; m_{i,1}(\mathbf{z}_{-i})) &= \exp\left(\boldsymbol{\mu}_i^\top \mathbf{\Sigma}^{-1}\mathbf{x} - \sum_{j \neq i} (z_j - \tfrac{1}{2}) \boldsymbol{\mu}_i^\top \mathbf{\Sigma}^{-1} \boldsymbol{\mu}_j\right) \\ &\quad \times \phi_{\mathbf{\Sigma}}(\mathbf{x}; m_{i,0}(\mathbf{z}_{-i})). \end{aligned} \tag{23}$$

Insert (23) into every term of $S_{i,1}(\mathbf{x})$. The factor $\exp(\boldsymbol{\mu}_i^\top \boldsymbol{\Sigma}^{-1}\mathbf{x})$ does not depend on $\mathbf{z}_{-i}$ and can be moved outside the sum. Therefore

$$S_{i,1}(\mathbf{x}) = \exp(\boldsymbol{\mu}_i^\top \boldsymbol{\Sigma}^{-1}\mathbf{x}) \sum_{\mathbf{z}_{-i}} \Pr(Z_{-i} = \mathbf{z}_{-i}) \times \exp\left(-\sum_{j\neq i} (z_j - \tfrac{1}{2}) \boldsymbol{\mu}_i^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_j \right) \phi_{\boldsymbol{\Sigma}}(\mathbf{x}; m_{i,0}(\mathbf{z}_{-i})). \tag{24}$$

Dividing by $S_{i,0}(\mathbf{x})$ and taking logarithms yields

$$\log \frac{S_{i,1}(\mathbf{x})}{S_{i,0}(\mathbf{x})} = \boldsymbol{\mu}_i^\top \boldsymbol{\Sigma}^{-1}\mathbf{x} + \log \frac{\sum_{\mathbf{z}_{-i}} \Pr(Z_{-i} = \mathbf{z}_{-i}) \exp\left(-\sum_{j\neq i} (z_j - \tfrac{1}{2}) \boldsymbol{\mu}_i^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_j \right) \phi_{\boldsymbol{\Sigma}}(\mathbf{x}; m_{i,0}(\mathbf{z}_{-i}))}{\sum_{\mathbf{z}_{-i}} \Pr(Z_{-i} = \mathbf{z}_{-i}) \phi_{\boldsymbol{\Sigma}}(\mathbf{x}; m_{i,0}(\mathbf{z}_{-i}))}. \tag{25}$$

The logarithm of the ratio of sums in (25) is precisely the residual interaction term $\delta_i(\mathbf{x})$. Since $S_{i,s}(\mathbf{x}) = p(\mathbf{x} \mid Z_i = s)$, we conclude that

$$\log \frac{p(\mathbf{x} \mid Z_i = 1)}{p(\mathbf{x} \mid Z_i = 0)} = \boldsymbol{\mu}_i^\top \boldsymbol{\Sigma}^{-1}\mathbf{x} + \delta_i(\mathbf{x}). \tag{26}$$

This proves the stated likelihood-ratio decomposition and completes the proof. □

II A LIMITATION OF POSITIVELY HOMOGENEOUS GATING

Proposition 2 (Limitation of positively homogeneous corrections)**.** *Let $\mathbf{x} \sim p$ have finite second moment, fix $a > 1$, and define the pre-training and fine-tuning inputs as $\mathbf{x}_{\mathrm{pt}} = \mathbf{x}$ and $\mathbf{x}_{\mathrm{ft}} = a\mathbf{x}$. Suppose the required correction is zero on pre-training inputs and $\mathbf{M}\mathbf{x}_{\mathrm{ft}}$ on fine-tuning inputs, where $\mathbf{M} \in \mathbb{R}^{d_y \times d_x}$. For a measurable correction $f : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$, define*

$$\mathcal{L}_{\mathrm{ft}}(f) := \tfrac{1}{2}\mathbb{E}\left[\|f(a\mathbf{x}) - a\mathbf{M}\mathbf{x}\|_2^2 \right], \qquad \mathcal{L}_{\mathrm{pt}}(f) := \tfrac{1}{2}\mathbb{E}\left[\|f(\mathbf{x})\|_2^2 \right],$$

and let $\eta^2 := \frac{1}{2}\mathbb{E}[\|\mathbf{M}\mathbf{x}\|_2^2] > 0$. Let $\mathcal{H}_{\mathrm{ph}}$ denote the class of positively homogeneous corrections, satisfying $f(t\mathbf{x}) = tf(\mathbf{x})$ for every $\mathbf{x} \in \mathbb{R}^{d_x}$ and $t > 0$. If $f \in \mathcal{H}_{\mathrm{ph}}$, then

$$\boxed{\frac{1}{a}\sqrt{\mathcal{L}_{\mathrm{ft}}(f)} + \sqrt{\mathcal{L}_{\mathrm{pt}}(f)} \geq \eta.} \tag{27}$$

Consequently, if $\mathcal{L}_{\mathrm{pt}}(f) \leq \varepsilon^2$ for some $0 \leq \varepsilon \leq \eta$, then

$$\mathcal{L}_{\mathrm{ft}}(f) \geq a^2(\eta - \varepsilon)^2. \tag{28}$$

The tradeoff is tight: for every $\alpha \in [0, 1]$, the positively homogeneous map $f_\alpha(\mathbf{x}) = \alpha\mathbf{M}\mathbf{x}$ satisfies

$$\mathcal{L}_{\mathrm{ft}}(f_\alpha) = a^2(1-\alpha)^2\eta^2, \qquad \mathcal{L}_{\mathrm{pt}}(f_\alpha) = \alpha^2\eta^2.$$

Moreover,

$$\inf_{f \in \mathcal{H}_{\mathrm{ph}}} \{\mathcal{L}_{\mathrm{ft}}(f) + \mathcal{L}_{\mathrm{pt}}(f)\} = \frac{a^2}{a^2+1}\eta^2. \tag{29}$$

The result applies, in particular, to the top-k and ReLU variants of GATED LORA *at every rank.*

Proof. Positive homogeneity and $\mathbf{x}_{\mathrm{ft}} = a\mathbf{x}$ imply $f(\mathbf{x}_{\mathrm{ft}}) = f(a\mathbf{x}) = af(\mathbf{x})$. Therefore

$$\mathcal{L}_{\mathrm{ft}}(f) = \frac{a^2}{2}\mathbb{E}\left[\|f(\mathbf{x}) - \mathbf{M}\mathbf{x}\|_2^2 \right], \qquad \mathcal{L}_{\mathrm{pt}}(f) = \frac{1}{2}\mathbb{E}\left[\|f(\mathbf{x})\|_2^2 \right]. \tag{30}$$

Starting from the definition of η, add and subtract $f(\mathbf{x})$:

$$2\eta^2 = \mathbb{E}\Big[\|\mathbf{M}\mathbf{x}\|_2^2\Big] = \mathbb{E}\Big[\|\mathbf{M}\mathbf{x} - f(\mathbf{x}) + f(\mathbf{x})\|_2^2\Big].$$

Expanding the square gives

$$\begin{aligned} 2\eta^2 = \mathbb{E}\Big[\|\mathbf{M}\mathbf{x} - f(\mathbf{x})\|_2^2\Big] + \mathbb{E}\Big[\|f(\mathbf{x})\|_2^2\Big] \\ + 2\mathbb{E}\big[(\mathbf{M}\mathbf{x} - f(\mathbf{x}))^\top f(\mathbf{x})\big]. \end{aligned}$$

The Euclidean Cauchy–Schwarz inequality followed by Cauchy–Schwarz for scalar random variables gives

$$\begin{aligned} \mathbb{E}\big[(\mathbf{M}\mathbf{x} - f(\mathbf{x}))^\top f(\mathbf{x})\big] &\le \mathbb{E}[\|\mathbf{M}\mathbf{x} - f(\mathbf{x})\|_2 \, \|f(\mathbf{x})\|_2] \\ &\le \sqrt{\mathbb{E}\Big[\|\mathbf{M}\mathbf{x} - f(\mathbf{x})\|_2^2\Big]}\sqrt{\mathbb{E}\Big[\|f(\mathbf{x})\|_2^2\Big]}. \end{aligned}$$

Substitution into the expansion yields

$$2\eta^2 \le \left(\sqrt{\mathbb{E}\Big[\|\mathbf{M}\mathbf{x} - f(\mathbf{x})\|_2^2\Big]} + \sqrt{\mathbb{E}\Big[\|f(\mathbf{x})\|_2^2\Big]}\right)^2.$$

Using (30), we obtain

$$\begin{aligned} 2\eta^2 &\le \left(\frac{\sqrt{2}}{a}\sqrt{\mathcal{L}_{\mathrm{ft}}(f)} + \sqrt{2}\sqrt{\mathcal{L}_{\mathrm{pt}}(f)}\right)^2 \\ &= 2\left(\frac{1}{a}\sqrt{\mathcal{L}_{\mathrm{ft}}(f)} + \sqrt{\mathcal{L}_{\mathrm{pt}}(f)}\right)^2. \end{aligned}$$

Both sides are nonnegative, so taking square roots and dividing by $\sqrt{2}$ proves (27). If $\mathcal{L}_{\mathrm{pt}}(f) \le \varepsilon^2$, then $\sqrt{\mathcal{L}_{\mathrm{pt}}(f)} \le \varepsilon$. Rearranging (27) gives

$$\sqrt{\mathcal{L}_{\mathrm{ft}}(f)} \ge a(\eta - \varepsilon).$$

Since $0 \le \varepsilon \le \eta$, squaring proves (28).

For tightness, consider $f_\alpha(\mathbf{x}) = \alpha\mathbf{M}\mathbf{x}$. Direct substitution into (30) gives

$$\mathcal{L}_{\mathrm{ft}}(f_\alpha) = a^2(1-\alpha)^2\eta^2, \qquad \mathcal{L}_{\mathrm{pt}}(f_\alpha) = \alpha^2\eta^2.$$

To calculate the minimum total loss over the positively homogeneous class, use the pointwise identity

$$a^2\|\mathbf{u} - \mathbf{v}\|_2^2 + \|\mathbf{u}\|_2^2 = (a^2+1)\left\|\mathbf{u} - \frac{a^2}{a^2+1}\mathbf{v}\right\|_2^2 + \frac{a^2}{a^2+1}\|\mathbf{v}\|_2^2. \tag{31}$$

Apply (31) with $\mathbf{u} = f(\mathbf{x})$ and $\mathbf{v} = \mathbf{M}\mathbf{x}$, take expectations, and divide by two. This gives

$$\begin{aligned} \mathcal{L}_{\mathrm{ft}}(f) + \mathcal{L}_{\mathrm{pt}}(f) &= \frac{a^2+1}{2}\mathbb{E}\left[\left\|f(\mathbf{x}) - \frac{a^2}{a^2+1}\mathbf{M}\mathbf{x}\right\|_2^2\right] + \frac{a^2}{a^2+1}\eta^2 \\ &\ge \frac{a^2}{a^2+1}\eta^2. \end{aligned}$$

Equality is attained by $f(\mathbf{x}) = \frac{a^2}{a^2+1}\mathbf{M}\mathbf{x}$, which is positively homogeneous. This proves (29).

It remains to verify that the result covers the standard top-k and ReLU forms of GATED LORA. For $\mathbf{A} \in \mathbb{R}^{d_y \times r}$ and $\mathbf{B} \in \mathbb{R}^{r \times d_x}$, write the top-$k$ correction as

$$f_{\mathrm{GL}}(\mathbf{x}) = \mathbf{A}\,\mathrm{diag}(g_{\text{top-}k}(\mathbf{B}\mathbf{x}))\,\mathbf{B}\mathbf{x},$$

where $g_{\text{top-}k}$ is the binary mask selecting the k coordinates of $\mathbf{B}\mathbf{x}$ with the largest magnitudes. For every $t > 0$, scaling all coordinates by t preserves their magnitude ordering, so

$$g_{\text{top-}k}(\mathbf{B}(t\mathbf{x})) = g_{\text{top-}k}(\mathbf{B}\mathbf{x}).$$

It follows that

$$f_{\mathrm{GL}}(t\mathbf{x}) = \mathbf{A}\,\mathrm{diag}(g_{\text{top-}k}(\mathbf{B}(t\mathbf{x})))\,\mathbf{B}(t\mathbf{x}) \\ = t\mathbf{A}\,\mathrm{diag}(g_{\text{top-}k}(\mathbf{B}\mathbf{x}))\,\mathbf{B}\mathbf{x} = t f_{\mathrm{GL}}(\mathbf{x}).$$

The ReLU form is also positively homogeneous because $\mathrm{ReLU}(t\mathbf{z}) = t\,\mathrm{ReLU}(\mathbf{z})$ for every $t > 0$:

$$f_{\mathrm{GL}}(t\mathbf{x}) = \mathbf{A}\,\mathrm{ReLU}(\mathbf{B}(t\mathbf{x})) = \mathbf{A}\,\mathrm{ReLU}(t\mathbf{B}\mathbf{x}) = t\mathbf{A}\,\mathrm{ReLU}(\mathbf{B}\mathbf{x}) = t f_{\mathrm{GL}}(\mathbf{x}).$$

Thus both variants are positively homogeneous at every rank, so all claims of the proposition apply to them. □

III Ablation: shared versus per-rank gating

We examine whether a separate gate for each rank-one component is necessary by comparing DISeL with a shared-gate variant. Instead of producing r gate values, the shared router produces one input-dependent scalar per token and adapted module, which multiplies the entire LoRA correction. The gate therefore still varies across inputs, tokens, modules, and layers; it is shared only across rank components within a module.

Both variants use rank 32 and the same training and evaluation settings within each experiment. We evaluate RoBERTa-base on QNLI and Mistral-7B on Magicoder, following the protocols in Appendices IV and V, respectively. The RoBERTa results report mean ± standard deviation across three seeds.

Table 2: Shared versus per-rank gating on RoBERTa-base/QNLI at rank 32. Higher QNLI accuracy is better; the remaining columns report retention perplexity, where lower is better.

Variant	QNLI (%)	WikiText-2	BookCorpus	CC-News
Base RoBERTa	–	7.64	5.94	5.63
Shared DISeL	92.56 ± 0.18	9.49 ± 1.44	7.43 ± 0.23	7.56 ± 0.24
Per-rank DISeL	92.74 ± 0.11	8.84 ± 0.19	7.14 ± 0.22	7.47 ± 0.23

Table 3: Shared versus per-rank gating on Mistral-7B/Magicoder at rank 32. Both columns are percentages; retention is the mean accuracy over the 14 out-of-domain benchmarks. Higher is better.

Variant	HumanEval	Retention
Shared DISeL	61.59	69.36
Per-rank DISeL	63.41	69.62

Per-rank gating improves mean QNLI accuracy by 0.18 percentage points and slightly lowers mean retention perplexity on all three corpora (Table 2). On Mistral, it improves HumanEval by 1.82 percentage points and retention by 0.26 points (Table 3). The shared variant nevertheless remains competitive while using fewer router parameters: in the RoBERTa experiment, sharing reduces the router from 2,656,512 to 83,016 parameters, a 32-fold reduction. These results support per-rank gating as a useful refinement rather than a prerequisite for effective input-dependent adaptation.

IV RoBERTa experiments: details and hyperparameters

This appendix provides the full setup for the RoBERTa-base experiments on five GLUE tasks reported in Section 4.1 of the main paper.

IV.1 Model and tasks

Architecture. We use RoBERTa-base (Liu et al., 2019) (125M parameters) initialized from the HuggingFace `roberta-base` checkpoint. Each task uses a randomly initialized linear classification head trained jointly with the adapter.

Adapter target modules. All adapters (and Full FT, when applicable) act on every linear layer in the encoder:

- Attention: `q_proj`, `k_proj`, `v_proj`, `attention.output.dense`.
- MLP: `intermediate.dense`, `output.dense`.

Tasks. Five GLUE tasks (Wang et al., 2018): MNLI, SST-2, QNLI, CoLA, and MRPC. The underlying task datasets are MULTINLI, SST, SQUAD-derived QNLI, COLA, and MRPC (Williams et al., 2018; Socher et al., 2013; Rajpurkar et al., 2016; Warstadt et al., 2019; Dolan & Brockett, 2005); sample counts and per-task validation metrics are listed in Table 4. Validation accuracy is reported per task; the headline score is their unweighted mean.

Table 4: The five GLUE tasks used for fine-tuning ROBERTa-base.

Task	# train	# classes	Validation metric
MNLI	392,702	3	matched accuracy
SST-2	67,349	2	accuracy
QNLI	104,743	2	accuracy
CoLA	8,551	2	accuracy
MRPC	3,668	2	accuracy

COLA reports accuracy rather than Matthews correlation so that all five tasks share a metric on the same scale, making the cross-task average meaningful.

IV.2 METHODS COMPARED

At every rank $r \in \{4, 8, 16, 32, 64\}$ we train four parameter-efficient methods: LORA (Hu et al., 2022), DORA (Liu et al., 2024), ADALORA (Zhang et al., 2023) (target rank r, initial rank $2r$), and DISeL. The four methods share the same target modules, optimizer, schedule, batch size, sequence length, and number of epochs (Table 5); they differ only in the per-method configuration listed in the next subsection. We additionally train Full FT — the entire encoder plus classification head — as an upper-bound reference; it is not a competing PEFT method, and its per-task hyperparameters are listed separately in Table 7. Each (method, task, rank) cell is run with three seeds; the points in Figure 1 are across-seed means with standard-deviation error bars.

IV.3 COMMON TRAINING HYPERPARAMETERS

The following are shared across all methods, tasks, and ranks unless explicitly overridden below.

IV.4 METHOD-SPECIFIC HYPERPARAMETERS

Rank-scaled learning rate (LORA, DORA, and the LORA branch of DISeL). For these three methods we scale the learning rate with the adapter rank following Shuttleworth et al. (2025).

GATED LORA. We use the ReLU variant of GATED LORA (Eom et al., 2025), without its auxiliary sparsity or orthogonality regularization. Its ranks and learning rates match LORA.

ADALORA. We set the initial rank to $\text{init}_r = 2r$ and prune in a single step to the target rank r. The pruning step is chosen per task, roughly in proportion to the training-set size, giving 37k for MNLI, 10k for QNLI, 6k for SST-2, 800 for COLA, and 350 for MRPC (so $t_{\text{init}} = t_{\text{final}}$). The remaining knobs follow the PEFT defaults: $\Delta_t = 10$, $\alpha = 2r$, and orthogonality-regularization weight 0.5. The learning rate is held at 8×10^{-4} for every task and rank.

DISeL gate hyperparameters. The gate parameters $(\mathbf{W}_g, \mathbf{b}_g)$ are trained jointly with the LORA factors $(\mathbf{A}, \mathbf{B})$. The LORA path uses the rank-scaled learning rate, and the gate uses a separate, larger one. We initialize $\mathbf{W}_g$ with the PEFT default (scaled orthogonal initialization) and tune the

Table 5: Common training hyperparameters shared across all methods, tasks, and ranks.

Hyperparameter	Value
Optimization	
Optimizer	AdamW (β_1=0.9, β_2=0.999, ϵ=10^{-8})
Weight decay	0.01
Max gradient norm	1.0
Learning-rate schedule	
Scheduler	cosine
Warmup ratio	0.02
Batch and sequence	
Per-device batch size	8
Gradient accumulation	2
Effective batch size	16
Max sequence length	512
Precision and hardware	
Precision	bfloat16 mixed precision
GPUs per run	1
Adapter and training duration	
LoRA dropout	0.0
Epochs (adapter methods)	15
Epochs (Full FT)	5 on MNLI/QNLI/SST-2; 10 on CoLA/MRPC

gate bias $\mathbf{b}_g$ and gate learning rate per task; ties on validation accuracy are broken in favor of the configuration with lower forgetting.

Table 6: Per-task DISeL gate hyperparameters.

Task	Gate bias init $\mathbf{b}_g$	Gate LR
MNLI	$-7 \cdot \mathbf{1}$ (uniform across ranks)	1×10^{-4}
SST-2	$-7 \cdot \mathbf{1}$	1×10^{-4}
QNLI	$-7 \cdot \mathbf{1}$	1×10^{-4}
CoLA	-3 at $r \in \{4, 16, 32\}$; -7 at $r \in \{8, 64\}$	1×10^{-3}
MRPC	-3 at $r \in \{4, 16\}$; -7 at $r \in \{8, 32\}$; -5 at r=64	1×10^{-3}

CoLA and MRPC use a larger gate learning rate because their training sets are relatively small. With $\mathrm{gate_lr} = 10^{-4}$, the gates remain close to their near-closed initialization, leading to under-training of the LoRA path and reduced accuracy. Increasing the gate learning rate to 10^{-3}, together with a less negative bias, allows the gates to activate more readily and yields competitive performance. In contrast, for MNLI, QNLI, and SST-2, the smaller gate learning rate is sufficient and results in lower forgetting. A summary is provided in Table 6.

Full FT (reference upper bound). No adapter is used; every encoder parameter and the classification head are trained jointly. All other training hyperparameters match the adapter runs. Per-task LR and epoch count can be found in Table 7.

IV.5 Evaluation protocol

Best-checkpoint selection. We go over all the checkpoints we saved during training and pick the one with the best fine-tuning accuracy. The forgetting accuracy is also measured on this checkpoint.

Table 7: Full FT per-task hyperparameters.

Task	Learning rate	Epochs
MNLI	1×10^{-5}	5
QNLI	1×10^{-5}	5
SST-2	1×10^{-5}	5
CoLA	2×10^{-5}	10
MRPC	2×10^{-5}	10

Fine-tuning score (y-axis of Figure 1). For each (method, rank, seed), the fine-tuning score is the unweighted mean of the five per-task validation accuracies,

$$\text{FT score} = \tfrac{1}{5}\big(\text{MNLI}_{\text{acc}} + \text{SST-2}_{\text{acc}} + \text{CoLA}_{\text{acc}} + \text{QNLI}_{\text{acc}} + \text{MRPC}_{\text{acc}}\big). \tag{32}$$

Forgetting score (x-axis of Figure 1). We measure forgetting as masked-LM perplexity on three held-out corpora. After fine-tuning, the encoder is paired with the original RoBERTa MLM head and evaluated on WikiText-2 (HuggingFace `wikitext`, `wikitext-2-raw-v1`, validation split), BookCorpusOpen (`bookcorpusopen`), and CC-News (`cc_news`); for the latter two we sample 10k sentences with a fixed seed. The figure plots the average of the three perplexities on a log scale. For reference, the pre-trained checkpoint scores 7.64, 5.94, and 5.63 on the three corpora ($\text{ppl}_{\text{base}} \approx 6.40$ overall).

IV.6 Infrastructure

Each run uses a single NVIDIA A100 (40GB) GPU. We implement experiments in Python 3.13.5 with PyTorch (CUDA 13), using the HuggingFace `transformers`, `peft`, and `datasets` libraries, as well as the `lm-evaluation-harness`.

V Llama and Mistral Experiments: Details and Hyperparameters

In this section, we provide the full setup for the LLM experiments reported in Section 4.2.

V.1 Models and training data

We fine-tune two open-weight 7B causal language models on two instruction-tuning corpora; the pairing of base model, corpus, sample count, and prompt template are summarized in Table 8.

Table 8: Base models and instruction-tuning corpora.

Setting	Base model	# training samples	Prompt template
Math	Llama 2-7B (Touvron et al., 2023) (`Llama-2-7b-hf`)	$\sim$395K	Alpaca
Code	Mistral-7B (Jiang et al., 2023) (`Mistral-7B-v0.1`)	$\sim$110K	Magicoder

The math corpus is MetaMathQA (Yu et al., 2024); the code corpus is Magicoder-Evol-Instruct-110K (Wei et al., 2024). Loss is computed only on response tokens; the instruction prefix is masked. We hold out a 5% random validation split (seed 42) and train on the remaining 95%. We do not perform validation-loss-based early stopping; every saved checkpoint is evaluated on the downstream benchmarks (Section V.6).

Prompt templates. The math runs use the standard Alpaca template:

```
### Instruction:
{instruction}
```

```
### Response:
{output}
```

The code runs use the Magicoder template:

```
You are an exceptionally intelligent coding assistant
that consistently delivers accurate and reliable responses
to user instructions.

@@ Instruction
{instruction}

@@ Response
{output}
```

V.2 COMMON TRAINING HYPERPARAMETERS

The following are shared across all methods, ranks, and seeds for both LLAMA and MISTRAL runs unless explicitly overridden in the method-specific section below; values are listed in Table 9.

Table 9: Training hyperparameters shared across all methods, ranks, and seeds.

Hyperparameter	**Value**
Optimization	
Optimizer	AdamW (β_1=0.9, β_2=0.999, ε=10^{-8})
Weight decay	0.01 (excluded from biases, LayerNorm, and gate parameters)
Max gradient norm	1.0
Learning-rate schedule	
Scheduler	cosine
Warmup ratio	0.02
Batch and sequence	
Effective batch size	128 sequences per optimizer step
Max sequence length	512 (math) / 2048 (code)
Precision and hardware	
Precision	bfloat16
Gradient checkpointing	enabled
GPUs per run	4× NVIDIA A100 (40 GB), data-parallel
Training duration	
Epochs	3
Checkpoints saved	2 per epoch (cap 16, oldest pruned)

V.3 ADAPTER CONFIGURATION

The four adapter methods (LoRA, DoRA, AdaLoRA, and DISeL) share the following baseline configuration.

- **Target modules.** Every linear projection inside the decoder blocks: `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, and `down_proj`. The LM head and embeddings are frozen.
- **Bias.** Not trained.
- **Adapter dropout.** 0.0.
- **Scaling.** $\alpha = 2r$.
- **Initialization.** The LoRA factor $\mathbf{A}$ is Kaiming-uniform, $\mathbf{B} = \mathbf{0}$.

V.4 METHOD-SPECIFIC HYPERPARAMETERS

LoRA. Standard LoRA (Hu et al., 2022) as implemented in HuggingFace `peft`: for each pre-trained weight $\mathbf{W} \in \mathbb{R}^{d_y \times d_x}$ in the target set, we learn $\mathbf{\Delta} = \frac{\alpha}{r}\,\mathbf{AB}$ with $\mathbf{A} \in \mathbb{R}^{d_y \times r}$ and $\mathbf{B} \in \mathbb{R}^{r \times d_x}$, matching the notation of Section 3.

GATED LoRA. We use the ReLU variant of GATED LoRA (Eom et al., 2025), without its auxiliary sparsity or orthogonality regularization. Its ranks and learning rates match LoRA (Table 12).

ADALORA. We use the original recipe of Zhang et al. (2023) via HuggingFace `peft`, with initial rank $\text{init}_r = 1.5\,r$ and the budget-reduction schedule of the paper's SQUAD setting (warm-up to $t_{\text{init}} \approx 17\%$ of total steps, pruning ending at $t_{\text{final}} \approx 33\%$, $\Delta_t = 10$, $\beta_1 = \beta_2 = 0.85$); per-setting step counts are given in Table 10. We set the orthogonality-regularization weight to 0.1, again following the original paper, since the PEFT default of 0.5 dominated the training loss in pilot runs. All other adapter settings ($\alpha = 2r$, dropout, target modules) match LoRA. The learning rate is 2×10^{-4} for $r \in \{16, 32, 64, 128\}$ and 1×10^{-4} for $r = 256$ to avoid divergence at $\text{init}_r = 384$.

Table 10: ADALORA budget-reduction schedule per setting. All ranks within a setting share the same total step count.

Setting	Total steps	t_{init}	t_{final}
LLAMA 2 7B / METAMATHQA	8 796	1 574	3 055
MISTRAL 7B / MAGICODER	2 478	420	820

Full FT. All transformer parameters are unfrozen (≈ 6.74B for LLAMA 2 7B and ≈ 7.24B for MISTRAL 7B), including the LM head and the input embeddings. Two learning rates were swept (10^{-5} and 5×10^{-5}); we report 10^{-5} in the main figure (5×10^{-5} was unstable on math). All other settings (optimizer, schedule, batch size, epochs, precision, gradient checkpointing) match the adapter runs.

DISeL gate hyperparameters. For every adapted layer, an input-dependent rank gate is attached to the LoRA branch:

$$\mathbf{\Delta}(\mathbf{x})\,\mathbf{x} \;=\; \frac{\alpha}{r}\,\mathbf{A}\,\mathbf{G}(\mathbf{x})\,\mathbf{B}\,\mathbf{x}, \qquad g(\mathbf{x}) \;=\; \sigma(\mathbf{W}_g\,\mathbf{x} + \mathbf{b}_g) \in (0,1)^r, \tag{33}$$

with $\mathbf{W}_g \in \mathbb{R}^{r \times d_x}$, matching the notation of Section 3. The gate adds $r\,d_x + r$ parameters per adapted layer. The remaining gate-specific hyperparameters (bias and weight initialization, gate normalization, gate learning rate, target modules) are listed in Table 11.

Table 11: DISeL gate hyperparameters used for both LLAMA and MISTRAL.

Gate hyperparameter	Value
Bias init $\mathbf{b}_g$	-3.0 ($\sigma(-3) \approx 0.047$, gates start nearly closed)
Weight init $\mathbf{W}_g$	Kaiming-uniform
Gate normalization	off
Gate learning rate	10^{-3} (separate AdamW group; weight decay disabled)
Targets	identical to LoRA

V.5 PER-RANK LEARNING RATES

For every method we report the learning rate that gave the highest target-task accuracy in our sweep over $\{10^{-4}, 2\times10^{-4}, 4\times10^{-4}\}$ for adapter methods, and $\{10^{-5}, 5\times10^{-5}, 10^{-4}\}$ for Full FT. The effective batch size is held at 128 for every run; per-device batch and gradient-accumulation count are adjusted at the largest ranks (down to per-device 2×16 for MISTRAL DISeL at r=256) to fit

GPU memory. The selected learning rates for every (method, rank, setting) cell are summarized in Table 12.

Table 12: Per-run learning rates for both fine-tuning settings.

Method	**Rank** r	**LLAMA 2 7B (math)**	**MISTRAL 7B (code)**
LoRA, GATED LoRA	16	2×10^{-4}	2×10^{-4}
LoRA, GATED LoRA	32	2×10^{-4}	2×10^{-4}
LoRA, GATED LoRA	64	2×10^{-4}	1×10^{-4}
LoRA, GATED LoRA	128	2×10^{-4}	2×10^{-4}
LoRA, GATED LoRA	256	1×10^{-4}	1×10^{-4}
DoRA	16	2×10^{-4}	2×10^{-4}
DoRA	32	2×10^{-4}	2×10^{-4}
DoRA	64	2×10^{-4}	2×10^{-4}
DoRA	128	2×10^{-4}	2×10^{-4}
DoRA	256	1×10^{-4}	1×10^{-4}
AdaLoRA	16	2×10^{-4}	2×10^{-4}
AdaLoRA	32	2×10^{-4}	2×10^{-4}
AdaLoRA	64	2×10^{-4}	2×10^{-4}
AdaLoRA	128	2×10^{-4}	2×10^{-4}
AdaLoRA	256	1×10^{-4}	1×10^{-4}
DISeL	16	2×10^{-4}	2×10^{-4}
DISeL	32	2×10^{-4}	2×10^{-4}
DISeL	64	2×10^{-4}	2×10^{-4}
DISeL	128	2×10^{-4}	2×10^{-4}
DISeL	256	1×10^{-4}	1×10^{-4}
Full FT	no adapter	1×10^{-5}	1×10^{-5}

V.6 EVALUATION PROTOCOL

All evaluation runs use a single A100 GPU with bfloat16 inference. The adapter is *not* merged into the base weights: the PEFT model is loaded in evaluation mode with all adapter parameters and (for DISeL) the gate parameters restored from disk.

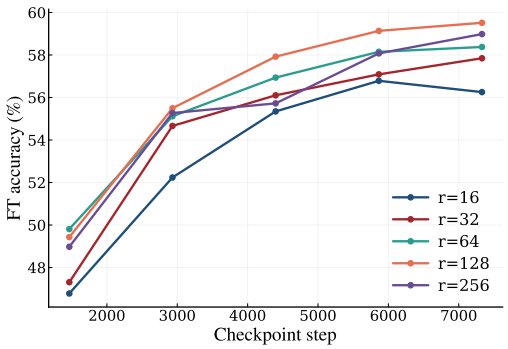
Target-task evaluation.

- **Math (LLAMA 2 7B / METAMATHQA)**: GSM8K, 5-shot, `exact_match,strict-match`.
- **Code (MISTRAL 7B / MAGICODER)**: HUMANEVAL pass@1 with greedy decoding, one sample per problem (164 problems).

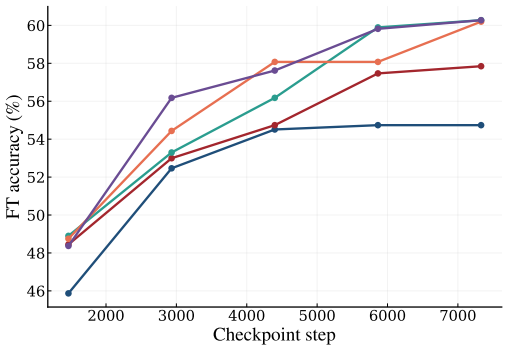
Forgetting suite (both settings). The retention score reported in the main figure is the unweighted mean accuracy across the 14 benchmarks listed in Table 13, all evaluated 5-shot under the same `lm-eval` configuration with `max_length=512`. The same suite is used for both math and code runs.

Table 13: Forgetting (retention) suite: 14 benchmarks evaluated 5-shot under the same `lm-eval` configuration for both LLAMA (math) and MISTRAL (code) runs.

Benchmark	**Category**	**Metric**
HELLASWAG (Zellers et al., 2019)	Commonsense reasoning	`acc_norm`
WINOGRANDE (Sakaguchi et al., 2021)	Commonsense reasoning	`acc`
ARC-CHALLENGE (Clark et al., 2018)	Science QA	`acc_norm`
ARC-EASY (Clark et al., 2018)	Science QA	`acc_norm`
LAMBADA-OPENAI (Radford et al., 2019)	Language modeling	`acc` (and perplexity)
BOOLQ (Clark et al., 2019)	Reading comprehension	`acc`
PIQA (Bisk et al., 2020)	Physical commonsense	`acc_norm`
TRIVIAQA (Joshi et al., 2017)	Open-domain QA	`acc`
OPENBOOKQA (Mihaylov et al., 2018)	Science QA	`acc_norm`
SCIQ (Welbl et al., 2017)	Science QA	`acc`
MMLU (57-task aggregate) (Hendrycks et al., 2021)	Multitask knowledge	`acc`
MEDQA-4OPTIONS (Jin et al., 2020)	Medical QA	`acc`
COMMONSENSEQA (Talmor et al., 2019)	Commonsense reasoning	`acc`
NATURALQUESTIONS-OPEN	Open-domain QA	`acc`

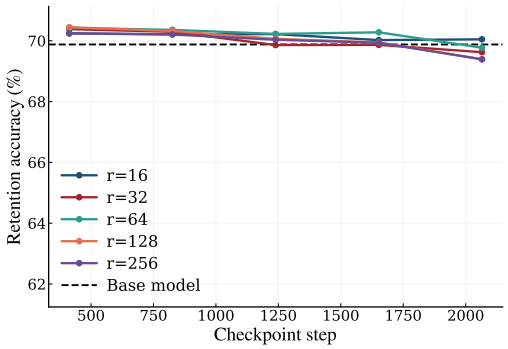


(a) **DISeL:** GSM8K accuracy improves over training across ranks.

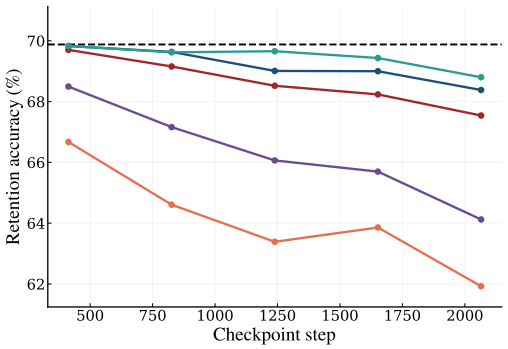


(b) **LoRA:** GSM8K accuracy also improves, but retention degrades at larger ranks.

Figure 5: **Fine-tuning accuracy over training checkpoints for LoRA and DISeL on Llama 2 7B / MetaMathQA.** Each curve reports GSM8K accuracy at every saved checkpoint, for adapter ranks $r \in \{16, 32, 64, 128, 256\}$.



(a) **DISeL:** retention stays close to the base model across ranks and training checkpoints.



(b) **LoRA:** retention degrades over training, with larger drops at higher ranks.

Figure 6: **Retention over training checkpoints for LoRA and DISeL on Mistral 7B / Magicoder.** Each curve tracks the unweighted mean accuracy across the 14-benchmark retention suite at every saved checkpoint, for adapter ranks $r \in \{16, 32, 64, 128, 256\}$.

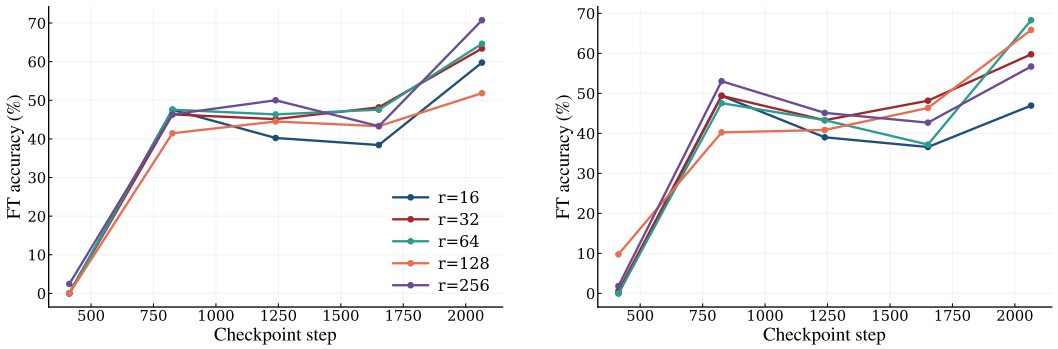
VI Checkpoint Dynamics: Retention and Fine-Tuning Accuracy

VI.1 Llama 2 7B / MetaMathQA

Section 6 reports the checkpoint-level retention curves for the Llama 2 7B / MetaMathQA run. Figure 5 shows the corresponding fine-tuning accuracy curves. Both DISeL and LoRA improve on GSM8K over training, but the retention curves in Figure 4 show that DISeL achieves this without the rank-dependent retention drift observed for LoRA.

VI.2 Mistral 7B / Magicoder

Figures 6 and 7 repeat the checkpoint-level analysis for the Mistral 7B / Magicoder code-generation run. The retention pattern mirrors the Llama math setting in Section 6: DISeL keeps retention close to the base model throughout training and across ranks, whereas LoRA exhibits rank-dependent drift. At the same time, the fine-tuning accuracy curves show that this retention advantage does not come from under-training; DISeL reaches comparable or stronger target-task accuracy at the final checkpoints.



(a) **DISeL:** target-task accuracy improves over training and remains competitive across ranks.

(b) **LoRA:** target-task accuracy also improves, but with larger retention loss at high ranks.

Figure 7: **Fine-tuning accuracy over training checkpoints for LoRA and DISeL on Mistral 7B / MagiCoder.** Each curve reports HumanEval pass@1 at every saved checkpoint, for adapter ranks $r \in \{16, 32, 64, 128, 256\}$.

VII Interpretability: Additional Results

This appendix complements the gate-activation analysis of Section 5, which focused on Llama 2 7B at rank r=32. We report (i) additional Llama 2 7B results across ranks and modules, and (ii) the corresponding analysis on Mistral 7B fine-tuned for code generation. The setup, evaluation domains, and statistics shown (sigmoid gate values aggregated over held-out inputs) are identical to Section 5; only the model, rank, or module under inspection differs.

VII.1 Llama 2 7B: additional ranks and modules

Module analysis. In Figure 8, we fix the input domain to held-out math examples and compare selected adapted layer types: the attention module (v_{proj}), the MLP up projection (up_{proj}), and the MLP down projection ($\text{down}_{\text{proj}}$). Here we see that gate activation is highly non-uniform across different module types. The MLP up projection has the largest gate values, especially in middle and late layers; the down projection shows a weaker version of the same trend; and the value projection remains comparatively closed. This suggests that the learned gates provide a useful diagnostic for where task-specific adapter capacity is used, and may help improve the design of future modules. Taken together, Figures 3 and 8 demonstrate that DISeL learns a conditional routing pattern: gate usage varies across input domains and concentrates in specific parts of the network rather than being distributed uniformly across all ranks, layers, and modules.

We extend the rank-32 analysis of Section 5 along two axes. First, we recompute the domain-conditional gate distributions (math, code, general text) at additional adapter ranks to check that the depth-band pattern reported in the main text is not specific to r=32. Second, we extend the per-module breakdown to the remaining linear projections in the decoder (`q_proj`, `k_proj`, `o_proj`, `gate_proj`), so the v_{proj} / up_{proj} / $\text{down}_{\text{proj}}$ comparison in Figure 8 can be read in context of the full attention and MLP stacks. The qualitative picture is the same as in the main text: math inputs reach the highest gate values, the activation is concentrated in mid-to-late layers, and within those layers, the MLP path (especially up_{proj}) carries most of the active capacity.

Figure 9 first aggregates the adapted projections into attention (`q_proj`, `k_proj`, `v_proj`, `o_proj`) and MLP (`gate_proj`, `up_proj`, `down_proj`) groups on held-out GSM8K inputs. Both groups become more active in middle and late layers, but the MLP gates are consistently more open: the mean gate value rises from about 0.31 in early layers to about 0.50 in middle and late layers, compared with about 0.15 to 0.34 for attention. Figure 10 breaks this down by projection. The largest mean activations occur in `gate_proj`, `up_proj`, and the attention key/query projections in later layers, while `v_proj` and early `down_proj` remain mostly closed. This finer view supports the main-text conclusion that DISeL does not simply open all ranks uniformly: it concentrates capacity in specific depths and projections.

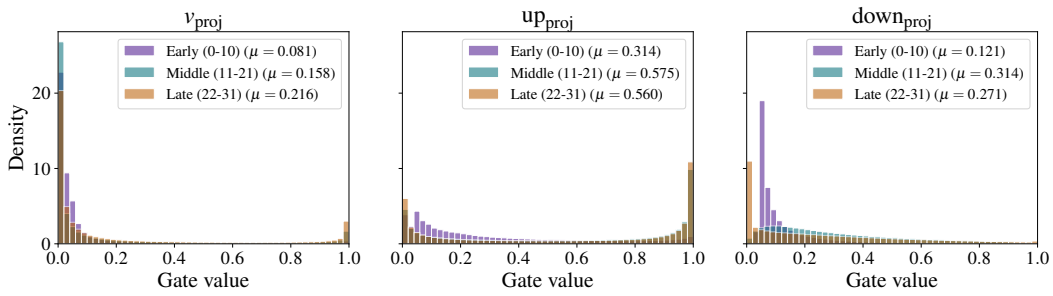


Figure 8: **Layer-wise gate usage within key modules.** We processed held-out math samples with LLAMA using a rank-32 DISeL adapter and measured the output distributions of each module's gates. This figure presents these distributions for the attention value projection (v_{proj}), the MLP up projection (up_{proj}), and the MLP down projection ($\text{down}_{\text{proj}}$). Each panel shows normalized histograms for early, middle, and late layers. In this example, the gates are most open in the MLP up projection, especially in the middle and late layers, while the value and down projections exhibit more moderate activation. This demonstrates that DISeL learns not only when to activate adapter components, but also where in the model those components are most useful.

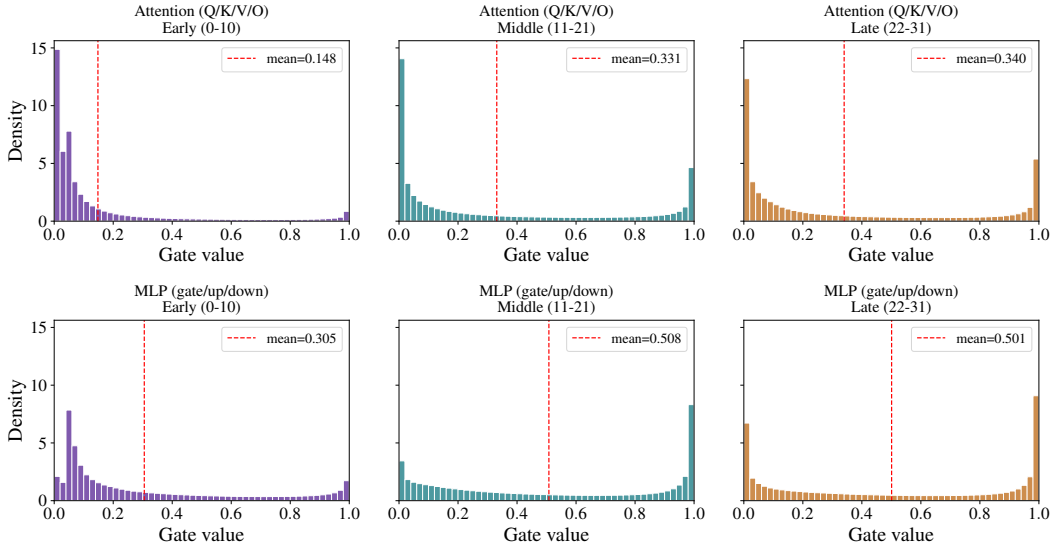


Figure 9: **Gate distributions by module family and depth on LLAMA math inputs.** We aggregate gate values from the rank-32 LLAMA math adapter over held-out GSM8K examples. Attention denotes `q_proj`, `k_proj`, `v_proj`, and `o_proj`; MLP denotes `gate_proj`, `up_proj`, and `down_proj`. Gates open most strongly in the middle and late layers, especially in the MLP stack.

VII.2 MISTRAL 7B FINE-TUNED ON CODE

We repeat the diagnostics of Section 5 on MISTRAL 7B fine-tuned on MAGICODER, using held-out code inputs as the fine-tuning domain and math/general text as contrast domains. Figure 11 shows the main domain-level difference from the LLAMA math adapter. On LLAMA, math inputs are clearly more activated than code, with general text lowest. On MISTRAL fine-tuned for code, code and math are nearly indistinguishable in mean gate value: early layers give 0.128 for code and 0.141 for math, middle layers give 0.224 and 0.235, and late layers give 0.331 and 0.333. However, the histogram shape still reflects the fine-tuning domain: code inputs have the largest mass of gates that are fully open, even though math sometimes has a slightly larger mean because it has more partially open gates. General text remains lower, but the gap is smaller than in the LLAMA math setting. This suggests that the code adapter treats math and code as closely related structured-symbolic inputs, while still opening its highest-confidence gates most often on code.

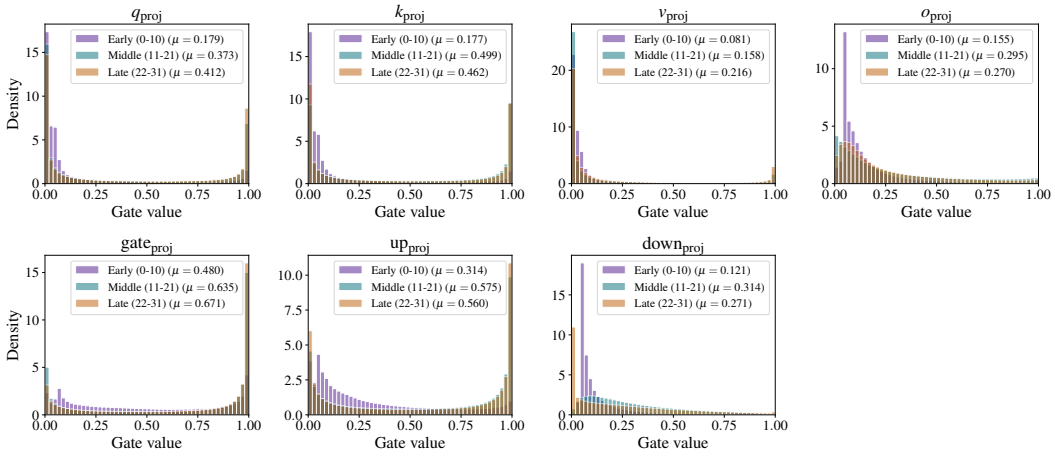


Figure 10: **Gate distributions by projection on LLAMA math inputs.** We show gate-value histograms for all adapted projections of the rank-32 LLAMA math adapter, grouped by layer-depth band. The MLP `gate_proj` and `up_proj` are the most active, while `v_proj` and early `down_proj` remain close to closed.

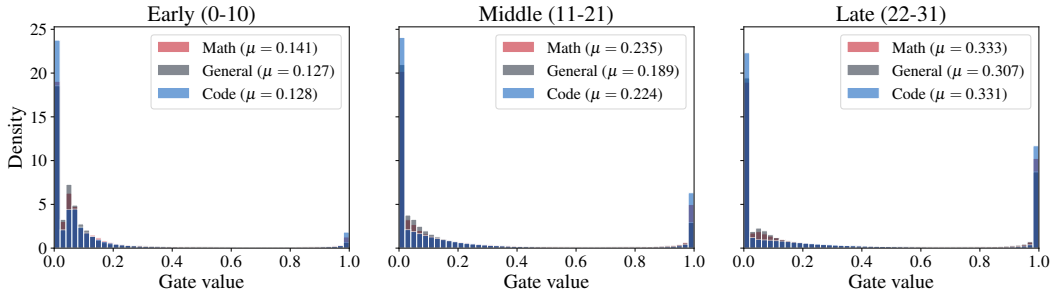


Figure 11: **Gate activation histograms across domains for the MISTRAL code adapter.** We analyze the rank-32 MISTRAL adapter fine-tuned on MAGICODER. Unlike the LLAMA math adapter, where math is clearly separated from code, the MISTRAL code adapter opens gates to a similar degree on held-out code and math inputs, while general text remains lower. Code nevertheless has the largest concentration of fully open gates, even when math has a comparable or slightly higher mean gate value.

The module-level views in Figures 12 and 13 show that the coarse organization is nevertheless similar to LLAMA. Gates open more in the middle and late layers than in the early layers, and the MLP stack is more active than the attention stack. Aggregating by module family, MLP mean activation rises from 0.211 in early layers to 0.416 in late layers, whereas attention rises from 0.066 to 0.267. The per-projection breakdown shows the strongest activations in `gate_proj` and `up_proj`, with `gate_proj` reaching the largest late-layer mean ($\mu \approx 0.686$). Thus, while the domain selectivity differs from LLAMA—code and math open the gates almost equally on MISTRAL—the layer and module selectivity remains consistent.

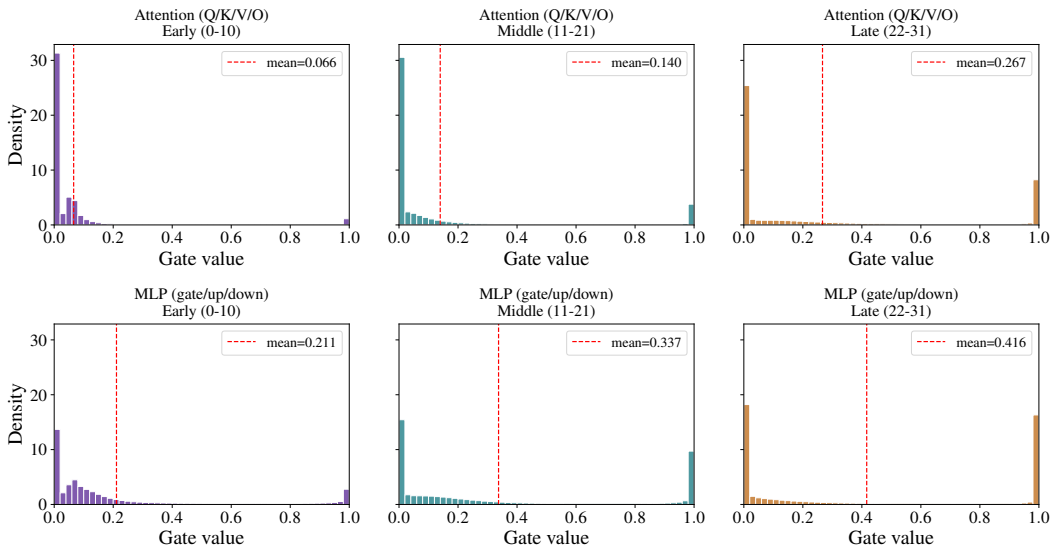


Figure 12: **Gate distributions by module family and depth on MISTRAL code inputs.** Attention denotes `q_proj`, `k_proj`, `v_proj`, and `o_proj`; MLP denotes `gate_proj`, `up_proj`, and `down_proj`. As in the LLAMA analysis, gates open most strongly in the MLP stack and in middle-to-late layers.

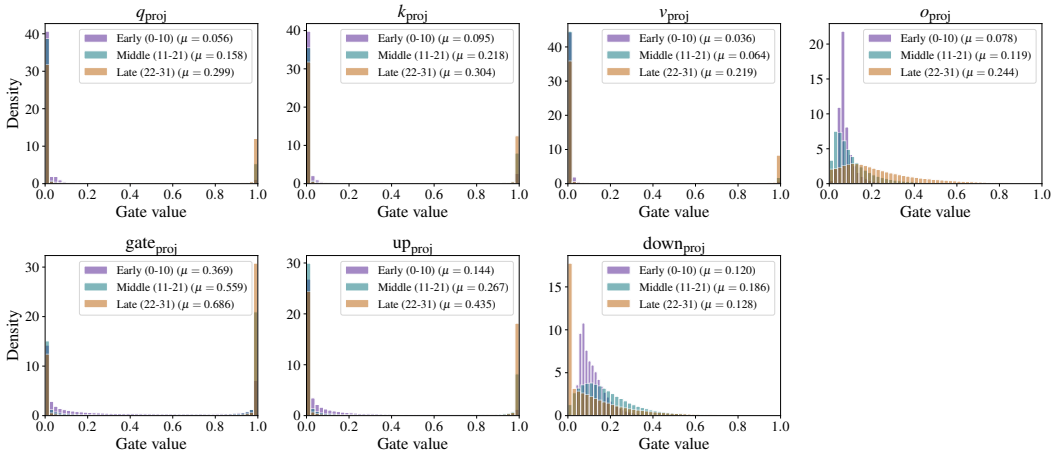


Figure 13: **Gate distributions by projection on MISTRAL code inputs.** We show gate-value histograms for all adapted projections of the rank-32 MISTRAL code adapter, grouped by layer-depth band. The largest activations occur in the MLP `gate_proj` and `up_proj`, while several attention projections remain mostly closed until late layers.