DISSECTING ATTENTION AND MLP ROLES: A STUDY OF DOMAIN SPECIALIZATION IN LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) perform well across diverse domains such as programming, medicine, and law, yet it remains unclear how domain information is represented and distributed within their internal mechanisms. A key open question is the *division of labor* between the Transformer's core components: self-attention and MLP layers. We address this question through a mechanistic study that dissects their roles by integrating three complementary analyses: representation separability via probes, parameter change under adaptation, and causal effects from activation swaps. We propose a clear division of labor: attention layers route domain identity, while MLP layers implement domain-specific computation. Causal interventions strongly support our claims. For instance, swapping attention activations at specific mid-depth layers (e.g., for Python $\leftrightarrow$ C++) reliably shifts the next-token distribution, whereas layers with low domain separability have a negligible effect. In contrast, while finetuning, MLP layers exhibit relatively larger weight changes, consistent with domain-specific knowledge being stored there. This pattern holds consistently across four models and six domains. In a supplementary experiment, we demonstrate that selecting a few components highlighted by our study can accelerate domain adaptation, indicating the potential for more focused fine-tuning.

1 INTRODUCTION

Large Language Models (LLMs) master diverse domains, yet the internal mechanisms governing this domain representation remain an open question. *How does the single monolithic network of blocks pivot from one domain's specialization to another? What is the division of labor between the Transformer's core components – the self-attention and the MLP layers?* In this paper, we address these questions through a causal, layer-level analysis and propose a functional specialization that holds across models and domains.

The field of mechanistic interpretability has developed powerful methods for such analysis, progressing from correlational analysis to causal interventions. Initial probing (Alain & Bengio, 2018; Tenney et al., 2019) analyses used simple neural classifiers to differentiate the outputs of a layer for varied inputs. A highly separable representation of domain identity, for example, would imply that a component contains domain-specific information. The contribution can be quantified by calculating the level of separation in the higher-dimensional space through separability scores, like v-usable information (Ethayarajh et al., 2022) , Xu et al. (2020) , Ju et al. (2024b) Fisher separability Fisher (1936), maximum mean discrepancyGretton et al. (2008) etc. Although these methods can show where information separates, but not if or how the model uses it for downstream tasks.

Subsequently, the focus shifted towards establishing causality by reverse-engineering the circuits (Elhage et al., 2021) for specific behaviors, through methods like activation patching (Meng et al., 2023a) Wang et al. (2022) and zero-out testing (Dai et al., 2021). This research has yielded an important result: MLP layers have been characterized as the primary locus of holding factual knowledge. Concurrently, attention mechanisms are understood as routers, moving and aligning information throughout the context, enabling capabilities like in-context learning (Olsson et al., 2022).

A parallel line of evidence comes from studying parameter adaptation. Research on techniques like Low-Rank Adaptation (LoRA) has shown that model behavior can be improved for targeted adaptation by modifying only a small subset of weights (Hu et al., 2022) Zhang et al. (2023). However, a critical gap remains. These three powerful lenses—representational, causal, and adaptational—have largely been applied in isolation and to micro-scale tasks (e.g., factual recall, syntactic phenomena). It is unknown whether the "attention-as-router, MLP-as-compute" principle scales to govern how models handle high-level, abstract domains like programming or medicine. Furthermore, no existing framework exists to synthesize these three orthogonal sources of evidence into a single, coherent map of a model's functional architecture.

Our work bridges this gap. In our study, we triangulate the function of components of each layer by asking: (1) Is domain information present in its activations? (2) How much do the layer parameters change during adaptation? (3) Does the layer have a causal effect on domain-specific output? The answers to which lie in representational probing, fine-tuning deltas and activation swapping respectively. Here, we qualify that domain control is complex and distributed; no single component type exclusively handles all aspects. Our analysis reveals relative differences in component contributions rather than absolute divisions. In summary, our contributions are:

- **A Unified Methodological Study for Domain Analysis** : We propose and validate three separate sources of evidence - representational separability, parameter changes under adaptation, and causal interventions - to produce a robust, layer-level study of a model's domain-handling mechanisms.
- **Evidence for a Scaled Division of Labor in Abstract Tasks** : We provide direct evidence that the "attention as router, MLP as memory" principle, previously observed in low-level factual tasks, also governs how models handle high-level, abstract capabilities like domain control. This suggests it is a fundamental organizing principle of the Transformer architecture.
- **Demonstration of Mechanistically-Informed, Parameter-Efficient Fine-Tuning** : We show that our mechanistic map can be used to have direct practical utility. Fine-tuning a small subset of components identified by our study as causally important yields equally satisfactory performance, while being trained on much fewer parameters on domain-specific benchmarks compared to fine-tuning the entire model.

2 Proposed Methodology

Our work examines the roles of attention and MLP components across layers through/via three perspectives: representational patterns (Probing analysis), parameter changes (Fine tuning analysis), and causal interventions.

2.1 Probing Analysis

The objective of this experiment is to identify which layers contain the most linearly separable information about domain identity. Classical classification accuracy saturated at around 100% across all layers, providing insufficient discriminative power to determine where domain information is most concentrated. We instead quantify the degree of separability using distributional metrics. A high degree of separability indicates that a layer's activations serve as a strong signal for the domain, a necessary condition for a component involved in routing or high-level control.

To quantify where domain identity is explicitly represented, we compute pairwise separability between domains for each layer and component using two complementary statistics: a scalar Fisher ratio Fisher (1936) and RBF-kernel Maximum Mean Discrepancy (MMD) Gretton et al. (2008). Let $X \in \mathbb{R}^{N\times D}$ be pooled activations for a given (layer, component) and $y \in \{1,\dots,K\}^N$ the domain labels. Denote by X_i the rows of X with label i, $N_i = |X_i|$, and $\mu_i = \frac{1}{N_i}\sum_{x\in X_i} x$.

Fisher : We report the scalar Fisher score between domains i and j:

$$\text{Fisher}_{ij} \;=\; \frac{\|\mu_i - \mu_j\|^2}{\sum_{d=1}^{D}\text{Var}(X_{i,\cdot d}) + \sum_{d=1}^{D}\text{Var}(X_{j,\cdot d}) + \varepsilon},$$

with $\varepsilon = 10^{-6}$ for numerical stability. This ratio is high when domain means are well-separated relative to within-domain variance, indicating linear discriminability.

MMD (RBF). Using an RBF kernel $k_\gamma(x, x') = \exp(-\gamma\|x - x'\|^2)$ we compute

$$\mathrm{MMD}^2_{k_\gamma}(X_i, X_j) = \frac{1}{N_i^2}\sum_{a,b\in X_i} k_\gamma + \frac{1}{N_j^2}\sum_{a,b\in X_j} k_\gamma - \frac{2}{N_i N_j}\sum_{a\in X_i}\sum_{b\in X_j} k_\gamma,$$

and report $\mathrm{MMD}_{ij} = \sqrt{\max(0, \mathrm{MMD}^2)}$. The kernel bandwidth γ is set by the median heuristic on pairwise distances.

Activations are extracted by registered forward hooks at two probe points per block: post-attention and post-MLP (before residual addition). (for details on pipeline see Appendix A.2). We display only Fisher and MMD scores because they capture complementary linear (mean-vs-variance) and nonlinear (higher-moment) distributional differences and provide the clearest layer-wise differentiation in our experiments. Rather than exhaustively reporting all $\binom{K}{2}$ pairwise scores, we compute **a 1-vs-all** statistic for each domain. For a domain D_i, activations from D_i are compared against the pooled activations from all other domains $\bigcup_{j\neq i} D_j$. This yields a per-layer, per-component separability score $S_{i,\ell}$ indicating how well layer ℓ distinguishes D_i from the rest of the corpus. To compare components on the same scale, we z-normalize scores across layers for each domain. Additional metrics (v-usable bits, cosine similarities, accuracy) are analyzed in Appendix D.3 for completeness.

2.2 Fine-tuning analysis

Probing identifies where domain identity is *separated* in activations; the complementary question is where parameters undergo adaptation. We answer this by measuring per-layer parameter updates under fine-tuning and by testing whether the layers that change most are also the layers that suffice for adaptation.

We use LoRA-style fine-tuning for targeted, parameter-efficient adaptation. For a dense weight $W \in \mathbb{R}^{n\times m}$ at layer ℓ the adapted weight is $W + \Delta W_\ell$ with $\Delta W_\ell = \frac{\alpha}{r} B_\ell A_\ell$ where $A_\ell \in \mathbb{R}^{r\times m}$, $B_\ell \in \mathbb{R}^{n\times r}$, r is the adapter rank and α is a scalar scaling. We summarize a layer's adaptivity by the Frobenius norm of the effective update

$$S_\ell = \|\Delta W_\ell\|_F,$$

and aggregate multiple adapter tensors that belong to the same Transformer block by summation: $S_\ell^{\text{block}} = \sum_{t\in T_\ell} \|\Delta W_t\|_F$. A high S_ℓ indicates that the parameters in ℓ layer are a primary site for storing new, domain-specific computation learned during adaptation Gupta et al. (2025).

We run three fine-tuning regimes: (i) full-model fine-tuning (baseline), (ii) LoRA targeted only to attention projection matrices (e.g., q, k, v, o per block), and (iii) LoRA targeted only to MLP projection matrices (e.g., gate/up/down). For domain perplexity evaluation, we additionally fine-tune only the top 1 and top 3 layers under each of these regimes. All fine-tune runs use fixed hyperparameters (epochs, learning rate, batch size, LoRA rank) and multiple random seeds to enable statistical comparison. (See Appendix D.1)

2.3 Causal activation swapping

Probing and fine-tuning establish where domain information is present and where the optimizer writes it; to show that a layer's activations actually *cause* domain-directed generation, we perform activation swapping. The experiment asks: if we transplant the hidden state from a donor prompt in domain D_b into a recipient prompt in domain D_a, does the model's next-token distribution shift toward D_b?

We construct matched prompt pairs $(x_a \in D_a, x_b \in D_b)$ that share a template and differ only in domain-specific tokens (e.g., "Write a short function in {Python/C++} that returns the n-th Fibonacci number. Respond with code only."). We focus our causal analysis on the C++ and Python domain pair as our primary case study. This pair offers several methodological advantages: (1) structural prompt similarity enables precise matched comparisons, (2) distinct tokenization patterns provide clear directional indicators, and (3) both domains require similar computational complexity, isolating domain identity from task difficulty effects. For a chosen layer ℓ and the first code token, we:

1. run a forward pass on the donor x_b and save donor activations $a_\ell^{\text{donor}}(t^\star)$;

2. run a forward pass on the recipient x_a but, at layer ℓ and position $t^\star$, replace the recipient activation with $a_\ell^{\text{donor}}(t^\star)$ and continue inference to obtain the patched distribution $p_{\text{swap}(\ell)}(\cdot \mid x_a)$;
3. repeat across many donor–recipient pairs and average metrics (see D.2).

Metrics. We quantify the effect of a swap with two complementary statistics that capture magnitude and directionality.

(1) KL Divergence. For a donor domain D_b and recipient domain D_a we define

$$\text{KL}_{\text{swap}_\ell}(D_a \leftarrow D_b) \;=\; \mathbb{E}_{x_a \sim D_a}\big[\,\text{KL}\big(p(\cdot \mid x_a) \,\|\, p_{\text{swap}(\ell)}(\cdot \mid x_a)\big)\,\big],$$

where $p(\cdot \mid x_a)$ is the original next-token distribution and $p_{\text{swap}(\ell)}(\cdot \mid x_a)$ is the patched distribution. $\text{KL}_{\text{swap}_\ell}$ measures how strongly the swap perturbs the model's predictive distribution at the intervention point.

(2) Delta bias. We define domain-token sets S_a, S_b (e.g., Python: {`def`, `import`, `:`, `lambda`, `print`}; C++: {`;`, `::`, `std`, `cout`, `#`, `{}`}). For a prompt x_a, let $P(S|x)$ be the probability mass on tokens S. Bias toward D_b is $\text{Bias}(x) = P(S_b|x) - P(S_a|x)$ We measure the change due to intervention as

$$\Delta\text{Bias}(D_a \xleftarrow{\ell} D_b) = \mathbb{E}\Big[\text{Bias}_{swap}(x_a \xleftarrow{\ell} x_b) - \text{Bias}_{base}(x_a)\Big]\,.$$

Positive values indicate a shift toward the donor domain D_b, since bias is always computed as preference of D_b over D_a. For complete details, see Appendix E.3

KL captures whether an intervention meaningfully alters the model's beliefs; the domain-token Shift tests whether the alteration is *directionally* consistent with the donor domain. Together they provide strong, local causal evidence that activations at layer ℓ not only correlate with domain identity but can drive domain-appropriate generation when transplanted into another context. The experimental conditions ensure that trivial scale differences do not drive observed effects. For more implementation details, see Experimental Setup D.2

3 Results and Discussion

Our investigation spans six domains: Medicine, Finance, Science, Mathematics, C++, and Python, and on four LLMs: Llama 3.2 3B, Llama 1B, Gemma 3 4B, and Gemma 3 1B (Grattafiori et al., 2024) (Team et al., 2025).For more details on datasets used, see Appendix B. The following discussion is for the Llama 3.2 3B model, which consists of 27 layers, each with an MLP head and an attention mechanism. For results on other models, see Appendix A.

3.1 Where domain knowledge is separated?

Figure 1 shows the 1-vs-all Fisher and MMD separability traces across layers for six domains, z-normalized to highlight relative variation in depth. Both Attention and MLP components exhibit non-uniform separability: some layers carry markedly stronger domain identity than others. While the overall trends are similar, the precise peaks do not fully coincide between Attention and MLP. This suggests that both components participate in domain representation, but their strongest contributions arise at slightly different depths.

After z-score normalization, Fisher and MMD traces nearly completely overlap across layers. This indicates that both linear mean-based separation (Fisher) and higher-moment distributional divergence (MMD) identify the same loci of domain information. Thus, the observed peaks are not artifacts of a particular separability metric, but reflect genuine structural patterns in the residual stream.

To compare components, Table 1 reports the mean and maximum 1-vs-all separability scores across layers. A clear pattern emerges. The mean separability is comparable between the Attention and MLP layers across all six domains. This suggests that domain-specific information is broadly and similarly represented in the activations of both components throughout the network. The maximum separability, however, tells a different story. For 5 out of 6 domains, the maximum Fisher and MMD scores are higher for Attention layers than for MLP layers. This indicates that while domain information is

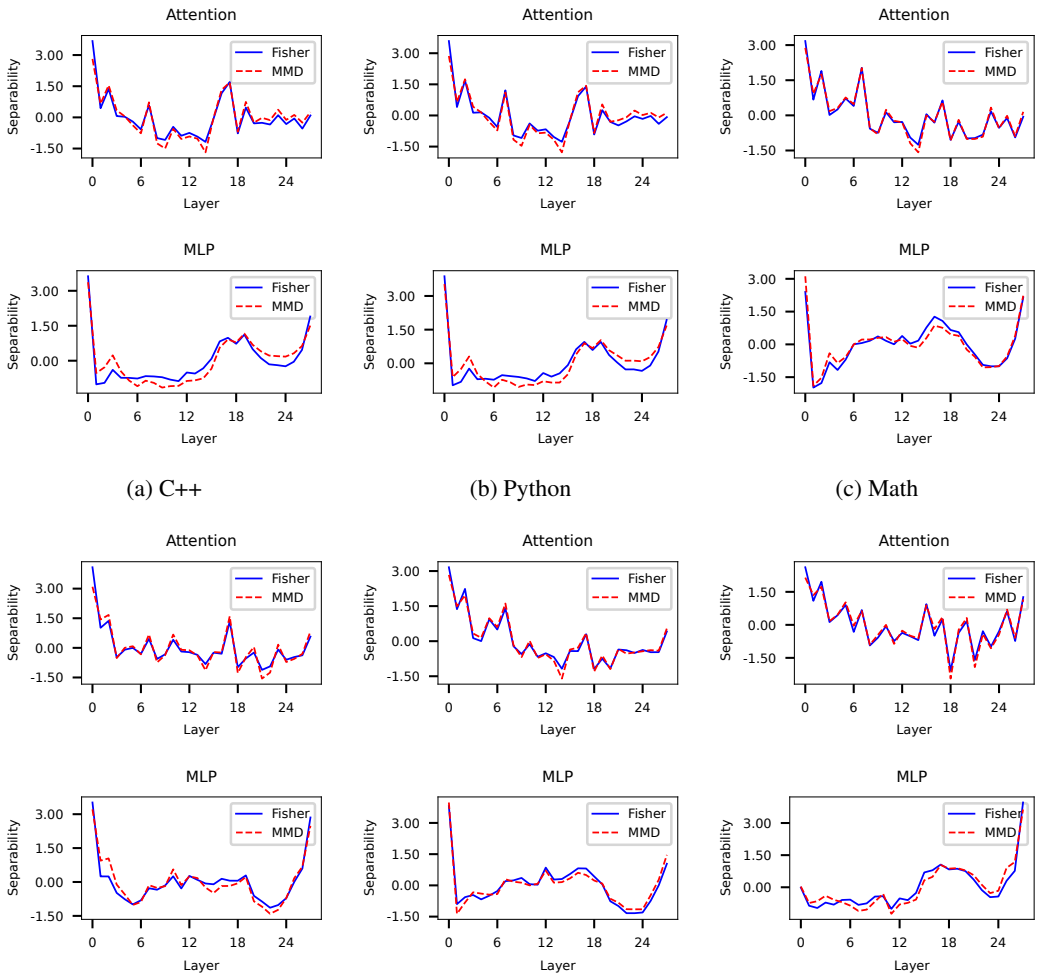


Figure 1: Separability scores across six domains. Each column displays Attention (top) and MLP (bottom) blocks for one domain.

Table 1: Mean and maximum 1-vs-all separability scores for Attention and MLP layers across six domains. Higher values indicate greater domain specificity for that component.

Domain	Attention				MLP			
	Fisher		MMD		Fisher		MMD	
	Mean	Max	Mean	Max	Mean	Max	Mean	Max
CPP	0.935	2.029	0.530	0.678	0.977	1.549	0.532	0.630
Python	0.868	1.783	0.519	0.660	0.912	1.356	0.520	0.615
Math	0.837	1.370	0.551	0.646	0.873	1.015	0.554	0.603
Medical	1.009	2.343	0.586	0.715	1.412	1.983	0.656	0.704
Science	0.849	1.528	0.550	0.662	0.960	1.123	0.567	0.615
Finance	1.696	2.668	0.677	0.741	2.140	3.158	0.726	0.767

generally available, it becomes highly concentrated at specific bottleneck layers within the Attention mechanism. For example, for the C++ domain, the most separable Attention layer (max Fisher=2.029) is 31% more distinct than the most separable MLP layer (max Fisher=1.549).

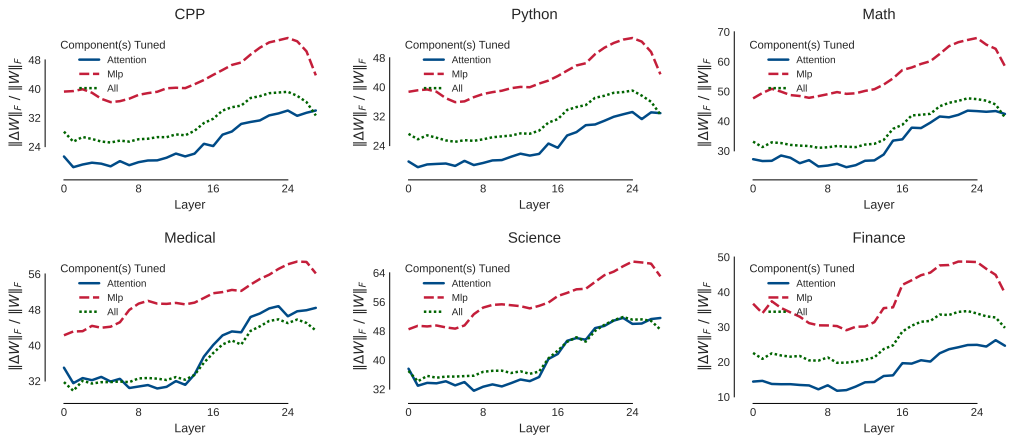


Figure 2: Change in the weights on Lora-based fine-tuning, separately on (1) Entire model, (2) Only Attention Layers, and (3) Only MLP Layers

3.2 Adaptational Analysis Points to MLP Layers

While probing analysis suggests concentrated signals in attention layers, adaptational analysis reveals a different picture. Figure 2 plots the average normalized weight change ($\|\Delta W\|/\|W\|$) per layer for three LoRA fine-tuning regimes: targeting the full model, only MLP components, or only attention components.

The magnitude of weight change in MLP-only fine-tuning is substantially and consistently higher than in attention-only fine-tuning. This indicates that MLP layers are the primary locus where new, domain-specific computation is written during adaptation to a specific domain. The results are unambiguous across all six domains. This implies that while attention layers had concentrated signals due to higher peaks of separability in specific layers, adapting to a new dataset always changes the MLP layers more, proposing that domain-specific knowledge is *stored* in the latter.

3.3 Validating the Proposed Layer Map via Targeted Fine-Tuning

Before performing causal interventions, we first seek to validate the practical utility of our proposed layer map. If the layers, either those with the largest parameter deltas (primarily MLPs) or those with the most separable representations (peak attention layers), are indeed the most important for adaptation, then fine-tuning only these layers should achieve satisfactory results in comparison to fine-tuning the entire model. We test this hypothesis by fine-tuning only the top-1 and the top-3 layers (for both MLP and Attention) with the highest separability scores and comparing their performance on the respective domain's specific perplexity task against fine-tuning the full model. For every domain, the domain perplexity was devised using a benchmark evaluation method, normalized between 0 and 1. Details of domain-specific evaluation are mentioned in Appendix C.2.

Interestingly, the results in Table 2 are even better than expected. Targeted fine-tuning of just the selected few layers (sometimes even 1) achieves domain-specific performance that is comparable to, and in some cases exceeds, that of fine-tuning the entire model, despite using a fraction of the parameters. For more insights, refer to Appendix C.1. The dataset used for fine-tuning had around 5000-7000 samples, as discussed in B.

It is important to note that due to the small scale of the models and limited fine-tuning data, fine-tuning can suffer from some forgetting of general capabilities. However, the relative performance gain across all fine-tuned results demonstrates that our layer importance map successfully identifies the most critical components for specialization. This provides strong evidence that the map is not just descriptive but predictive, *highlighting its potential* for interpretable fine-tuning strategies.

	PT	MLP	Attn	Both	Top-1 MLP	Top-1 Attn	Top-3 MLP	Top-3 Attn
Math	0.07	0.03	0.00	0.02	0.08	0.07	**0.12**	0.03
Science	0.82	0.73	0.80	0.62	0.76	0.76	**0.86**	0.66
CPP	0.31	0.06	0.05	0.04	0.01	0.19	0.02	**0.41**
Python	0.60	0.16	0.36	0.02	0.14	**0.57**	0.19	0.56
Finance	0.16	0.06	0.05	0.02	0.05	0.05	**0.08**	0.06
Medical	0.58	0.91	0.84	0.89	**0.93**	0.30	0.91	0.54

Table 2: Performance of Llama-3.2-3B across domains on that domain-perplexity metric (normalized between 0 and 1). PT stands for pre-trained model. All the other column names resemble the components fine-tuned during adaptation.

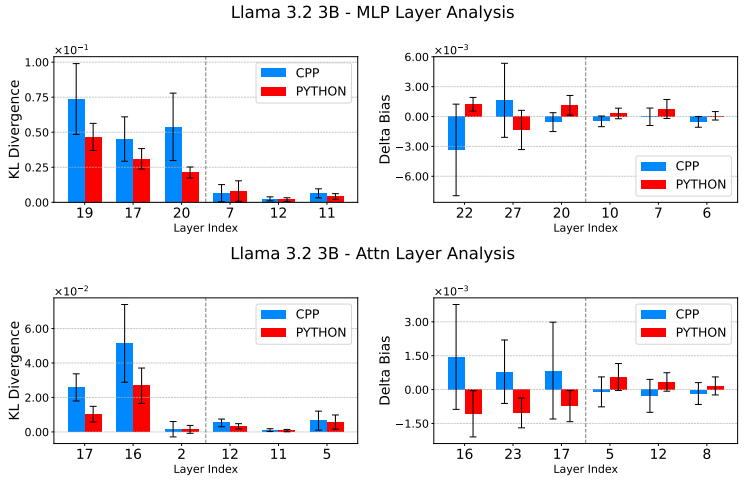


Figure 3: Analysis of layers in Llama-3B, comparing KL divergence (left) and a Delta Bias (right) between C++ and Python inputs. The layers on the left section of both graphs are the top-ranked layers based on Fisher score, while the layers on the right section are layers with the lowest Fisher score.

3.4 Causal Swapping Reveals Attention as Domain Router

Probing identified separable domain representations, and adaptation revealed MLPs as the primary locus of parameter change. To test which components actually *cause* domain-directed behavior, we use activation swapping in our C++/Python case study.

We measure the overall magnitude of the intervention's effect using KL divergence as shown in figure 3. For components in layers with high Fisher separability, swapping activations from either an attention block or an MLP block induces a significant relative perturbation in the next-token distribution, resulting in high KL divergence. This confirms that both components in these layers are computationally active and influential on the final output. Conversely, interventions on components in low-Fisher layers produce a negligible KL divergence, confirming that the effect is localized to the information-rich parts of the network. Early layers occasionally exhibit high Fisher but low causal effect (e.g., Attention layer-2), suggesting the occurrence of "hydra" effect McGrath et al. (2023) (Discussion E.1) here.

However, a disruptive effect does not imply directional control. To test if a layer steers the output towards a specific domain, we measure the shift in probability mass towards the donor domain's characteristic tokens (e.g., Python tokens like def after a Python→C++ swap). For more details, see Discussion E.2. When swapping the output of a high-Fisher attention layer, the effect is strong and, while being highly variational, consistently directional. Swapping Python activations into a C++ prompt increases the probability of Python tokens, and vice-versa. This provides direct causal evidence that these attention layers are not just active, but are providing a specific steering

signal for domain identity. In contrast, swapping the output of a high-Fisher MLP layer does not produce a consistent directional shift. While the intervention is disruptive (high KL), the effect on domain-specific token probability (Delta Bias) is centered around zero, albeit with high variance. This suggests that while the MLP is performing critical, domain-relevant computations, it is not the source of the high-level control signal that dictates "we are in the Python domain now."

On bringing together these observations, we can conclude that the MLP layers change most during fine-tuning because they are the computational workbenches where domain-specific knowledge (e.g., library functions, syntax patterns) is implemented. Intervening on them is disruptive because it interrupts this computation. However, it is the peak attention layers that act as the causal routers. Their activations, though less plastic during fine-tuning, carry the high-level steering signal that *directs* the downstream computational machinery of the MLPs.

4 DISCUSSION

Our investigation began with a foundational question: how does a monolithic network manage distinct domains? By analyzing the three lenses as proposed, we have moved beyond simple observation to a causal, mechanistic explanation. Our results resolve the apparent contradiction between representational and adaptational analyses, revealing a clear and consistent division of labor between the Transformer's core components. Here, we synthesize these findings, discuss their implications for the field, and outline the limitations of our work to chart a path for future research.

Transferability across models. Our findings are not confined to a single checkpoint. We executed all analyses on LLaMA-1B, LLaMA-3.2B, and Gemma 3-1B/4B (See A). The overall pattern holds: attention layers exhibit localized, high-separability peaks that act as causal routers, while MLP layers accumulate the bulk of adaptation updates. Interestingly, Gemma models display sharper, more localized separability in causal swap experiments, with a single attention layer causing large directional effects. This acute localization of causal influence suggests a more specialized, hub-like routing mechanism within Gemma's architecture, suggesting that architectural choices, such as logit soft-capping or normalization, may influence the concentration of domain representation. These findings highlight the need to explore how such architectural decisions affect causal control and domain adaptation, offering a promising direction for future research.

A coherent mechanistic picture. Taken together, our three experiments point to a consistent proposition. Probes show that both Attention and MLP layers encode domain information, but Attention peaks are sharper and more localized. Adaptation analysis shows that MLPs absorb the majority of parameter changes when learning a new domain, functioning as workbenches for computation. Causal swaps reveal that Attention layers provide clean, directional control: transplanting their activations reliably shifts token probabilities toward the donor domain. In the domain level of abstraction, attention acts as the router, steering domain identity, while MLPs implement the downstream computations that realize domain-specific behavior.

Implications. This proposal has two important implications. First, it provides a layer-level map of where to look for domain control in Transformers, guiding mechanistic interpretability beyond micro-circuits to higher-level behaviors. Second, it has practical value: we highlight the potential to identify a small set of components whose targeted adaptation suffices to replicate full-model domain tuning, offering a mechanistically-grounded complement to parameter-efficient fine-tuning.

Limitations and caveats. Our study has several limitations. (i) We focus our causal analysis on C++ vs Python due to their structural similarity and distinct token sets; other domains are noisier and require more refined prompt design. (ii) We adopt a 1-vs-all separability framework, which simplifies analysis but may collapse informative pairwise distinctions between domains. (iii) Our models are relatively small and fine-tuned on modest datasets; effects may differ in larger-scale LLMs with broader training. (iv) Early-layer separability peaks (e.g., A2) did not always yield causal effects, consistent with the hydra effect, where distributed signals do not translate into single-point steering handles. (v) Finally, our causal swaps measure immediate next-token shifts; long-horizon effects and global coherence remain to be tested.

Future directions. These caveats suggest clear paths forward. Future work should extend our work to larger and more diverse models, refine domain prompts beyond code pairs, and analyze per-head

specialization within the identified router layers. A natural next step is to connect layer-level maps to explicit circuit motifs, integrating coarse-grained and fine-grained mechanistic interpretability. On the practical side, our study could be used to guide efficient domain adaptation or controlled editing, narrowing the intervention space to the components that matter most.

5 Related Work

Representation analysis : The use of simple linear classifiers, or probes, to correlate internal activations with linguistic properties marked an early effort to map knowledge in neural networks (Alain & Bengio, 2018; Tenney et al., 2019). This method was quickly refined in response to critiques that high accuracy does not guarantee task-relevance, leading to the development of control methods and more sophisticated layer-wise analyses of information gain (Hewitt & Liang, 2019; Ravichander et al., 2020; Kunz & Kuhlmann, 2022). Applied to contemporary LLMs, these refined techniques have revealed clear knowledge hierarchies: the "Concept Depth" hypothesis posits that complex concepts are processed in deeper layers (Jin et al., 2024), while abstract traits like personality are localized to the middle-to-upper layers (Ju et al., 2024a). The search for greater precision has led to techniques like sparse probing for isolating the specific neurons responsible for a concept (Gurnee et al., 2023), and has connected analysis to action by using probe results to guide targeted edits on model behavior (Li et al., 2024).

Causal interventions: To move from correlation to causation, a central method is activation patching: a family of techniques that swap activations between inputs to measure their causal effect (Vig et al., 2020; Geiger et al., 2021; Heimersheim & Nanda, 2024). Its application to model editing began with locating and updating single facts via ROME (Meng et al., 2022), a process later scaled to thousands of facts with MEMIT (Meng et al., 2023b) and made more efficient by SaLEM (Mishra et al., 2024). The scope of such causal analysis has since expanded beyond discrete facts, used to map the locality of categorical knowledge (Burger et al., 2024) and to reverse-engineer entire computational circuits 'in the wild' (Wang et al., 2022).

Functional Specialization of Transformer Components : Causal analysis reveals a functional specialization between a transformer's primary sub-layers. MLP layers are established as key-value memories that store factual knowledge (Geva et al., 2021), a view substantiated by causal editing (Meng et al., 2022) and shown to hold in multilingual contexts (Fierro et al., 2023). Conversely, attention mechanisms act as dynamic routers, moving information through the residual stream (Elhage et al., 2021; Olsson et al., 2022). This simple dichotomy has evolved into a more nuanced view of integrated knowledge circuits, with work formalizing how attention filters information for MLPs to store (Xu & Chen, 2023) and detailing direct Attention-MLP interactions (Yao et al., 2024; Neo et al., 2024).

Parameter-Efficient Fine-Tuning as a Locus of Knowledge: A parallel line of research frames Parameter-Efficient Fine-Tuning (PEFT) as a mechanistic diagnostic. While foundational methods like Adapter-tuning (Houlsby et al., 2019) and LoRA (Hu et al., 2022) were developed for engineering efficiency, why and where they work has deep mechanistic implications. Analyses suggest LoRA learns low-rank updates that mimic full fine-tuning (Zhang et al., 2023), and critically, that the efficacy of these updates is highly dependent on their layer-wise placement (An et al., 2024; He et al., 2022). This localization principle is further exemplified by methods like LoFiT, which use interpretability to identify and then fine-tune only a sparse subset of task-critical attention heads (Yin et al., 2024).

6 Conclusion

We demonstrate a clear division of labor in Transformers at the high-level scale of complex, real-world domains: attention layers route domain identity, while MLP layers store domain-specific knowledge. This work establishes that the "router-compute" principle—previously observed in low-level tasks—organizes high-level domain specialization across programming, medicine, and other complex domains. By triangulating probing, adaptation, and causal interventions, we provide a definitive functional map: attention layers serve as domain routers that causally steer model behavior, while MLP layers act as domain-specific computational units. This architectural insight provides a blueprint for more interpretable and efficient model adaptation, advancing our understanding of how large language models master diverse capabilities.

REFERENCES

Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes, 2018. URL https://arxiv.org/abs/1610.01644.

Zhaofeng An, Ziyang Wang, Hong-Kyun Li, and Eun-Kyu Park. Layer-domain control in llms. *arXiv preprint arXiv:2410.15858*, 2024.

Christopher Burger, Yifan Hu, and Thai Le. Beyond individual facts: Investigating categorical knowledge locality of taxonomy and meronomy concepts in gpt models. *arXiv preprint arXiv:2404.18820*, 2024.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

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

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021b. URL https://arxiv.org/abs/2110.14168.

Damai Dai, Li Dong, Furu Zheng, Yifei Wang, Hao Zhou, Ke Xu, and Furu Wei. Knowledge neurons in pretrained transformers. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4683–4695, 2021.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, et al. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021.

Kawin Ethayarajh, Yejin Choi, and Swabha Swayamdipta. Understanding dataset difficulty with $\mathcal{V}$-usable information. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 5988–6008. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/ethayarajh22a.html.

Constanza Fierro, Negar Foroutan, Desmond Elliott, and Anders Søgaard. How do multilingual language models remember facts? In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 5555–5567, 2023.

R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2): 179–188, 1936. doi: 10.1111/j.1469-1809.1936.tb02137.x.

Atticus Geiger, Zhengxuan Wu, Christopher Potts, Thomas Icard, and Noah D Goodman. Causal abstractions of neural networks. In *Advances in Neural Information Processing Systems*, volume 34, pp. 21147–21159, 2021.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 5484–5495, 2021.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James

Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Arthur Gretton, Karsten Borgwardt, Malte J. Rasch, Bernhard Scholkopf, and Alexander J. Smola. A kernel method for the two-sample problem, 2008. URL https://arxiv.org/abs/0805.2368.

Akshat Gupta, Christine Fang, Atahan Ozdemir, Maochuan Lu, Ahmed Alaa, Thomas Hartvigsen, and Gopala Anumanchipalli. Norm growth and stability challenges in localized sequential knowledge editing, 2025. URL https://arxiv.org/abs/2502.19416.

Wes Gurnee, Zizheng Beredo, Yonatan Belinkov, Max Tegmark, and Dimitris Bertsimas. Finding neurons in a haystack: Case studies with sparse probing. *arXiv preprint arXiv:2305.01610*, 2023.

Junxian He, Kevin Kwok, Zhibin Zhou, Graham Neubig, and Pengtao Peng. Towards optimal adapter placement for efficient transfer learning. In *International Conference on Learning Representations (ICLR)*, 2022. URL: https://openreview.net/forum?id=RxQOKupaui.

Stefan Heimersheim and Neel Nanda. How to use and interpret activation patching. *arXiv preprint arXiv:2404.15255*, 2024.

John Hewitt and Percy Liang. Designing and interpreting probes with control tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 2733–2743, 2019.

Neil Houlsby, Andrei Giurgiu, Stanisław Jastrzeńbski, Bruna Morrone, Quentin de Laroussilhe, Alberto Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

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 *International Conference on Learning Representations (ICLR)*, 2022.

Siming Huang, Tianhao Cheng, Jason Klein Liu, Jiaran Hao, Liuyihan Song, Yang Xu, J. Yang, J. H. Liu, Chenchen Zhang, Linzheng Chai, Ruifeng Yuan, Zhaoxiang Zhang, Jie Fu, Qian Liu, Ge Zhang, Zili Wang, Yuan Qi, Yinghui Xu, and Wei Chu. Opencoder: The open cookbook for top-tier code large language models. 2024. URL https://arxiv.org/pdf/2411.04905.

Mingyu Jin, Qisheng Chen, Xiao Zhang, Jiahua Li, Yequan Liu, Wenyu Liu, Beichen Wang, Zhaofeng Yang, Shuai Wang, and Yongfeng Zhang. Exploring concept depth: How large language models acquire knowledge at different layers? *arXiv preprint arXiv:2402.13289*, 2024.

Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William W. Cohen, and Xinghua Lu. Pubmedqa: A dataset for biomedical research question answering, 2019. URL https://arxiv.org/abs/1909.06146.

Matt Gardner Johannes Welbl, Nelson F. Liu. Crowdsourcing multiple choice science questions. 2017.

Tianjie Ju, Zhenyu Shao, Bowen Wang, Yujia Chen, Zhuosheng Zhang, Hao Fei, Mong-Li Lee, Wynne Hsu, Sufeng Duan, and Gongshen Liu. Probing then editing response personality of large language models. *arXiv preprint arXiv:2405.19522*, 2024a.

Tianjie Ju, Weiwei Sun, Wei Du, Xinwei Yuan, Zhaochun Ren, and Gongshen Liu. How large language models encode context knowledge? a layer-wise probing study, 2024b. URL https://arxiv.org/abs/2402.16061.

Jenny Kunz and Marco Kuhlmann. Where does linguistic information emerge in neural language models? measuring gains and contributions across layers. In *Proceedings of the 29th International Conference on Computational Linguistics*, pp. 4976–4988, 2022.

Yufan Li, Zongyi Ji, Huibing Duan, and Anthony Zhou. Probing then editing response personality of large language models. *arXiv preprint arXiv:2404.09849*, 2024.

Spencer Mateega, Carlos Georgescu, and Danny Tang. Financeqa: A benchmark for evaluating financial analysis capabilities of large language models, 2025. URL https://arxiv.org/abs/2501.18062.

Thomas McGrath, Matthew Rahtz, Janos Kramar, Vladimir Mikulik, and Shane Legg. The hydra effect: Emergent self-repair in language model computations, 2023. URL https://arxiv.org/abs/2307.15771.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. In *Advances in Neural Information Processing Systems*, volume 35, pp. 17359–17372, 2022.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt, 2023a. URL https://arxiv.org/abs/2202.05262.

Kevin Meng, Yonatan Belinkov, and David Bau. Mass-editing memory in a transformer. In *International Conference on Learning Representations*, 2023b.

Kshitij Mishra, Aniket Singh, Ankur P Parikh, and Anoop Kumar. Correcting language model outputs by editing salient layers. *Findings of the Association for Computational Linguistics: EMNLP 2024*, 2024.

Clement Neo, Shay B Cohen, and Fazl Barez. Interpreting context look-ups in transformers: Investigating attention-mlp interactions. *arXiv preprint arXiv:2405.02839*, 2024.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

Abhilasha Ravichander, Yonatan Belinkov, and Eduard Hovy. Probing the probing paradigm: Does probing accuracy entail task relevance? In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pp. 3109–3119, 2020.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, Francesco Visin, Kathleen Kenealy, Lucas Beyer, Xiaohai Zhai, Anton Tsitsulin, Robert Busa-Fekete, Alex Feng, Noveen Sachdeva, Benjamin Coleman, Yi Gao, Basil Mustafa, Iain Barr, Emilio Parisotto, David Tian, Matan Eyal, Colin Cherry, Jan-Thorsten Peter, Danila Sinopalnikov, Surya Bhupatiraju, Rishabh Agarwal, Mehran Kazemi, Dan Malkin, Ravin Kumar, David Vilar, Idan Brusilovsky, Jiaming Luo, Andreas Steiner, Abe Friesen, Abhanshu Sharma, Abheesht Sharma, Adi Mayrav Gilady, Adrian Goedeckemeyer, Alaa Saade, Alex Feng, Alexander Kolesnikov, Alexei Bendebury, Alvin Abdagic, Amit Vadi, András György, André Susano Pinto, Anil Das, Ankur Bapna, Antoine Miech, Antoine Yang, Antonia Paterson, Ashish Shenoy, Ayan Chakrabarti, Bilal Piot, Bo Wu, Bobak Shahriari, Bryce Petrini, Charlie Chen, Charline Le Lan, Christopher A. Choquette-Choo, CJ Carey, Cormac Brick, Daniel Deutsch, Danielle Eisenbud, Dee Cattle, Derek Cheng, Dimitris Paparas, Divyashree Shivakumar Sreepathihalli, Doug Reid, Dustin Tran, Dustin Zelle, Eric Noland, Erwin Huizenga, Eugene Kharitonov, Frederick Liu, Gagik Amirkhanyan, Glenn Cameron, Hadi Hashemi, Hanna Klimczak-Plucińska, Harman Singh, Harsh Mehta, Harshal Tushar Lehri, Hussein Hazimeh, Ian Ballantyne, Idan Szpektor, Ivan Nardini, Jean Pouget-Abadie, Jetha Chan, Joe Stanton, John Wieting, Jonathan Lai, Jordi Orbay, Joseph Fernandez, Josh Newlan, Ju yeong Ji, Jyotinder Singh, Kat Black, Kathy Yu, Kevin Hui, Kiran Vodrahalli, Klaus Greff, Linhai Qiu, Marcella Valentine, Marina Coelho, Marvin Ritter, Matt Hoffman, Matthew Watson, Mayank Chaturvedi, Michael Moynihan, Min Ma, Nabila Babar, Natasha Noy, Nathan Byrd, Nick Roy, Nikola Momchev, Nilay Chauhan, Noveen Sachdeva, Oskar Bunyan, Pankil Botarda, Paul Caron, Paul Kishan Rubenstein, Phil Culliton, Philipp Schmid, Pier Giuseppe Sessa, Pingmei Xu, Piotr Stanczyk, Pouya Tafti, Rakesh Shivanna, Renjie Wu, Renke Pan, Reza Rokni, Rob Willoughby, Rohith Vallu, Ryan Mullins, Sammy Jerome, Sara Smoot, Sertan Girgin, Shariq Iqbal, Shashir Reddy, Shruti Sheth, Siim Põder, Sijal Bhatnagar, Sindhu Raghuram Panyam, Sivan Eiger, Susan Zhang, Tianqi Liu, Trevor Yacovone, Tyler Liechty, Uday Kalra, Utku Evci, Vedant Misra, Vincent Roseberry, Vlad Feinberg, Vlad Kolesnikov, Woohyun Han, Woosuk Kwon, Xi Chen, Yinlam Chow, Yuvein Zhu, Zichuan Wei, Zoltan Egyed, Victor Cotruta, Minh Giang, Phoebe Kirk, Anand Rao, Kat Black, Nabila Babar, Jessica Lo, Erica Moreira, Luiz Gustavo Martins, Omar Sanseviero, Lucas Gonzalez, Zach Gleicher, Tris Warkentin, Vahab Mirrokni, Evan Senter, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, Yossi Matias, D. Sculley, Slav Petrov, Noah Fiedel, Noam Shazeer, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Jean-Baptiste Alayrac, Rohan Anil, Dmitry, Lepikhin, Sebastian Borgeaud, Olivier Bachem, Armand Joulin, Alek Andreev, Cassidy Hardin, Robert Dadashi, and Léonard Hussenot. Gemma 3 technical report, 2025. URL https://arxiv.org/abs/2503.19786.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. BERT rediscovers the classical NLP pipeline. In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4593–4601, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1452. URL https://aclanthology.org/P19-1452/.

Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Yaron Singer, and Stuart Shieber. Investigating gender bias in language models using causal mediation analysis. In *Advances in Neural Information Processing Systems*, volume 33, pp. 12388–12401, 2020.

Kevin Wang, Vatsal Varma, Neel Nanda, Jacob Steinhardt, and Catherine Ebel. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small. *arXiv preprint arXiv:2211.00593*, 2022.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions, 2017. URL https://arxiv.org/abs/1707.06209.

Ruichen Xu and Kexin Chen. Filtering with self-attention and storing with mlp: One-layer transformers can provably acquire and extract knowledge. *arXiv preprint arXiv:2310.11495*, 2023.

Yilun Xu, Shengjia Zhao, Jiaming Song, Russell Stewart, and Stefano Ermon. A theory of usable information under computational constraints. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=r1eBeyHFDH.

Yunzhi Yao, Ningyu Zhang, Zekun Xi, Mengru Wang, Ziwen Xu, Shumin Deng, and Huajun Chen. Knowledge circuits in pretrained transformers. *arXiv preprint arXiv:2404.14358*, 2024.

Fangcong Yin, Xi Ye, and Greg Durrett. LoFiT: Localized fine-tuning on LLM representations. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2024.

Zihan Zhang, Ming Li, and Yang Liu. Understanding the mechanism of low-rank adaptation. *arXiv preprint arXiv:2304.01933*, 2023.

A Results on Other Models

A.1 Fine Tuning Analysis

Stage 1: Comprehensive Adaptational Mapping. The initial stage conducted a broad, component-wise analysis for each of the six domains independently. To map the division of labor between Transformer components, we applied LoRA adapters under three distinct regimes:

- **Attention-Only**: LoRA was applied exclusively to the attention projection matrices (`q_proj, k_proj, v_proj, o_proj`) in every layer.
- **MLP-Only**: LoRA was applied exclusively to the MLP projection matrices (`gate_proj, up_proj, down_proj`) in every layer.
- **Full Model (All)**: LoRA was applied to all attention and MLP components simultaneously, establishing a baseline for unconstrained, full-model adaptation.

The primary objective of this stage was to quantify the magnitude of parameter updates for each component $c \in \{\text{Attn, MLP}\}$ at each layer ℓ, measured by the Frobenius norm of the effective weight change, $S_{\ell,c} = \|\Delta W_{\ell,c}\|_F$. The results from this analysis provide the data for the adaptational plots in the main paper (Figure 2) and this appendix.

Stage 1 Results for Other Models

The adaptational patterns observed in the Llama 3.2 3B model hold consistently across other model families and sizes, as shown below.

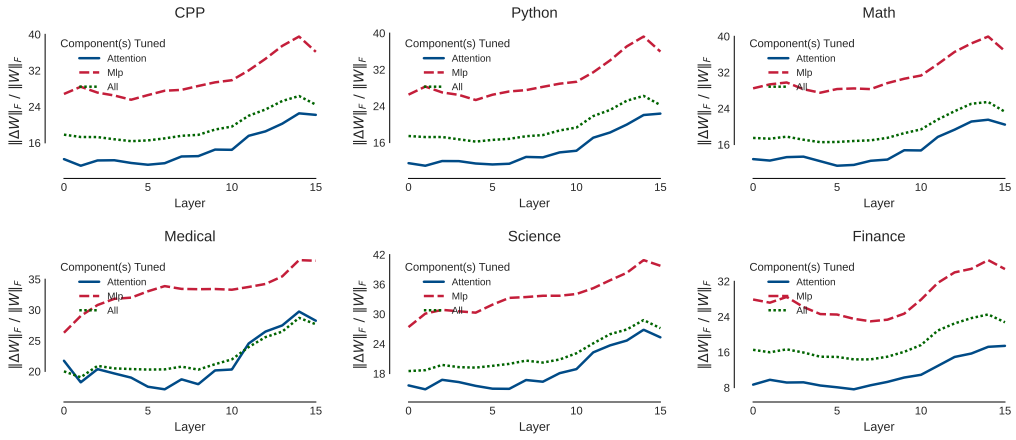


Figure 4: Layer-wise magnitude of parameter updates (S_ℓ) for **Llama 3.2 3B** under three LoRA fine-tuning regimes across six domains.

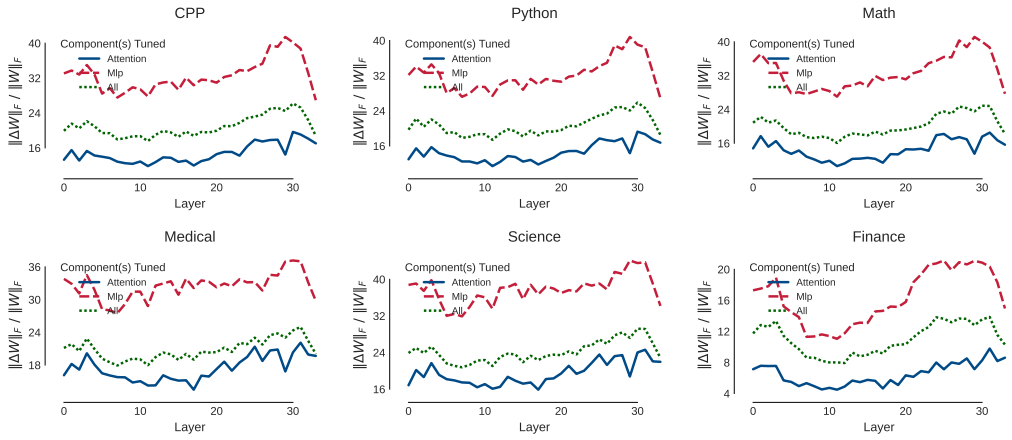


Figure 5: Layer-wise magnitude of parameter updates (S_ℓ) for **Gemma 3 4B** under three LoRA fine-tuning regimes across six domains.

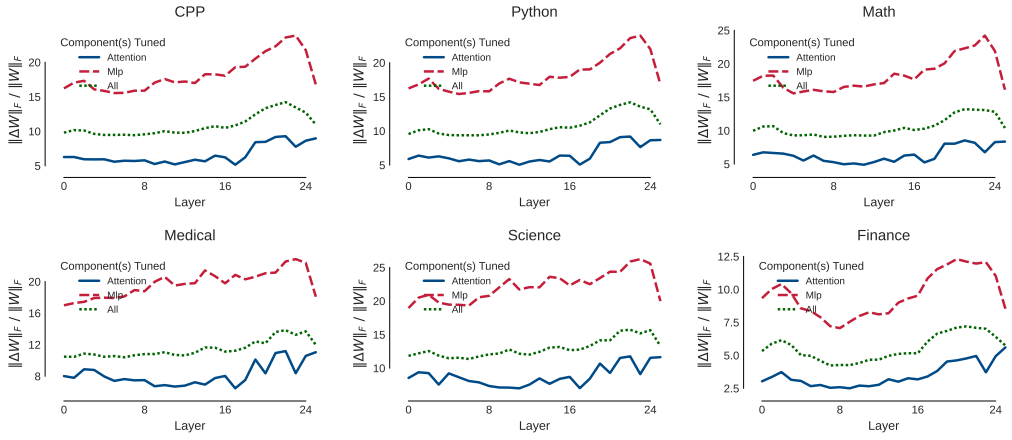


Figure 6: Layer-wise magnitude of parameter updates (S_ℓ) for **Gemma 3 1B** under three LoRA fine-tuning regimes across six domains.

ADAPTATIONAL NORM ANALYSIS

To dissect the dynamics of targeted adaptation, we compare the norms of LoRA weight updates ($\|\Delta W_\ell\|_F$) for the top-3 most-adapted layers across three analytical contexts. The summary tables aggregate these norms to reveal overarching patterns.

- **Avg. Full Run Norm**: The average norm of a component group (e.g., Top-3 MLPs) from the Stage 1 "Full Model" regime, where all layers were adapted on a single domain. This represents the baseline update magnitude in an unconstrained setting.
- **Avg. Ensemble Norm**: The average norm of a component group from a Stage 2 "Ensemble Tuning" run, where *only* those specific components (e.g., only the Top-3 MLP layers) were adapted. This measures the update magnitude under targeted, multi-component fine-tuning.
- **Top Solo Run Norm**: The norm of the single highest-ranking component from a Stage 2 "Soloist Tuning" run, where it was the *only* component adapted in the entire model. This quantifies a component's adaptational capacity in complete isolation.

Table 3: Aggregated LoRA weight update norms for the Llama 3.2 3B model across all domains.

Domain	Component Group	Avg. Full Run Norm	Avg. Ensemble Norm	Top Solo Run Norm
CPP	Top-3 MLP Components (Avg.)	1.019×10^2	1.287×10^2	1.651×10^2
	Top-3 Attn Components (Avg.)	7.042×10^1	9.357×10^1	1.149×10^2
Finance	Top-3 MLP Components (Avg.)	9.463×10^1	6.990×10^1	9.945×10^1
	Top-3 Attn Components (Avg.)	5.464×10^1	4.687×10^1	6.056×10^1
Math	Top-3 MLP Components (Avg.)	1.007×10^2	1.360×10^2	1.676×10^2
	Top-3 Attn Components (Avg.)	6.580×10^1	8.377×10^1	9.766×10^1
Medical	Top-3 MLP Components (Avg.)	9.545×10^1	1.239×10^2	1.560×10^2
	Top-3 Attn Components (Avg.)	9.134×10^1	9.702×10^1	1.181×10^2
Python	Top-3 MLP Components (Avg.)	1.010×10^2	1.311×10^2	1.744×10^2
	Top-3 Attn Components (Avg.)	6.978×10^1	9.599×10^1	1.250×10^2
Science	Top-3 MLP Components (Avg.)	1.019×10^2	1.343×10^2	1.660×10^2
	Top-3 Attn Components (Avg.)	8.013×10^1	9.999×10^1	1.145×10^2

Table 4: Aggregated LoRA weight update norms for the Llama 3.2 1B model across all domains.

Domain	Component Group	Avg. Full Run Norm	Avg. Ensemble Norm	Top Solo Run Norm
CPP	Top-3 MLP Components (Avg.)	1.131×10^2	1.508×10^2	1.944×10^2
	Top-3 Attn Components (Avg.)	8.660×10^1	1.150×10^2	1.389×10^2
Finance	Top-3 MLP Components (Avg.)	1.063×10^2	8.506×10^1	1.201×10^2
	Top-3 Attn Components (Avg.)	6.711×10^1	5.694×10^1	7.221×10^1
Math	Top-3 MLP Components (Avg.)	1.151×10^2	1.607×10^2	2.025×10^2
	Top-3 Attn Components (Avg.)	8.405×10^1	1.062×10^2	1.209×10^2
Medical	Top-3 MLP Components (Avg.)	1.116×10^2	1.457×10^2	1.798×10^2
	Top-3 Attn Components (Avg.)	1.139×10^2	1.196×10^2	1.402×10^2
Python	Top-3 MLP Components (Avg.)	1.123×10^2	1.535×10^2	2.042×10^2
	Top-3 Attn Components (Avg.)	8.578×10^1	1.173×10^2	1.493×10^2
Science	Top-3 MLP Components (Avg.)	1.189×10^2	1.610×10^2	2.009×10^2
	Top-3 Attn Components (Avg.)	1.024×10^2	1.278×10^2	1.446×10^2

Table 5: Aggregated LoRA weight update norms for the Gemma-3 4B model across all domains.

Domain	Component Group	Avg. Full Run Norm	Avg. Ensemble Norm	Top Solo Run Norm
CPP	Top-3 MLP Components (Avg.)	7.481×10^1	8.510×10^1	9.509×10^1
	Top-3 Attn Components (Avg.)	4.523×10^1	5.179×10^1	6.092×10^1
Finance	Top-3 MLP Components (Avg.)	3.211×10^1	3.883×10^1	4.720×10^1
	Top-3 Attn Components (Avg.)	2.398×10^1	2.806×10^1	3.566×10^1
Math	Top-3 MLP Components (Avg.)	6.152×10^1	7.033×10^1	7.748×10^1
	Top-3 Attn Components (Avg.)	3.345×10^1	3.862×10^1	4.418×10^1
Medical	Top-3 MLP Components (Avg.)	7.913×10^1	8.882×10^1	1.060×10^2
	Top-3 Attn Components (Avg.)	4.881×10^1	5.361×10^1	6.759×10^1
Python	Top-3 MLP Components (Avg.)	7.612×10^1	8.496×10^1	9.706×10^1
	Top-3 Attn Components (Avg.)	4.755×10^1	5.305×10^1	6.187×10^1
Science	Top-3 MLP Components (Avg.)	8.339×10^1	9.547×10^1	1.049×10^2
	Top-3 Attn Components (Avg.)	4.698×10^1	5.223×10^1	5.652×10^1

Table 6: Aggregated LoRA weight update norms for the Gemma-3 1B model across all domains.

Domain	Component Group	Avg. Full Run Norm	Avg. Ensemble Norm	Top Solo Run Norm
CPP	Top-3 MLP Components (Avg.)	4.315×10^1	5.039×10^1	6.484×10^1
	Top-3 Attn Components (Avg.)	2.451×10^1	2.822×10^1	3.337×10^1
Finance	Top-3 MLP Components (Avg.)	2.478×10^1	2.891×10^1	3.953×10^1
	Top-3 Attn Components (Avg.)	1.691×10^1	1.956×10^1	3.240×10^1
Math	Top-3 MLP Components (Avg.)	4.022×10^1	4.570×10^1	5.823×10^1
	Top-3 Attn Components (Avg.)	2.003×10^1	2.292×10^1	2.922×10^1
Medical	Top-3 MLP Components (Avg.)	4.811×10^1	5.544×10^1	7.106×10^1
	Top-3 Attn Components (Avg.)	2.955×10^1	3.401×10^1	4.053×10^1
Python	Top-3 MLP Components (Avg.)	4.297×10^1	4.926×10^1	6.502×10^1
	Top-3 Attn Components (Avg.)	2.501×10^1	2.846×10^1	3.237×10^1
Science	Top-3 MLP Components (Avg.)	4.973×10^1	5.627×10^1	6.923×10^1
	Top-3 Attn Components (Avg.)	2.516×10^1	2.830×10^1	3.364×10^1

A.2 PROBING ANALYSIS

The process of calculating separability scores between each pair of datasets, layer-wise, consists of 2 main components:

1) Hooking to get activations

2) Using these activations to get the Separability Scores

Hook placement and construction of per-sample representations. When analyzing representations inside transformer layers, forward hooks are placed on sub-modules corresponding to the **Attention block**, **MLP block**, and **Residual stream activations**. Each hook captures the output tensor of shape $[B, S, D]$, where B is the batch size (examples per forward pass), S is the sequence length (tokens per example), and D is the hidden dimension of the representation. To simplify, the token dimension is mean-pooled, giving a $[B, D]$ embedding for each batch. These embeddings are concatenated across multiple forward passes to construct a design matrix $X \in \mathbb{R}^{N \times D}$, where N is the total number of collected samples. Alongside, a label vector $y \in \{0, \ldots, C-1\}^N$ is created so that each row X_r corresponds to its class label y_r.

To compute **Fisher separability** between two classes i and j, we first isolate the subsets of X belonging to those labels, giving matrices $X_i \in \mathbb{R}^{n_i \times D}$ and $X_j \in \mathbb{R}^{n_j \times D}$. The mean representation of each class (μ_i, μ_j) is calculated across their samples, and the variance within each class $(\text{var}_i, \text{var}_j)$ is also estimated. Fisher's score is then defined as the squared distance between the two class means, normalized by the sum of their variances. Intuitively, if the means are far apart relative to how spread out the classes are internally, the score is high, indicating that the two classes are well separated in the representation space.

For the **Maximum Mean Discrepancy (MMD)**, the same class-specific subsets X_i and X_j are compared using a kernel function, typically a Gaussian RBF kernel. Pairwise distances between samples are used to determine the kernel bandwidth γ, and kernel similarity matrices are constructed: within-class (K_{ii}, K_{jj}) and cross-class (K_{ij}). The MMD score is then computed as the difference between average within-class similarities and average cross-class similarities. A larger MMD value means the two distributions X_i and X_j are more dissimilar, capturing not just differences in means but also higher-order mismatches in distributional shape.

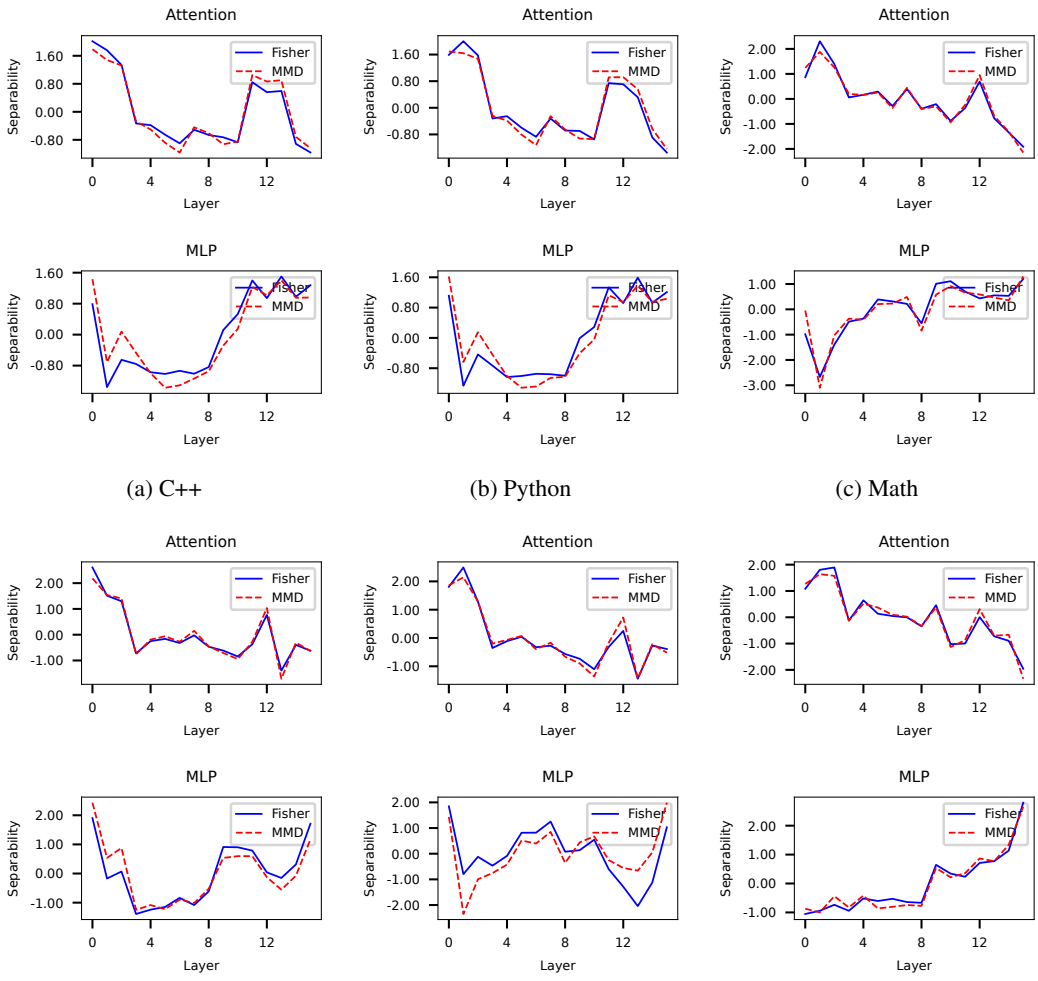
EXPERIMENT PARAMETERS

Samples per domain (forward pass)	MLP hook	Attention hook	Batch size
1000	up_proj	o_proj	8

Parameters used for all models: Llama 3.2 3B, Llama 3.2 1B, Gemma 3 4B, and Gemma 3 1B.

A.3 CAUSAL INTERVENTION

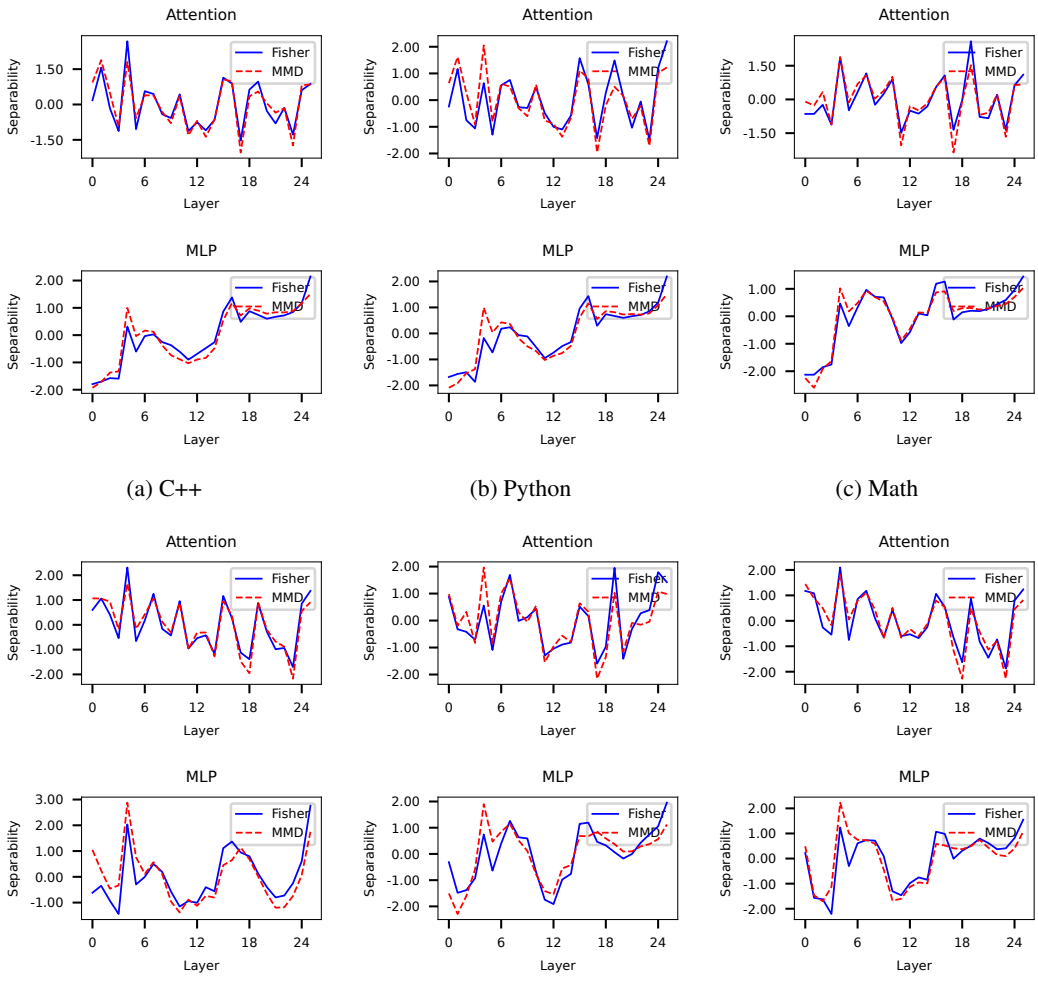
The results for the causal activation swapping case study of C++ $\leftrightarrow$ Python, for other models, are given in Figures 10-12.



(a) C++ (b) Python (c) Math

(d) Medical (e) Science (f) Finance

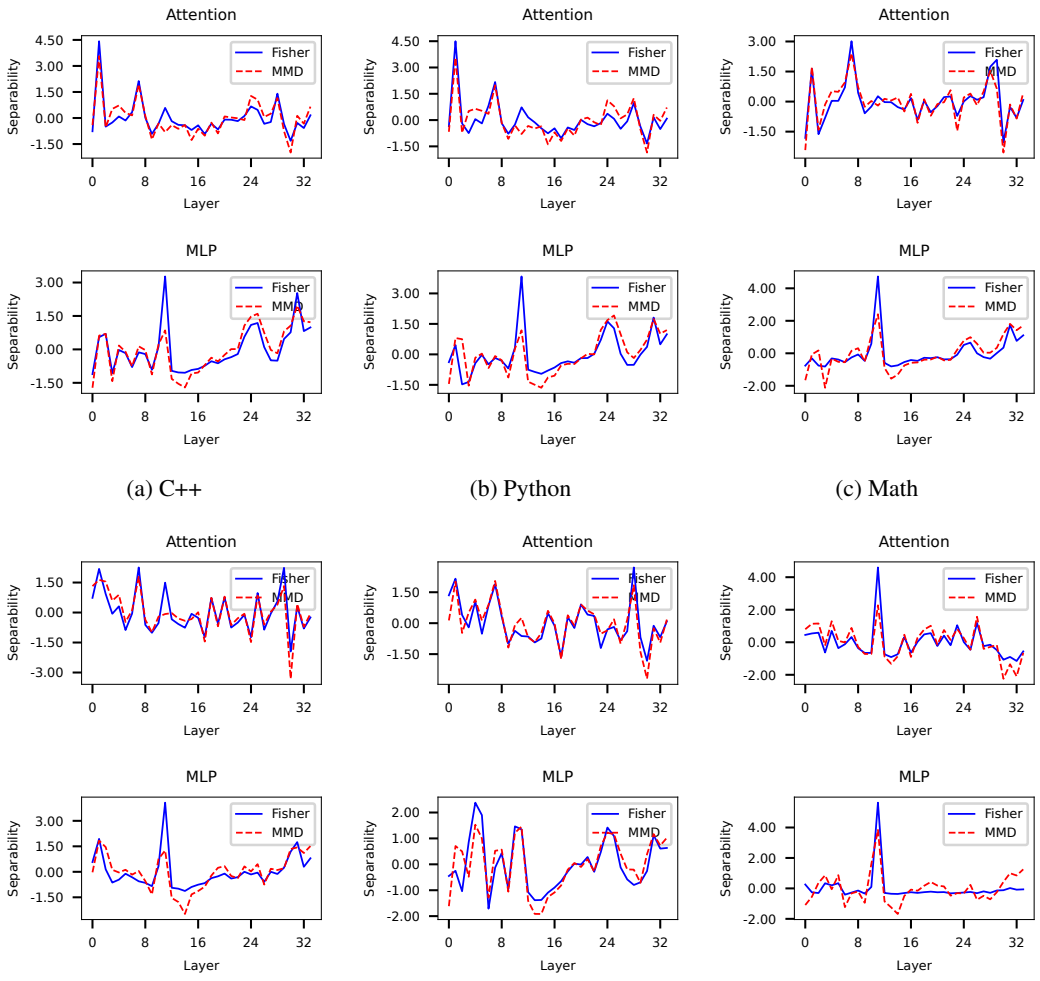
Figure 7: probe separability results for Llama 1B Model



(a) C++ (b) Python (c) Math

(d) Medical (e) Science (f) Finance

Figure 8: probe separability results for Gemma 1B Model



(a) C++ (b) Python (c) Math

(d) Medical (e) Science (f) Finance

Figure 9: probe separability results for Gemma 4B Model

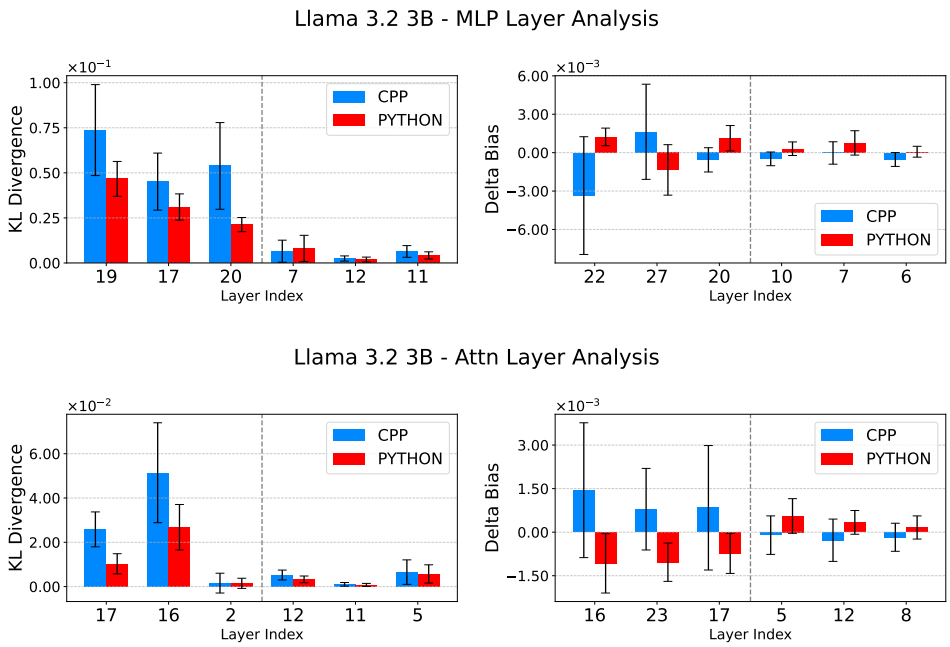


Figure 10: Analysis of layers in Llama-3B, comparing KL divergence (left) and a Delta Bias (right) between C++ and Python inputs. The layers on left section ar e layers with highest Fisher score and right section have lowest Fisher score. Top-ranked layers show substantially higher KL divergence and Delta Bias, reflecting higher influence on final output.

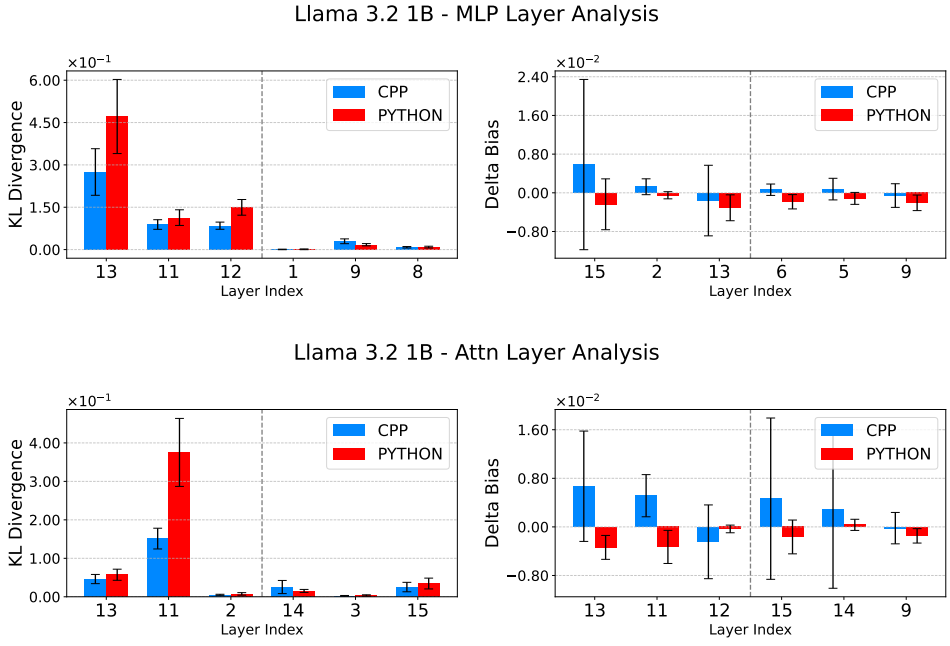


Figure 11: Analysis of layers in Llama-1B, comparing KL divergence (left) and Delta Bias between C++ and Python inputs. Top MLP layers have high variance in Delta Bias, and high fisher layers show more KL divergence.

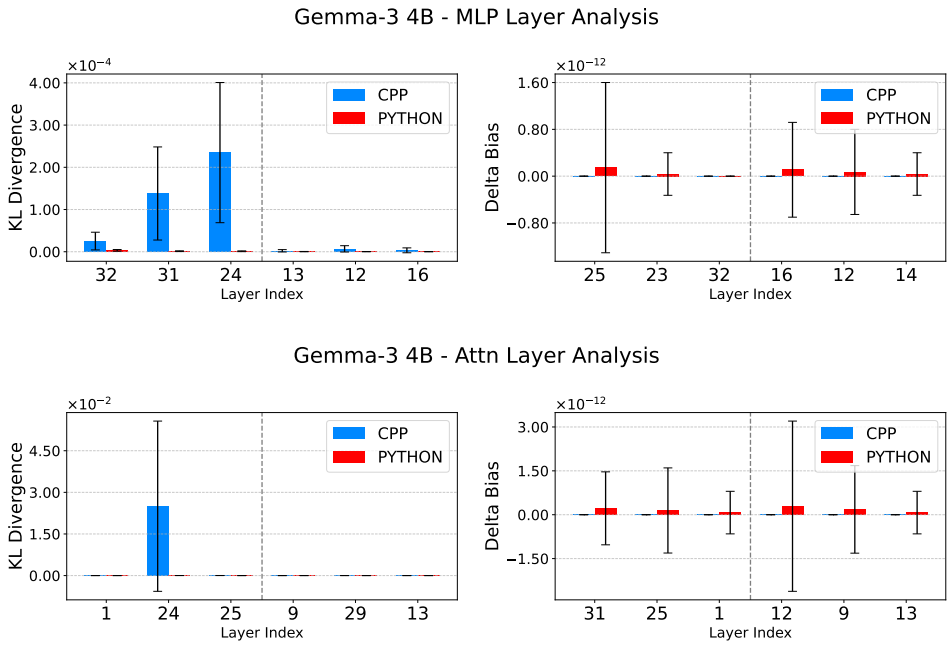


Figure 12: Analysis of layers in Gemma-4B, comparing KL divergence (left) and a Delta Bias (right) between C++ and Python inputs. A handful of layers show significant spikes in contribution and shifting operation in this model. Reason being high confidence of model on a single token in logits

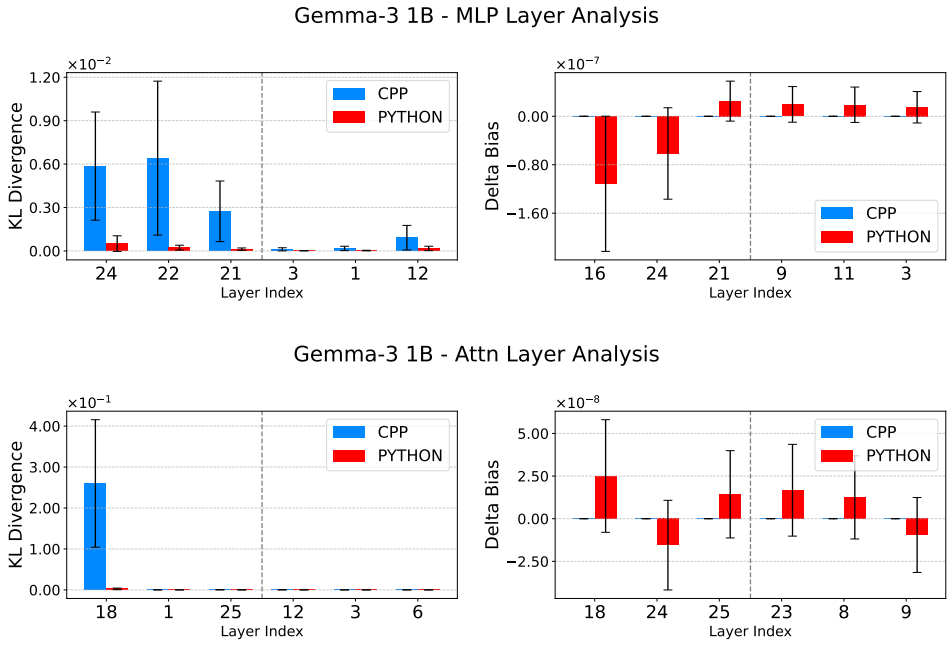


Figure 13: Analysis of layers in Gemma-1B, comparing KL divergence (left) and Delta Bias (right) between C++ and Python inputs. A handful of layers show significant spikes in contribution and shifting operation in this model.

B DATASETS

C++, Python For our coding datasets, we have used the Open Coder LLM Annealing Corpus (Huang et al. (2024)) which contains functional code snippets on various coding questions. This dataset aligns with our Human Benchmark Evaluation tests since it uses the same formatting. Each data point has a top level comment describing the task followed by a function that implements the task. The original dataset also contains inline comments inside the function body but these have been striped for conciseness. Listing 1 and Listing 14 showcase examples from our dataset on C++ and Python snippets.

Science We have used the SciQ dataset (Johannes Welbl, 2017) which contains crowd-sourced questions on Physics, Chemistry and Biology. The questions are in multiple-choice format with 4 answer options each. For our purposes we have formatted the data-points into Context, Question and Answer.

```
Context: Enzymes are critical to the body's healthy functioning.
    They assist, for example, with the breakdown of food and its
    conversion to energy. In fact, most of the chemical reactions
    in the body are facilitated by enzymes.
Question: Most of the chemical reactions in the body
    are facilitated by what?
Options: A. proteins B. enzymes C. vitamins D. carbohydrates
Answer: B
```

Mathematics The Math dataset is GSM8K (Cobbe et al., 2021a) which is a dataset of 8.5k high quality math word problems. The dataset contains question answering on basic mathematical problems that require multi-step reasoning. The datapoints are also similarly formatted into Question, Answer and Final Answer.

```
Question: Natalia sold clips to 48 of her friends in April, and
    then she sold half as many clips in May. How many clips did
    Natalia sell altogether in April and May?
Answer: Natalia sold 48/2 = <<48/2=24>>24 clips in May. Natalia
    sold 48+24 = <<48+24=72>>72 clips altogether in April and May. #### 72
Final Answer: 72.
```

Finance The Finance dataset (Mateega et al., 2025) is a set of financial question and answer pairs extracted from company annual reports, balance sheets, and financial statements.The datapoints contain context with some financial values and the model is questioned upon some value that is dependant on this information. A similar formatting technique is used where we explicitly state the context, question and answer.

```
Context: Liabilities: 8,537.39 Total Capital And Liabilities:
    13,410.53 ASSETS: nan NON-CURRENT ASSETS: nan Tangible
    Assets: 74.2 Intangible Assets: 4.16 Capital Work-In-Progress:
    0 Other Assets: 0 Fixed Assets: 98.73 Non-Current
    Investments: 0 Deferred Tax Assets [Net]: 0 Long Term Loans And Advances: 0
    Other Non-Current Assets: 15.61 Total Non-Current Assets: nan
Question: What is the total value of assets of the company?
Answer: The total value of assets of the company is $13,410.53.
Final Answer: 13410.53.
```

Medical We use the ReasonMed dataset (link lingshu-medical-mllm/ReasonMed) which is an open-source synthetic medical reasoning dataset containing multi-step chain-of-thought (CoT) rationales and concise summaries of LLMs such as Qwen-2.5-72B, DeepSeek-R1-Distill-Llama-70B, and HuatuoGPT-o1-70B on medical questions.

```
The question presents a radiographic scenario: a PA
(posteroanterior) ulnar deviation view of the wrist, asking
for the most likely diagnosis among the following options:
Osteomyelitis, De Quervain tenosynovitis, Hypertrophic
osteoarthropathy, and Rheumatoid arthritis. The correct answer
```

```python
def find_pivot_index(nums: list[int]) -> int:
    """Finds the pivot index of a list of numbers.

    The pivot index is where the sum of the numbers to the left of
         the index
    is equal to the sum of the numbers to the right of the index.

    Args:
        nums: A list of integers.

    Returns:
        The pivot index if one exists, otherwise -1.

    Examples:
        >>> find_pivot_index([1, 7, 3, 6, 5, 6])
        3
        >>> find_pivot_index([1, 2, 3])
        -1
        >>> find_pivot_index([2, 1, -1])
        0
    """
    total_sum = sum(nums)
    left_sum = 0

    for i, num in enumerate(nums):
        if left_sum == total_sum - left_sum - num:
            return i
        left_sum += num

    return -1
```

Listing 1: A Python snippet from our dataset illustrating a simple coding problem with a docstring that explicitly describes the working of the function

```cpp
// This function takes a positive integer as input and returns a
    list of its prime factors,
// which are the prime numbers that multiply together to equal the
     original number.
// The prime factors are returned in ascending order.
//
// Parameters:
// * number: A positive integer to be factorized into its prime
    factors.
//
// Examples:
// * find_prime_factors(8) => [2, 2, 2]
// * find_prime_factors(25) => [5, 5]
// * find_prime_factors(70) => [2, 5, 7]
std::vector<int> find_prime_factors(int number) {
    std::vector<int> prime_factors;

    while (number % 2 == 0) {
        prime_factors.push_back(2);
        number /= 2;
    }

    for (int i = 3; i <= std::sqrt(number) + 1; i += 2) {
        while (number % i == 0) {
            prime_factors.push_back(i);
            number /= i;
        }
    }

    if (number > 2) {
        prime_factors.push_back(number);
    }

    return prime_factors;
}
```

Figure 14: A C++ snippet from our dataset featuring a prime factorization problem. Each example contains a descriptive comment above the function body and clear naming conventions for the function itself.

```
is De Quervain tenosynovitis. To comprehensively understand
and justify this answer, it's essential to dissect
each component... (truncated)
```

C EVALUATION

C.1 EVALUATION RESULTS

Our evaluation of domain-specific performance uses two accuracy metrics tailored to the task type. For the Math, Science, Finance, and Medical domains, we report standard classification accuracy, defined as:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Samples}}$$

For the programming domains (C++ and Python), we evaluate code generation correctness using the **pass@k** metric. Specifically, we use **pass@10**, where the model generates 10 candidate solutions for each problem. A problem is considered solved if at least one of these candidates passes all unit tests. The accuracy is therefore calculated as:

$$\text{pass@10} = \frac{\text{Number of Problems with at least one passing solution}}{\text{Total Number of Problems}}$$

It is important to note that the results presented, particularly for the smaller 1B models, may exhibit some noise. These models operate with fewer parameters, making performance sensitive to minor variations in fine-tuning, which can affect the robustness of the generated outputs.

	PT	MLP	Attn	Both	Top-1 MLP	Top-1 Attn	Top-3 MLP	Top-3 Attn
Math	0.040	**0.070**	0.050	0.020	0.050	0.040	0.030	0.040
Science	0.395	0.390	**0.535**	0.475	0.385	0.290	0.310	0.325
CPP	0.120	0.020	0.020	0.000	0.050	0.040	**0.130**	0.040
Python	0.440	0.020	0.180	0.040	0.040	**0.350**	0.160	0.290
Finance	0.180	0.020	0.010	0.000	0.060	0.040	0.020	**0.070**
Medical	0.847	0.687	0.787	0.813	0.904	0.424	**0.916**	0.864

Llama-3.2-1B

	PT	MLP	Attn	Both	Top-1 MLP	Top-1 Attn	Top-3 MLP	Top-3 Attn
Math	0.100	0.030	0.060	0.030	**0.140**	0.060	0.040	0.040
Science	0.625	**0.755**	0.700	0.650	0.610	0.600	0.425	0.425
CPP	0.320	0.000	0.030	0.000	0.000	**0.286**	0.000	0.150
Python	0.470	0.040	0.300	0.050	0.286	**0.371**	0.220	0.340
Finance	0.080	0.020	**0.050**	0.040	0.025	0.000	0.030	0.030
Medical	0.900	0.713	**0.912**	0.512	0.880	0.880	**0.912**	0.888

Gemma-3-1B

	PT	MLP	Attn	Both	Top-1 MLP	Top-1 Attn	Top-3 MLP	Top-3 Attn
Math	0.080	0.080	0.060	0.080	0.200	0.133	**0.400**	0.267
Science	0.715	0.780	0.780	0.760	**0.840**	0.820	0.760	0.700
CPP	0.833	0.033	0.000	0.000	0.028	**0.457**	0.000	0.286
Python	0.300	0.233	0.333	0.033	**0.371**	0.343	0.286	0.343
Finance	0.040	0.000	0.000	0.000	0.000	0.000	**0.025**	**0.025**
Medical	0.925	**0.950**	**0.950**	0.300	0.875	**0.950**	0.725	0.850

Gemma-3-4B

As an alternative performance metric, we measured the asymptotic validation loss for different component combinations. The results aligned with our separability analysis: layers identified as having high activation separability consistently outperformed those with lower separability, converging to a significantly lower validation loss.

C.2 Domain Evaluation

C.2.1 Math

Dataset chosen: *GSM8K* (Grade School Math 8K) introduced by Cobbe et al. (2021b) is a collection of grade-school level math word problems designed to evaluate multi-step arithmetic and reasoning ability. The dataset emphasizes chain-of-thought style reasoning where intermediate steps are useful to arrive at the correct numeric result.

GSM8K is used here as it's a widely used benchmark for studying reasoning behavior in language models and for evaluating self-consistency / majority-vote sampling methods. Also, it is not too difficult, hence used for evaluation on the small models considered.

Prompt–Output Illustration:

```
# <prefix text (8-shot demos)
    provided for context>

Q: John has 3 apples.
He buys 2 more.
How many apples does he have
   now?

A: Let's reason step by step.
At the end, give the final
   numeric
answer on its own line in
   this exact format:
#### <number>
Answer:
```

```
# Example reasoning and
   output

Step 1: John starts with 3
   apples.
Step 2: He buys 2 more.
Step 3: Total apples = 3 + 2
   = 5.

#### 5
Answer:
```

Illustration of the prompt (left) and an example of the expected LLM output (right).

Evaluation Samples	Sampling Amount Per Sample	Max Generation Tokens	Temperature	Top_p
100	10	1024	0.7	0.90

(Hyper-Parameters used during Model inference For Evaluation(Self-Consistency)

C.2.2 Finance

Dataset chosen: *FinanceQA* introduced by Mateega et al. (2025) is a curated set of financial question–answer pairs extracted from company filings (annual reports, balance sheets, and reports). It supplies queries, short factual answers, and the supporting context passage from the source document (e.g., a few sentences or table rows). Focus is on **numerical output comparison and extraction**.

FinanceQA is used for evaluation as it provides a domain-specific **"finance + math"** evaluation setting, requiring both factual retrieval and quantitative reasoning.

Prompt–Output Illustration:

```
# FinanceQA prompt builder

(context + query)

Context:
<supporting passage from
    financial filings>

Question:
<query here>

Answer: The final answer is

Final Answer:
```

```
# Example reasoning and
    output

Step 1: From the context, the
     net profit
margin in 2021 is explicitly
    given.
Step 2: The reported margin
    is 11.04%.

Final Answer: 11.04%
```

Illustration of the FinanceQA prompt template (left) and an example expected LLM output (right).

Evaluation Samples	Sampling Amount Per Sample	Max Generation Tokens	Temperature	Top_p
100	10	512	0.7	0.95

(Hyper-Parameters used during Model inference For Evaluation(Self-Consistency))

C.2.3 MEDICAL

Dataset chosen: *PubMedQA* introduced by Jin et al. (2019) is a dataset of biomedical research questions paired with contexts and a short (yes/no) final decision derived from biomedical articles. Each sample often contains an abstract or supporting passage and a question about the clinical finding; the ground truth is typically a binary decision. Sometimes if LLM is highly undecisive the output of LLM is assumed 'None'

We use PubMedQA because it is a widely-used , biomedical QA benchmark for evaluating concise, high-precision yes/no answers in the clinical/research domain.

Prompt–Output Illustration:

```
# PubMedQA prompt builder (
    question + context)

Context:
<concatenated context sentences
    or abstract>

Question: <question here>

Based on the context above,
    answer the question
with exactly 'yes' or 'no' (
    lowercase),
and do NOT provide any
    explanation.
Answer:
```

Illustration prompt template used Sample output is simply Yes/No , In case Bad output Then None is interpreted

Evaluation Samples	Sampling Amount Per Sample	Max Generation Tokens	Temperature	Top_p
250	1	512	0.0	1.00

(Hyper-Parameters used during Model inference For Evaluation (Greedy))

C.2.4 SCIENCE

Dataset chosen: *SciQ* introduced by Welbl et al. (2017) is a data set of multiple choice science questions that contains short grade-level science questions with four answer options (A–D) and optional supporting facts. Each example includes a question, four candidate answers, and (sometimes) a support passage.

SciQ is used because it provides well-formed multiple-choice prompts suitable for evaluation,it is easy for a small LLM hence it is used.

Prompt–Output Illustration:

```
// SciQ prompt builder (
    question + options)

Question:
<question text>

Options:
A. <option A>
B. <option B>
C. <option C>
D. <option D>

Answer with the letter of the
    correct option only (A, B,
    C, or D).
Do NOT provide any explanation.
Answer:
```

```
Answer:B
```

Illustration: left = prompt template used for SciQ , model output is a single letter A/B/C/D.

Evaluation Samples	Sampling Amount Per Sample	Max Generation Tokens	Temperature	Top_p
200	1	256	0.0	1.00

(Hyper-Parameters used during Model inference For Evaluation(Greedy))

C.2.5 PYTHON

Dataset chosen: *HumanEvalPack (multilingual / Python subset)* Introduced by Chen et al. (2021) is a collection of programming problems with formal problem descriptions, expected function signatures, and test harnesses.

Inputs in the form of coding questions are provided, and the model is expected to output corresponding code which is executed against test cases. The accuracy used for evaluation is **pass@k**, a standard metric for code-generation tasks, rather than simple string-matching accuracy.

HumanEvalPack is used here because it provides language-specific (C++/Python/etc.) prompts with a standard "declaration + examples + tests" scheme. The problems are relatively simple, making this dataset ideal for comparing small models on code generation and correctness.

Prompt–Output Illustration:

```
# Problem:
<prompt_or_instruction>

# Signature:
<signature>

# Docstring:
<docstring>

# Examples:
<example_test>

Write the complete Python
    function
implementation only.
Output only valid Python code
    for the
function (no explanation, no
    tests,
no surrounding markdown).
Make sure the function name and
signature match the signature
    above.

Implementation:
```

```
# Example implementation for:
# def add(a: int, b: int) ->
    int

def add(a: int, b: int) -> int:
    # simple implementation
    return a + b
```

Illustration of the Python prompt template (left) and an example expected LLM output (right).

Evaluation Samples	Sampling Amount Per Sample	Max Generation Tokens	Temperature	Top_p
100	10	1024	0.7	0.95

(Hyper-Parameters used during Model inference For Evaluation(Self-Consistency))

C.2.6 CPP

Dataset chosen: *HumanEvalPack (multilingual / C++ subset)* Introduced by Chen et al. (2021) is a collection of programming problems with formal problem descriptions, expected function declarations/signatures, and test harnesses .Inputs in the form of coding questions are provided, and the model is expected to output corresponding code which is compiled against test cases.

HumanEvalPack is used here because it provides language-specific (C++/Python/etc.) prompts with a standard "declaration + examples + tests" scheme. The problems are relatively simple, making this dataset ideal for comparing small models on code generation and correctness.

Prompt–Output Illustration:

```
// Problem:
<prompt_or_instruction>

// Declaration:
<declaration>

// Docstring / Notes:
<docstring>

// Examples:
<example_test>

Write the C++ implementation
    only
(no explanation, no tests, no
    surrounding markdown).
Include necessary #include
    lines if needed.
Ensure function name and
    signature match the
    declaration above.

Implementation:
```

```
#include <bits/stdc++.h>
using namespace std;

// Example implementation for:
    int add(int a, int b)
int add(int a, int b) {
    // simple implementation
    return a + b;
}
```

Illustration of the C++ prompt template (left) and an example expected LLM output (right).

Evaluation Samples	Sampling Amount Per Sample	Max Generation Tokens	Temperature	Top_p
100	10	1024	0.7	0.95

(Hyper-Parameters used during Model inference For Evaluation(Self-Consistency))

D Experimental Setup

D.1 Fine Tuning

All experiments were run on NVIDIA H100 GPUs, using PyTorch and the Hugging Face ‘transformers‘ and ‘peft‘ libraries. To maximize computational throughput, the model was JIT-compiled using ‘torch.compile()‘. A fixed set of hyperparameters, detailed in Table 7, was used across all experiments to ensure fair comparison.

Table 7: Common hyperparameters for all fine-tuning experiments.

Parameter	Value
Training Configuration	
Optimizer	AdamW
Learning Rate	1×10^{-3}
Batch Size	8
Epochs (Stage 1 Mapping)	10
Epochs (Stage 2 Validation)	3
Seed	42
Precision	‘bfloat16‘
LoRA Configuration	
Rank (r)	16
Alpha (α)	32 ($2 \times r$)
Dropout	0.05
Target Modules (Attn)	`q_proj, k_proj, v_proj, o_proj`
Target Modules (MLP)	`gate_proj, up_proj, down_proj`

D.2 Causal Intervention

For causal intervention we use specific prompts on Python and C++ that follow the same format. It is ensured that the prompts differ with most 1 tokens at the exact same spot. This reduces noise and makes the model's output predictable.

```
C++ Prompt:
Write a short function in C++ that returns
the n-th Fibonacci number.\nRespond
with code only.\n```

Python Prompt:
Write a short function in Python that returns
the n-th Fibonacci number.\nRespond
with code only.\n```
```

The ``` at the end prompts the model to output ”cpp” or ”python” to conform to markdown conventions and thus also forcing the model to focus on domain specific information. Both prompts also ask the model to perform the same task but in different languages. This eliminates all unknown variables regarding linguistics and content of the task itself, so the differentiating point is the language used only.

For our experiments we use 100 such sample prompt pairs on all 4 models. The top 5 layers and bottom 5 layers are selected according to ranking by Fisher score metric for visualizing the contrasting behavior. Token sets are generated by reverse intervention process (See E.2) and used to compute Delta Bias values.

D.3 Probing Analysis

In addition to Fisher Separability and Maximum Mean Discrepancy (MMD), we also evaluated probing separability using other metrics such as classification probing accuracy, cosine similarity, and V-bits. However, for high-level abstraction tasks such as *Domain Separability*, the results across layers were not clearly distinguishable. This arises because, in such tasks, the points in the activation hyperspace are widely dispersed. Consequently, strong metrics such as V-bits or probing classification accuracy can easily separate these spread-out representations, making them less informative for fine-grained layer-wise analysis. In contrast, weaker metrics such as Fisher separability and MMD are more useful in these cases, as they provide more sensitive distinctions when the data is already well separated.

On the other hand, for low-level abstraction tasks such as *Concept-level Separability*, the points in the activation hyperspace are closely packed. In these scenarios, strong metrics such as V-bits prove more effective, yielding

clearly distinguishable results across layers. This observation is consistent with findings reported in Ju et al. (2024b).

E Extended Discussion

E.1 Hydra Effect

The Hydra Effect describes a form of self-repair capability present in LLMs. As described by McGrath et al. (2023), it refers to the mismatch between a layer's apparent contribution (measured by projecting its activations through the unembedding mechanism, $\Delta_{unembed}$) and its functional importance (measured by ablating the layer, Δ_{ablate}). We expect the ablation to reduce the model's confidence proportionally to its apparent contribution, but downstream layers reconstruct the corrupted signal so that

$$\Delta_{\text{ablate,l}} < \Delta_{\text{unembed,l}}$$

During interventions, the KL divergence is lower for early layers with high fisher score due to this reason since the intervention done is reverted to some extent by downstream layers.

E.2 Characteristic Tokens

The process of selecting characteristic tokens is derived from the same causal intervention process done in reverse. Instead of finding layers that do the most change to specific tokens, we find tokens that are most sensitive to interventions on all layers. This process is coined as the reverse causal intervention on a model.

When we do an intervention on a single layer from one domain to another, the tokens of the new domain are shifted up in probability. The overall shift across the vocabulary is averaged across all layers and the Top-k "promoted" tokens are saved in a list for the intervening dataset. For example, we have found when intervening C++ prompts with Python activations, tokens such as def, $import$ and $python$ are promoted. These form the characteristic token set for Python and this set is used in our causal intervention experiments further on.

E.3 Delta Bias

Let V be the entire vocabulary of the model. We denote the probability associated with a subset of vocabulary $S \subset V$ as $P(S|x) = \sum_{i \in S} p(i|x)$ with a prompt x. Suppose we perform the intervention $x_A \overset{l}{\leftarrow} x_B$ where activations of prompt of domain B are inserted into the forward pass of A at layer l. Before intervention, $P_{base}(S_A|x_a)$ and $P_{base}(S_B|x_a)$ denote the probabilities of characteristic tokens of A and B before intervention, and $P_{swap}(S_A|x_A \overset{l}{\leftarrow} x_B)$ and $P_{swap}(S_B|x_A \overset{l}{\leftarrow} x_B)$ as the probabilities of the set of characteristic tokens of A and B after intervention. The $Bias$ present in the probability distribution is defined as Bias $= P(S_B) - P(S_A)$. This represents the model's preference on predicting the intervening subset of tokens.

$$\begin{aligned}
\text{Bias}_{base}(x_A) &= P_{base}(S_B|x_A) - P_{base}(S_A|x_A) \\
\text{Bias}_{swap}(x_A \overset{l}{\leftarrow} x_B) &= P_{swap}(S_B|x_A \overset{l}{\leftarrow} x_B) - P_{swap}(S_A|x_A \overset{l}{\leftarrow} x_B) \\
\Delta\text{Bias}(A \overset{l}{\leftarrow} B) &= \mathbb{E}_{x_A \sim A, x_B \sim B}\left[\text{Bias}_{swap}(x_A \overset{l}{\leftarrow} x_B) - \text{Bias}_{base}(x_A)\right]
\end{aligned}$$

In our results, we use the convention for when $A \overset{l}{\leftarrow} B$ is done, we plot bias with a positive sign, and when we do intervention $B \overset{l}{\leftarrow} A$, we plot bias with a negative sign to preserve perspective with respect to the set of characteristic tokens B. So, all bias computations are visualized as the shift in preference of B over A.