THE INFLUENCE SPECTRUM: THE POWER OF LAYER-WISE IMPORTANCE IN LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

The black-box nature of Large Language Models (LLMs) poses significant challenges to their transparency and controllability. Understanding which layers are most relevant to a given task provides a practical basis for interpreting LLM behavior and enabling effective task adaptation. We propose the influence spectrum framework to characterize task-specific layer importance in LLMs. At its core, the Parameter-space Influence Function (PIF) measures local loss sensitivity to parameter perturbations through first-order gradient information and second-order directional curvature. Using Hessian-vector products (HVPs), PIF condenses massive second-order operations into just two sequential gradient evaluations. Layer-masking experiments validate the diagnostic value of the resulting influence spectra, while extensive evaluations across diverse open-source LLMs and benchmarks demonstrate the utility of our framework in Parameter-Efficient Fine-Tuning (PEFT), continual learning, dynamic model defense, and knowledge editing.

1 INTRODUCTION

Large Language Models (LLMs) have achieved strong performance across a wide range of natural language processing and domain-specific tasks. However, their internal mechanisms remain difficult to interpret and control Bommasani et al. (2021). A central challenge is to identify which model components contribute most to task-specific behavior. Characterizing these contributions provides a practical basis for interpreting model behavior and enabling targeted adaptation Elhage et al. (2021); Zou et al. (2023); Templeton et al. (2026).

A natural way to study such task-specific contributions is through layer importance. Existing approaches estimate layer importance using parameter magnitude, gradients, Fisher information, or layer removal Li et al. (2017); Kirkpatrick et al. (2017); Zhang et al. (2024); Shi et al. (2025). These methods provide useful empirical signals but capture different notions of importance. Magnitude is task-agnostic, gradients reflect only first-order sensitivity, diagonal Fisher approximations ignore parameter interactions, and removal-based methods require repeated intervention and evaluation. More broadly, they lack a unified formulation linking layer importance to the local response of the task objective, motivating a principled sensitivity-based definition.

Influence functions (IF) characterize the local sensitivity of a statistical functional to infinitesimal contamination of the underlying distribution Hampel (1974); Ronchetti & Huber (2009). In machine learning, IF measures how upweighting a training example affects model parameters and predictions Koh & Liang (2017). Recent work extends methods for estimating training-data influence to LLMs Grosse et al. (2023); Kwon et al. (2024); Askari et al. (2025); Bao et al. (2025); Tu et al. (2025); Li et al. (2026b). For example, LayerIF uses training-example influence to estimate layer relevance Askari et al. (2025), but still operates through data-space perturbations and requires inverse-HVP estimation. We extend the same local-sensitivity principle directly to parameter space by perturbing model parameters and studying the resulting task-loss response. Based on this view, we introduce the Influence Spectrum and its core measure, the Parameter-space Influence Function (PIF), which combines first-order sensitivity with second-order directional curvature to characterize layer importance conditioned on the task, parameter block, and perturbation direction.

Our framework computes directional curvature through Hessian–vector products (HVPs) without explicitly constructing or inverting the Hessian, enabling practical second-order sensitivity analysis

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

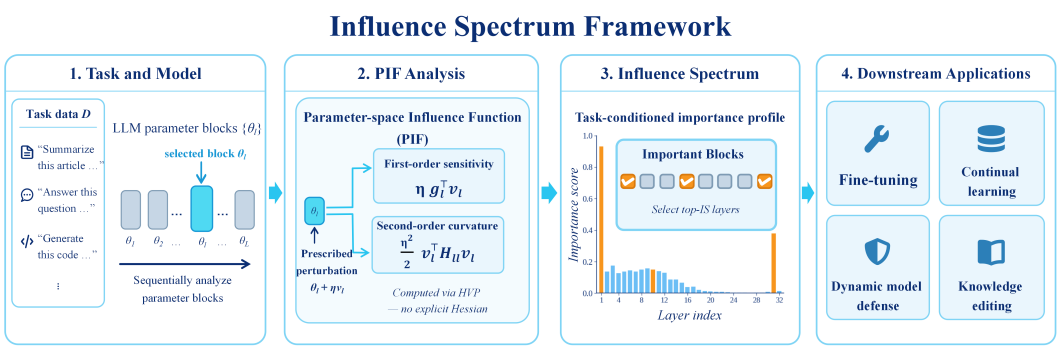

Figure 1: Overview of the Influence Spectrum framework.

in LLMs. The task-loss responses are organized into layer-wise influence spectra (Figure 1). Layer-masking experiments support the diagnostic value of the resulting importance profiles. We further evaluate our framework across several downstream tasks, including Parameter-Efficient Fine-Tuning (PEFT), continual learning, dynamic model defense, and knowledge editing. Experimental results demonstrate the utility of the proposed framework across these settings.

2 Preliminaries

Problem setup. Throughout this paper, we use ∇ and ∇^2 to denote the gradient and Hessian operators, respectively. Δ represents the change, $(\cdot)^{-1}$ denotes the matrix inverse, and $(\cdot)^T$ denotes the transpose. Let $\theta \in \mathbb{R}^D$ denote the parameters of a pretrained language model, and let $\theta_l \in \mathbb{R}^{d_l}$ denote a selected parameter block. Given a task dataset $\mathcal{Z} = \{z_i\}_{i=1}^n$ and sample loss $L(z;\theta)$, we define the empirical task loss as

$$\mathcal{L}_{\mathcal{Z}}(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(z_i;\theta). \tag{1}$$

We study the local response of the task loss to a prescribed perturbation of θ_l along a direction $\boldsymbol{v} \in \mathbb{R}^{d_l}$ with magnitude $\eta \in \mathbb{R}$. This local response provides the basis for the task-conditioned parameter-importance profile introduced below.

Connection to influence functions. Classical influence functions characterize how perturbations in training data propagate to model parameters and predictions Koh & Liang (2017); Ronchetti & Huber (2009). Let $\mathcal{X} = \{x_i\}_{i=1}^n$ be a training dataset and let $\hat{\theta}$ be a local minimizer of

$$\mathcal{R}(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(x_i;\theta). \tag{2}$$

Upweighting a training sample x by ϵ defines

$$\mathcal{R}_{\epsilon,x}(\theta) = \mathcal{R}(\theta) + \epsilon L(x;\theta). \tag{3}$$

Let $\hat{\theta}_{\epsilon,x}$ denote the corresponding local minimizer, with $\hat{\theta}_{0,x} = \hat{\theta}$. Under standard differentiability assumptions and a nonsingular Hessian

$$H_{\hat{\theta}} = \nabla^2_\theta \mathcal{R}(\hat{\theta}), \tag{4}$$

the local parameter response is

$$\left.\frac{d\hat{\theta}_{\epsilon,x}}{d\epsilon}\right|_{\epsilon=0} = -H_{\hat{\theta}}^{-1}\nabla_\theta L(x;\hat{\theta}). \tag{5}$$

Classical IF analyzes an indirect response chain from data perturbations to changes in an optimized model. In contrast, PIF directly probes the local loss geometry in parameter space by perturbing a selected parameter block at the current checkpoint. Consequently, PIF does not require the checkpoint to minimize the downstream task loss and avoids inversion of the training-loss Hessian.

3 PROPOSED METHOD AND CALCULATION ANALYSIS

3.1 PARAMETER-SPACE INFLUENCE FUNCTION

For a selected parameter block θ_l, let $\boldsymbol{v} \in \mathbb{R}^{d_l}$ denote a prescribed perturbation direction and $\eta \in \mathbb{R}$ a perturbation magnitude. The perturbed block is

$$\theta_l(\eta) = \theta_l + \eta \boldsymbol{v}. \tag{6}$$

We partition the full model parameters as $\theta = (\theta_l, \theta_{-l})$, where θ_{-l} denotes all parameters outside the selected block. When perturbing θ_l, we keep θ_{-l} fixed.

The direction $\boldsymbol{v}$ is treated as fixed when differentiating with respect to η, including when it is constructed from the current parameters or gradients. For a task sample z, define the loss response

$$\Delta L = L(z; \theta_l + \eta \boldsymbol{v}, \theta_{-l}) - L(z; \theta_l, \theta_{-l}), \tag{7}$$

where the dependence on parameters outside θ_l is made explicit for clarity. Assuming sufficient smoothness along the perturbation path near $\eta = 0$, a second-order Taylor expansion gives

$$\Delta L = \eta \left[\frac{d}{d\eta} L(z; \theta_l + \eta \boldsymbol{v}, \theta_{-l}) \right]_{\eta=0} + \frac{\eta^2}{2} \left[\frac{d^2}{d\eta^2} L(z; \theta_l + \eta \boldsymbol{v}, \theta_{-l}) \right]_{\eta=0} + o(\eta^2). \tag{8}$$

Applying the chain rule, we derive the first-order PIF term $PIF_l^{(1)}$ in parameter space. Equation equation 9 measures the first-order directional sensitivity of the task loss to the prescribed perturbation. The derivation is provided in Appendix A.1.

$$PIF_l^{(1)}(z) = \eta \nabla_{\theta_l} L(z; \theta)^T \boldsymbol{v}. \tag{9}$$

The second-order PIF term $PIF_l^{(2)}$ captures directional curvature within the selected parameter block, including second-order interactions among parameters inside the block. It can be expressed as the quadratic form induced by the block Hessian and the perturbation direction, as shown in Equation equation 10. The derivation is provided in Appendix A.2.

$$PIF_l^{(2)}(z) = \frac{\eta^2}{2} \boldsymbol{v}^T H_{ll}(z; \theta) \boldsymbol{v}, \qquad H_{ll}(z; \theta) = \nabla^2_{\theta_l \theta_l} L(z; \theta). \tag{10}$$

Here, $H_{ll}(z; \theta)$ is the Hessian block with respect to the selected parameter block, evaluated at the unperturbed checkpoint. The quadratic form $\boldsymbol{v}^T H_{ll}(z; \theta) \boldsymbol{v}$ measures the second-order curvature along the prescribed perturbation direction.

3.2 PERTURBATION DIRECTION

In our implementation, the perturbation direction is constructed from the loss gradient at the unperturbed checkpoint. Specifically, for each parameter tensor p within the selected block, we define

$$\boldsymbol{v}_p = -\frac{\nabla_p L(z; \theta)}{\|\nabla_p L(z; \theta)\|_2}. \tag{11}$$

Each parameter tensor is normalized independently to reduce scale differences across tensors, and the block-level direction $\boldsymbol{v}$ is formed by collecting these tensor-wise normalized directions. Once constructed, $\boldsymbol{v}_p$ is detached from the computational graph and treated as fixed when computing directional derivatives and Hessian–vector products.

3.3 HVP COMPUTATION AND COMPUTATIONAL COST

Directly forming the block Hessian $H_{ll}(z; \theta) \in \mathbb{R}^{d_l \times d_l}$ requires $O(d_l^2)$ storage. Since PIF only requires the action of the block Hessian on the prescribed direction $\boldsymbol{v}$, we compute the corresponding HVP without explicitly constructing the Hessian. Treating $\boldsymbol{v}$ as fixed,

$$H_{ll}(z;\theta)\boldsymbol{v} = \nabla_{\theta_l}\left(\nabla_{\theta_l} L(z;\theta)^T \boldsymbol{v}\right). \tag{12}$$

Substituting Equation equation 12 into Equation equation 10 gives

$$PIF_l^{(2)}(z) = \frac{\eta^2}{2}\boldsymbol{v}^T \nabla_{\theta_l}\left(\nabla_{\theta_l} L(z;\theta)^T \boldsymbol{v}\right). \tag{13}$$

The proof is provided in Appendix A.3, and the computational procedure in Appendix A.4.

Computation. For each sample, we compute the loss gradient, construct the corresponding perturbation direction, and detach it from the computational graph. We then use automatic differentiation to obtain the HVP $H_{ll}(z;\theta)\boldsymbol{v}$ along this fixed direction. The first- and second-order PIF terms are obtained from the corresponding directional inner products.

Computational cost. For each block-direction pair, PIF requires only one first-order gradient evaluation and one additional reverse-mode differentiation to obtain the HVP, avoiding explicit Hessian construction, storage, and inversion. Its directional storage cost is $O(d_l)$ rather than $O(d_l^2)$ for a dense block Hessian, while the total runtime scales with the number of samples, blocks, and perturbation directions.

Accuracy and scope. HVPs compute the directional Hessian action up to numerical precision without diagonal or low-rank approximations. Therefore, PIF enables matrix-free second-order directional sensitivity analysis, while only remaining subject to Taylor truncation and finite-sample variability.

4 Experiments

We first conduct layer-masking experiments to validate the diagnostic value and computational efficiency of our method (Section 4.1). We then evaluate several downstream applications to demonstrate how our proposed method facilitates the adaptation of LLMs: Parameter-Efficient Fine-Tuning (PEFT) (Section 4.2), continual learning (Section 4.3), dynamic model defense (Section 4.4), and knowledge editing (Section 4.5).

4.1 Layer-Masking

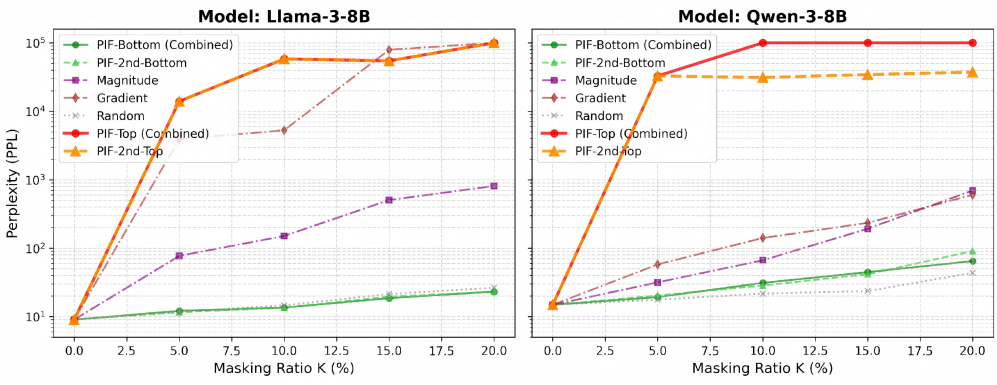

Figure 2: Degradation of language modeling performance under different layer-masking strategies across varying masking ratios ($K\%$). Detailed definitions of each masking strategy are provided in Appendix A.5.

Experimental Setup and Baselines. We use three representative models: the classic Post-LN GPT-2-XL (1.5B) Radford et al. (2019), alongside modern Pre-LN/RMSNorm architectures Llama-3

(8B) Grattafiori et al. (2024) and Qwen-3 (8B) Yang et al. (2025). We evaluate PIF from two perspectives: (1) **Validity.** For hierarchical importance assessment, we compare PIF against random selection (Random), weight magnitude (L_2-norm Li et al. (2017; 2026a); Men et al. (2025)), and empirical gradients. On Llama-3-8B, we further compare PIF with Fisher, EWC-removal, leave-one-layer-out (LOO), Neighborhood Shapley, and ILA. (2) **Efficiency.** We benchmark peak VRAM usage and average execution time per layer on a single NVIDIA A100 (80GB) GPU. Comparison algorithms include exact Hessian inversion, stochastic approximation (LiSSA Li et al. (2025)), and block-diagonal approximation (K-FAC van der Ouderaa et al. (2024); Seung et al. (2024)).

Table 1: Layer-masking comparison on Llama-3-8B. The PPL of the unmodified model is 6.675. Larger PPL indicates greater degradation after layer removal.

Method	**PPL**		
	2 blocks	**3 blocks**	**6 blocks**
Gradient	14.6	26.1	39.1
Fisher	59,997.1	78,367.1	20,161.4
EWC-removal	380.8	910.0	37,976.8
Magnitude	53.5	114.8	649.6
LOO	59,997.1	575,061.5	1,856,120.3
Neighborhood Shapley	62,188.8	575,061.5	427,901.6
ILA	11.5	13.6	3,230.9
PIF	**62,188.8**	**91,953.2**	**154,752.6**

To validate our proposed PIF, we conduct layer-masking experiments on Llama-3 and Qwen-3 (8B) and evaluate perplexity (PPL Jurafsky & Martin (2024)) on WikiText-2 Merity et al. (2017) (Figure 2). Masking only 5% of PIF-Top layers causes PPL to increase sharply from $\sim$10 to over 30,000, leading to sustained output degradation in the 10%–20% range. This supports the ability of PIF to localize performance-sensitive layers. Conversely, standard baselines (Gradient and Magnitude) fail to induce comparable degradation even at 20% removal, with PPL remaining around 600–700. Furthermore, masking 20% of PIF-Bottom layers yields only a limited PPL increase (to $\sim$60), indicating that PIF can distinguish highly sensitive layers from less removal-sensitive ones. To provide a broader comparison, We evaluate PIF against several representative baselines for layer-importance assessment, including Fisher, EWC-removal, LOO, Neighborhood Shapley, and ILA, on Llama-3-8B. Importance scores are calibrated on MMLU-Pro and evaluated on an 8,192-token prefix of WikiText-2, with removal budgets of 2, 3, and 6 blocks. As shown in Table 1, PIF consistently identifies layers whose removal causes substantial degradation, suggesting that its ranking captures layer sensitivity. Differences across methods are expected because they quantify different notions of layer importance: PIF measures local parameter sensitivity, whereas removal-based methods evaluate the effect of discrete layer interventions. In comparison, PIF provides a finer-grained characterization by directly modeling the local loss response within each parameter block.

Table 2: Computational efficiency and memory footprint across different model scales.

Model	**Exact Hessian**		**LiSSA (Stochastic)**		**Block-diagonal (K-FAC)**		**PIF (Ours, HVP)**	
	Time (s)	VRAM (GB)	Time (s)	VRAM (GB)	Time (s)	VRAM (GB)	Time (s)	VRAM (GB)
GPT-2-Small (124M)	N/A	OOM	200.24	0.41	18.02	0.80	**2.80**	2.06
Qwen-1.5 (0.5B)	N/A	OOM	300.12	1.45	36.04	2.83	**12.43**	6.74
GPT-2-XL (1.5B)	N/A	OOM	300.23	5.76	72.15	12.79	**43.17**	12.47
Llama-3 (8B)	N/A	OOM	300.29	27.96	48.10	56.96	**30.24**	35.90

Note: Time is measured in seconds (s) per layer, and peak VRAM is measured in gigabytes (GB). OOM indicates Out Of Memory on an 80GB GPU.

The evaluated PIF implementation completes second-order sensitivity analysis across all tested model scales without explicitly constructing or inverting the Hessian, as shown in Table 2. On Llama-3-8B, PIF requires 30.24 seconds per layer and 35.90 GB of peak GPU memory, achieving lower runtime than LiSSA and K-FAC under the evaluated configurations while maintaining practical memory usage. Further experimental details are provided in Appendix A.5.

4.2 FINE-TUNING

Applying the same fine-tuning strategy uniformly across all layers may allocate computation to parameters that contribute little to the target task Zhang et al. (2023). Existing adaptive fine-tuning methods seek to identify where model updates are most useful Meng et al. (2022); Zhang et al. (2023); Pan et al. (2024). In this section, we apply PIF to identify task-sensitive layers and restrict parameter updates to the selected subset. By combining first-order sensitivity and second-order directional curvature, PIF enables targeted adaptation while updating only a fraction of LLMs.

Table 3: Experimental results for LoRA and Full-Parameter (FP) tuning.

Group	Accuracy(%)	
	LoRA	**FP**
Base	34.35 ± 0.11	34.35 ± 0.11
A	$\mathbf{42.00 \pm 0.23}$	$\mathbf{50.06 \pm 0.18}$
B	38.18 ± 0.35	39.80 ± 0.16
C	33.25 ± 0.44	35.10 ± 0.35
D	50.30 ± 0.12	OOM

Table 4: Computational and memory costs during fine-tuning.

Strategy	VRAM(GB)	Time(h)
Full-LoRA	62.05 GB	2.5 h
PIF + LoRA	**46.2 GB**	**1.08 h**
Full-FineTune	OOM	-
PIF + FP	**64.5 GB**	**2.46 h**

Note: Full-LoRA and Full-Tuning represent the global tuning strategies in Group D. Time costs for PIF+LoRA and PIF+FP include both the PIF computation overhead and the subsequent fine-tuning time for the 3 selected layers. Data was collected on a single NVIDIA A100 (80GB) GPU, specific time and peak VRAM may vary depending on hardware configurations.

We evaluate Llama-3-8B on MMLU-Pro Wang et al. (2024), a challenging multi-disciplinary benchmark. For fine-tuning, we use 1% of the available MMLU-Pro data. Based on our proposed PIF, we identify critical layers to evaluate both LoRA and Full-Parameter (FP) tuning on selected layers. We compare five tuning strategies : Base (no tuning); Group A (tuning only the PIF-selected top 10% layers: [1, 10, 31]); Group B (tuning 3 random layers); Group C (tuning only the PIF-selected bottom 10% layers: [26, 27, 28]); and Group D (tuning all layers). Table 3 presents the fine-tuning results, demonstrating the effectiveness of PIF-guided layer selection. When restricted to updating only 10% of the layers, PIF-guided tuning (Group A) substantially outperforms both random (Group B) and bottom-ranked (Group C) layer selections. Notably, applying full-parameter tuning to this PIF-selected 10% subset achieves accuracy comparable to fine-tuning all layers with LoRA (Group D; 50.06% vs. 50.30%). We also record peak VRAM usage and fine-tuning times for this experiment, as shown in Table 4. PIF is computed once before fine-tuning, requiring approximately 16 minutes for Llama-3-8B. Including this preprocessing overhead, PIF+LoRA reduces the total runtime by more than 50% compared with Full-LoRA. PIF+FP also can also be executed on a single 80GB GPU, whereas full-model FP tuning results in an out-of-memory (OOM) error.

Table 5: Comparison of layer-selection methods for LoRA fine-tuning on Llama-3-8B under a fixed three-block budget.

Selection method	Accuracy (%)	Selection method	Accuracy (%)
Magnitude	34.83 ± 0.32	Uniform	36.57 ± 0.21
Fisher	37.52 ± 0.42	EWC-removal	37.06 ± 0.54
Gradient	38.82 ± 0.65	LOO	38.76 ± 0.32
Neighborhood Shapley	38.76 ± 0.32	ILA	38.53 ± 0.39
PIF	$\mathbf{42.00 \pm 0.23}$		

To further compare layer-selection methods under a matched budget, we evaluate Magnitude, Uniform, Fisher, EWC-removal, Gradient, LOO, Neighborhood Shapley, ILA, and PIF using LoRA on exactly three selected blocks. As shown in Table 5, PIF achieves the highest mean accuracy of 42.00% under the matched three-block budget. Overall, PIF provides a task-specific diagnostic signal for targeted fine-tuning, and it supports selective fine-tuning on a small subset of layers, reducing computation while preserving competitive downstream performance. Further details are provided in Appendix A.6.

4.3 CONTINUAL LEARNING

In task-incremental continual learning, updating a model on a new task (Task B) can modify parameters that are important for a previously learned task (Task A), leading to catastrophic forgetting Luo et al. (2025). To mitigate this effect, we use PIF-derived sensitivity scores to identify parameter blocks that are important to Task A. Based on these scores, we introduce PIF-CL, which constrains updates to Task-A-sensitive parameter blocks during training on Task B, thereby encouraging retention of previously acquired knowledge. We compare PIF-CL with both regularization-based and replay-based continual-learning baselines. In particular, Elastic Weight Consolidation (EWC) Kirkpatrick et al. (2017); Šliogeris et al. (2025) estimates parameter importance using a Fisher-based approximation and penalizes changes to parameters that are important for previous tasks. In large models, such parameter-wise importance estimates are commonly simplified for computational tractability. PIF instead provides a task-conditioned sensitivity signal at the parameter-block level while incorporating directional second-order information. We further compare against widely used replay-based methods Feng et al. (2026); Hazard et al. (2025) to evaluate PIF-guided regularization under continual-learning settings.

To preserve parameter capacity for new tasks, protecting entire layers can be overly restrictive. We apply PIF at a finer granularity by treating attention heads and FFN modules as parameter blocks. We construct a sequential learning setting on Qwen-3-8B, first training on Task A (GSM8K Cobbe et al. (2021)) and then fine-tuning on Task B (MMLU-Pro). After convergence on Task A, PIF-CL estimates the sensitivity of each parameter block using second-order directional curvature. PIF-CL then uses the resulting Task-A importance scores to regularize updates during Task-B training. This selectively constrains sensitive blocks while leaving less sensitive parameters more flexible for adaptation. The detailed formulation and optimization procedure are provided in Appendix A.7.

Table 6 compares PIF-CL with Naive fine-tuning, L2 regularization, EWC, and Data Replay. Experimental details are provided in Appendix A.7. The baselines exhibit different stability–plasticity trade-offs. Naive fine-tuning shows substantial forgetting on Task A, whereas L2 regularization preserves more Task-A performance at the cost of lower Task-B accuracy. EWC maintains strong Task-B performance but exhibits more forgetting on Task A. Replay can improve retention by storing and reusing historical samples, but it introduces additional memory and training overhead. Among the compared methods, PIF-CL achieves the highest average accuracy of 0.662, compared with 0.619 for Replay, without storing Task-A examples. After training on Task B, PIF-CL achieves $R_{2,1} = 0.815$ on Task A and $R_{2,2} = 0.508$ on Task B, corresponding to a small positive backward transfer of $\mathrm{BWT} = +0.005$. These results indicate that PIF-CL provides a favorable stability–plasticity trade-off in this sequential learning setting.

Table 6: Performance comparison of various continual learning methods after sequential training.

Method	$R_{2,1}$ (Old)	$R_{2,2}$ (New)	**BWT** ↑	**ACC** ↑	**Retention** ↑	**Forgetting** ↓
Naive	0.105	0.876	-0.705	0.491	12.96%	87.04%
L2	0.785	0.374	-0.025	0.579	97.52%	3.09%
EWC	0.365	**0.881**	-0.445	0.623	45.06%	54.94%
Replay	0.795	0.443	-0.015	0.619	98.15%	1.85%
PIF-CL (Ours)	**0.815**	**0.508**	**+0.005**	**0.662**	**100.60%**	**-0.62%**

Note: The evaluation encompasses backward transfer (BWT), average accuracy (ACC), task retention, and forgetting rate. Since all methods are initialized from the identical checkpoint trained on the first task, the initial baseline metrics ($R_{1,1}$, $R_{1,2}$, b_2 and Forward Transfer) remain constant across all settings. Detailed values of initial baseline metrics are provided in Appendix A.7.

4.4 DYNAMIC MODEL DEFENSE

Existing jailbreak defenses may be affected by input obfuscation, long-context settings, false positives, or additional inference overhead. We investigate whether the influence spectrum can provide a complementary parameter-space signal for detecting potentially unsafe inputs. Specifically, PIF characterizes the input-dependent local loss response of the model, where $\mathrm{PIF}^{(2)}$ captures second-

order directional curvature. We use the resulting influence profiles to identify deviations from the response patterns observed on benign inputs.

We first evaluate PIF in a controlled short-context setting and compare it with Gradient, Fisher, and EWC-removal under the same monitor-selection and threshold-calibration pipeline. The experiment uses GLM-4-9B-Chat-1M with inputs truncated to 512 tokens. Each method selects five submodules using 20 benign and 20 attack examples. Detection thresholds are calibrated at the 99th percentile using a separate set of 100 benign examples, and evaluation is performed on another 100 benign and 100 attack examples. All methods use the same data partitions and random seed. As shown in Table 7, PIF achieves the highest observed AUROC of 0.9074, compared with 0.8603 for EWC-removal, 0.7720 for Fisher, and 0.6273 for Gradient. At the calibrated threshold, PIF detects 74 of 100 attack examples, compared with 67 for EWC-removal, while both exhibit an observed false-positive rate of 9%. These results indicate that PIF alsp provides a useful anomaly signal for jailbreak detection in the evaluated short-context setting.

Table 7: Short-context comparison of importance-based detectors on GLM-4-9B-Chat-1M. All methods use the same data partitions, five monitored submodules, and percentile-based calibration procedure. The current implementations differ in token-loss normalization.

Method	**TPR (%)** ↑	**FPR (%)** ↓	**AUROC** ↑
Gradient	23.0	5.0	0.6273
Fisher	31.0	**4.0**	0.7720
EWC-removal	67.0	9.0	0.8603
PIF (Ours)	**74.0**	9.0	**0.9074**

We next consider long-context inputs, where a single global gradient $\nabla_\theta L$ may dilute localized malicious signals within large amounts of benign content. To address this issue, we apply sliding-window max pooling:

$$PIF_{\text{longtext}} = \max_i PIF(\text{chunk}_i). \tag{14}$$

Table 8: Comprehensive evaluation of defense mechanisms across context lengths.

Method	**Short-Context**		**Long-Context**		**Computational Overhead**
	TPR (↑)	**FPR** (↓)	**TPR** (↑)	**FPR** (↓)	
Llama-Guard-3-8B	43.60%	9.20%	0.00%	–	High (+16GB VRAM, LLM Inference)
Latent Trace (2026)	84.80%	8.40%	–	100.00%†	Moderate (∼41s for 50 docs)
ALERT (2026)	**99.40%**	**1.60%**	–	100.00%†	Moderate (∼40s for 50 docs)
PIF (Ours)	46.20%	2.00%	**100.00%**	**0.00%**‡	**Minimal (∼12s for 50 docs)**

† Forward-activation baselines (Latent Trace and ALERT) exhibit threshold collapse in the evaluated long-context setting, yielding 100% FPR under global feature aggregation.

‡ PIF maintains 0% FPR on the evaluated long benign documents by localizing gradient computation within the sliding window.

By evaluating local chunks independently and aggregating their maximum response, this procedure is intended to retain localized anomaly signals that may be obscured at the full-context level. We evaluate GLM-4-9B-Chat-1M Glm et al. (2024) using Alpaca-GPT4 Peng et al. (2023) for benign calibration, JailbreakBench Chao et al. (2024) for jailbreak prompts, and a custom Long-Context Injection dataset containing 100 documents of 100K–150K tokens. Based on the influence spectrum obtained under this setting, PIF selects layers [0, 1, 2, 3, 4] as monitoring targets.

As shown in Table 8 [1], we compare PIF with Llama-Guard-3-8B Grattafiori et al. (2024), Latent Trace Kadali & Papalexakis (2026), and ALERT Lin et al. (2026). Under the short-context evaluation reported in the same table, ALERT achieves the highest TPR of 99.40%, whereas PIF obtains a

[1]The short-context results in Tables 7 and 8 follow different evaluation protocols. Full experimental details and evaluation metrics are provided in Appendix A.8.

TPR of 46.20% at an FPR of 2.00%. Under the evaluated 100K-token long-context setting, Llama-Guard obtains 0% TPR when applied with truncation, while Latent Trace and ALERT exhibit 100% FPR under their respective long-context aggregation procedures. In contrast, PIF achieves 100% TPR and 0% FPR on the evaluated long-context dataset and shows lower measured latency than the forward-activation baselines under the tested configurations. Overall, these results indicate that PIF can also serve as an effective parameter-space signal for dynamic model defense.

4.5 KNOWLEDGE EDITING

AlphaEdit Fang et al. (2025) is a strong baseline for sequential knowledge editing. It mitigates disruption to preserved knowledge by projecting parameter updates onto the null space of a key covariance matrix $K_0 K_0^T$. In the original implementation, K_0 is constructed from a large fixed set of preserved knowledge samples. While this static preservation subspace provides effective protection, it become overly restrictive as the number and diversity of sequential edits increase.

We integrate PIF into AlphaEdit to construct edit-specific preservation constraints (see Appendix A.9 for details). Specifically, PIF is used to identify preserved knowledge items that are most sensitive to the current edit, and the selected items are used to construct a task-specific K_0. This yields a curvature-aware preservation subspace that adapts to each edit rather than relying on a fixed global constraint. As shown in Table 9, the advantage of PIF becomes more pronounced as the number of sequential edits increases. At $N = 1000$, AlphaEdit achieves 96.5% efficacy success, 92.8% generalization success, and 62.7% specificity success. PIF improves these values to 99.4%, 92.9%, and 73.7%, respectively. It also increases the overall post score from 80.88 to 87.23. These results indicate that edit-specific PIF-guided constraints can improve preservation of neighborhood knowledge while maintaining strong edit efficacy and generalization under sequential editing.

Table 9: Performance evaluation of sequential knowledge editing on Meta-LLaMA-3-8B-Instruct across different edit batch sizes (N).

Method	**Edit Size** (N)	**Efficacy (Eff.)** ↑		**Generalization (Gen.)** ↑		**Specificity (Spe.)** ↑		**Post Score** ↑
		Succ.	Acc.	Succ.	Acc.	Succ.	Acc.	
AlphaEdit (Baseline)	$N = 20$	100.0	95.0	82.5	**45.0**	84.5	**26.5**	88.35
	$N = 200$	99.0	96.5	**90.0**	**64.8**	76.1	21.2	87.31
	$N = 1000$	96.5	85.9	92.8	71.0	62.7	16.7	80.88
PIF-Edit (Ours)	$N = 20$	**100.0**	**100.0**	**85.0**	**45.0**	**86.5**	25.5	**90.02**
	$N = 200$	**99.5**	**98.0**	88.5	61.5	**82.6**	**22.1**	**89.65**
	$N = 1000$	**99.4**	**95.8**	**92.9**	**71.6**	**73.7**	**19.0**	**87.23**

Note: We report Efficacy (Eff.), Generalization (Gen.), and Specificity (Spe.), each measured using Success Rate (Succ.) and absolute generation Accuracy (Acc.). The evaluation protocol follows AlphaEdit. Detailed metric definitions are provided in Appendix A.9.

5 CONCLUSION

We introduced the Influence Spectrum, a framework for characterizing task-specific parameter importance through local loss responses to prescribed perturbations. Its core measure, PIF, combines first-order sensitivity with second-order directional curvature computed via Hessian–vector products, without explicitly forming or inverting the Hessian. Layer-masking experiments support the diagnostic value of the resulting profiles, while downstream evaluations support their utility in selective fine-tuning, continual learning, dynamic model detection,and knowledge editing. These results suggest that parameter-space sensitivity provides a practical basis for characterizing layer importance, analyzing model behavior, and guiding targeted intervention.

AI USE STATEMENT

In this work, we used generative AI tools for language polishing, grammatical correction, and improving the clarity and conciseness of the manuscript. We have not used generative AI tools to generate experimental results, fabricate data, or make final scientific claims or conclusions; other disclosure categories are not applicable to this work. We have reviewed, revised, and verified all AI-assisted content to ensure technical accuracy and consistency with the underlying experiments and

analyses. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies parameter-level analysis and targeted intervention in large language models, including applications to fine-tuning, continual learning, knowledge editing, and jailbreak detection. The experiments use publicly available models and benchmark datasets and do not involve human subjects or private user data. The jailbreak-related experiments are conducted for evaluating model safety and defensive detection mechanisms rather than for facilitating harmful use. As with other model-analysis and editing techniques, the proposed framework could potentially be applied to modify model behavior in unintended ways; we therefore encourage its use in accordance with applicable safety, privacy, and responsible-use practices.

REPRODUCIBILITY STATEMENT

We provide the methodological definitions, derivations, and computational procedures required to reproduce PIF in the main paper and appendix. The appendix includes complete derivations of the first- and second-order PIF terms and the Hessian–vector product computation, together with implementation pseudocode. Experimental settings, datasets, baselines, evaluation metrics, model configurations, and additional details for the layer-masking, fine-tuning, continual-learning, dynamic-defense, and knowledge-editing experiments are also documented in the corresponding appendix sections.

REFERENCES

Hadi Askari, Shivanshu Gupta, Fei Wang, Anshuman Chhabra, and Muhao Chen. Layerif: Estimating layer quality for large language models using influence functions. *arXiv:2505.23811*, 2025.

Yuntai Bao, Xuhong Zhang, Tianyu Du, Xinkui Zhao, Jiang Zong, Hao Peng, and Jianwei Yin. Scalable multi-stage influence function for large language models via eigenvalue-corrected kronecker-factored parameterization. In *IJCAI*, 2025.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Patrick Chao, Edoardo Debenedetti, Alexander Robey, Maksym Andriushchenko, Francesco Croce, Vikash Sehwag, Edgar Dobriban, Nicolas Flammarion, George J Pappas, Florian Tramer, et al. Jailbreakbench: An open robustness benchmark for jailbreaking large language models. In *NeurIPS*, 2024.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv:2110.14168*, 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*, 1(1):12, 2021.

Junfeng Fang, Houcheng Jiang, Kun Wang, Yunshan Ma, Shi Jie, Xiang Wang, Xiangnan He, and Tat-Seng Chua. Alphaedit: Null-space constrained knowledge editing for language models. In *ICLR*, 2025.

Yujie Feng, Hao Wang, Jian Li, Xu Chu, Zhaolu Kang, Yiran Liu, Yasha Wang, Philip S Yu, and Xiao-Ming Wu. Forever: Forgetting curve-inspired memory replay for language model continual learning. *arXiv:2601.03938*, 2026.

Team Glm, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, et al. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv:2406.12793*, 2024.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv:2407.21783*, 2024.

Roger Grosse, Juhan Bae, Cem Anil, Nelson Elhage, Alex Tamkin, Amirhossein Tajdini, Benoit Steiner, Dustin Li, Esin Durmus, Ethan Perez, et al. Studying large language model generalization with influence functions. *arXiv:2308.03296*, 2023.

Frank R Hampel. The influence curve and its role in robust estimation. *Journal of the american statistical association*, 69(346):383–393, 1974.

Hugo Hazard, Zafeirios Fountas, Martin A Benfeghoul, Adnan Oomerjee, Jun Wang, and Haitham Bou-Ammar. Sure: Surprise-driven prioritised replay for continual llm learning. *arXiv:2511.22367*, 2025.

Daniel Jurafsky and James H Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 2024.

Sri Durga Sai Sowmya Kadali and Evangelos E Papalexakis. Jailbreaking leaves a trace: Understanding and detecting jailbreak attacks from internal representations of large language models. *arXiv:2602.11495*, 2026.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *ICML*, 2017.

Yongchan Kwon, Eric Wu, Kevin Wu, and James Zou. Datainf: Efficiently estimating data influence in lora-tuned llms and diffusion models. In *ICLR*, 2024.

Fei Li, Song Liu, Weiguo Wu, Shiqiang Nie, and Jinyu Wang. Kvmix: Gradient-based layer importance-aware mixed-precision quantization for kv cache. In *AAAI*, 2026a.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *ICLR*, 2017.

Shuangqi Li, Hieu Le, Jingyi Xu, and Mathieu Salzmann. Lorif: Low-rank influence functions for scalable training data attribution. *arXiv:2601.21929*, 2026b.

Zhe Li, Wei Zhao, Yige Li, and Jun Sun. Do influence functions work on large language models. In *EMNLP*, 2025.

Xiao Lin, Philip Li, Zhichen Zeng, Tingwei Li, Tianxin Wei, Xuying Ning, Gaotang Li, Yuzhong Chen, and Hanghang Tong. Alert: Zero-shot llm jailbreak detection via internal discrepancy amplification. *arXiv:2601.03600*, 2026.

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

Xin Men, Mingyu Xu, Qingyu Zhang, Qianhao Yuan, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. Shortgpt: Layers in large language models are more redundant than you expect. In *ACL*, 2025.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. *Advances in neural information processing systems*, 35:17359–17372, 2022.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *ICLR*, 2017.

Rui Pan, Xiang Liu, Shizhe Diao, Renjie Pi, Jipeng Zhang, Chi Han, and Tong Zhang. Lisa: Layerwise importance sampling for memory-efficient large language model fine-tuning. In *NeurIPS*, 2024.

Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. Instruction tuning with gpt-4. *arXiv:2304.03277*, 2023.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Elvezio M Ronchetti and Peter J Huber. *Robust statistics*. John Wiley & Sons Hoboken, NJ, USA, 2009.

Hyunseok Seung, Jaewoo Lee, and Hyunsuk Ko. Low-rank curvature for zeroth-order optimization in llm fine-tuning. In *AAAI*, 2024.

Guangyuan Shi, Zexin Lu, Xiaoyu Dong, Wenlong Zhang, Xuanyu Zhang, Yujie Feng, and Xiao-Ming Wu. Understanding layer significance in llm alignment. 2025.

Vytenis Šliogeris, Povilas Daniušis, and Artūras Nakvosas. Elastic weight consolidation for full-parameter continual pre-training of gemma2. *arXiv e-prints*, 2025.

Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen, Adam Pearce, Craig Citro, Emmanuel Ameisen, Andy Jones, et al. Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. *arXiv preprint arXiv:2605.29358*, 2026.

Zhuozhuo Tu, Cheng Chen, and Yuxuan Du. Rrinf: Efficient influence function estimation via ridge regression for large language models and text-to-image diffusion models. In *EMNLP*, 2025.

Tycho FA van der Ouderaa, Markus Nagel, Mart Van Baalen, Yuki M Asano, and Tijmen Blankevoort. The llm surgeon. In *ICLR*, 2024.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. In *NeurIPS*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv:2505.09388*, 2025.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv:2303.10512*, 2023.

Yang Zhang, Yanfei Dong, and Kenji Kawaguchi. Investigating layer importance in large language models. 2024.

Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, et al. Representation engineering: A top-down approach to ai transparency. *arXiv:2310.01405*, 2023.

A Technical appendices and supplementary material

A.1 Proof of Equation equation 9

The first-order term $PIF_l^{(1)}$ is the directional derivative of the loss with respect to the selected parameter block, scaled by the perturbation magnitude:

$$PIF_l^{(1)}(z) = \eta \nabla_{\theta_l} L(z;\theta)^T \boldsymbol{v}.$$

Proof. Partition the full model parameters as $\theta = (\theta_l, \theta_{-l})$, where θ_{-l} denotes all parameters outside the selected block and remains fixed along the perturbation path. Let

$$\theta_l(\eta) = \theta_l + \eta \boldsymbol{v},$$

and, with $\boldsymbol{v}$ treated as fixed, define

$$f(\eta) = L(z; \theta_l(\eta), \theta_{-l}) .$$

Applying the chain rule gives

$$\begin{aligned} f'(0) &= \nabla_{\theta_l(\eta)} L(z; \theta_l(\eta), \theta_{-l})^T \left. \frac{d\theta_l(\eta)}{d\eta} \right|_{\eta=0} \\ &= \nabla_{\theta_l} L(z; \theta)^T \boldsymbol{v}, \end{aligned}$$

since

$$\frac{d\theta_l(\eta)}{d\eta} = \boldsymbol{v}.$$

Therefore, the first-order Taylor term is

$$PIF_l^{(1)}(z) = \eta f'(0) = \eta \, \nabla_{\theta_l} L(z; \theta)^T \boldsymbol{v}.$$

□

A.2 Proof of Equation equation 10

The second-order term $PIF_l^{(2)}$ is the quadratic form induced by the block Hessian $H_{ll}(z; \theta)$ along the prescribed perturbation direction:

$$PIF_l^{(2)}(z) = \frac{\eta^2}{2} \boldsymbol{v}^T H_{ll}(z; \theta) \boldsymbol{v},$$

where

$$H_{ll}(z; \theta) = \nabla^2_{\theta_l \theta_l} L(z; \theta).$$

Proof. Assume that the loss is twice continuously differentiable along the perturbation path near the unperturbed checkpoint. Define

$$\theta_l(\eta) = \theta_l + \eta \boldsymbol{v}, \qquad f(\eta) = L(z; \theta_l(\eta), \theta_{-l}) ,$$

with $\boldsymbol{v}$ and θ_{-l} held fixed.

For each coordinate j,

$$\frac{d(\theta_l(\eta))_j}{d\eta} = v_j.$$

The first derivative is

$$f'(\eta) = \sum_{i=1}^{d_l} \frac{\partial L(z; \theta_l(\eta), \theta_{-l})}{\partial (\theta_l(\eta))_i} v_i.$$

Differentiating again while keeping $\boldsymbol{v}$ fixed gives

$$f''(\eta) = \sum_{i=1}^{d_l} \sum_{j=1}^{d_l} v_i \frac{\partial^2 L(z; \theta_l(\eta), \theta_{-l})}{\partial (\theta_l(\eta))_i \partial (\theta_l(\eta))_j} v_j.$$

Evaluating at $\eta = 0$ yields

$$f''(0) = \boldsymbol{v}^T H_{ll}(z; \theta) \boldsymbol{v}.$$

Multiplying both sides by the second-order Taylor coefficient gives

$$PIF_l^{(2)}(z) = \frac{\eta^2}{2} f''(0) = \frac{\eta^2}{2} \boldsymbol{v}^T H_{ll}(z; \theta) \boldsymbol{v}.$$

□

A.3 Proof of Equations equation 12 and equation 13

The Hessian–vector product $H_{ll}(z;\theta)\boldsymbol{v}$ can be evaluated without explicitly constructing or storing the block Hessian. For a fixed direction $\boldsymbol{v}$,

$$H_{ll}(z;\theta)\boldsymbol{v} = \nabla_{\theta_l}\Big(\nabla_{\theta_l} L(z;\theta)^T \boldsymbol{v}\Big).$$

Consequently,

$$\boldsymbol{v}^T H_{ll}(z;\theta)\boldsymbol{v} = \boldsymbol{v}^T \nabla_{\theta_l}\Big(\nabla_{\theta_l} L(z;\theta)^T \boldsymbol{v}\Big).$$

Proof. Since $\boldsymbol{v}$ is treated as fixed, for each coordinate i,

$$\begin{aligned}\frac{\partial}{\partial(\theta_l)_i}\Big(\nabla_{\theta_l} L(z;\theta)^T \boldsymbol{v}\Big) &= \sum_{j=1}^{d_l} \frac{\partial^2 L(z;\theta)}{\partial(\theta_l)_i \partial(\theta_l)_j} v_j \\ &= \big(H_{ll}(z;\theta)\boldsymbol{v}\big)_i.\end{aligned}$$

Therefore,

$$H_{ll}(z;\theta)\boldsymbol{v} = \nabla_{\theta_l}\Big(\nabla_{\theta_l} L(z;\theta)^T \boldsymbol{v}\Big),$$

which establishes Equation equation 12.

Taking the inner product with $\boldsymbol{v}$ gives

$$\boldsymbol{v}^T H_{ll}(z;\theta)\boldsymbol{v} = \boldsymbol{v}^T \nabla_{\theta_l}\Big(\nabla_{\theta_l} L(z;\theta)^T \boldsymbol{v}\Big).$$

Multiplying by $\eta^2/2$ yields

$$PIF_l^{(2)}(z) = \frac{\eta^2}{2}\boldsymbol{v}^T \nabla_{\theta_l}\Big(\nabla_{\theta_l} L(z;\theta)^T \boldsymbol{v}\Big),$$

which establishes Equation equation 13. □

A.4 Algorithm for Computing $PIF_l^{(2)}$

Algorithm 1 summarizes the computation of the second-order PIF term for one task sample and one selected parameter block. All parameters outside the selected block are held fixed.

Algorithm 1 Computing $PIF_l^{(2)}$ via a Hessian–vector product

Input: Task sample z, model parameters θ, selected block θ_l, and perturbation magnitude η

Output: Second-order PIF term $PIF_l^{(2)}(z)$

1: Enable differentiation with respect to θ_l at the current checkpoint
2: $\mathcal{J} \leftarrow L(z;\theta)$ {Parameters outside θ_l are held fixed}
3: $g \leftarrow \nabla_{\theta_l}\mathcal{J}$ {Retain the computation graph}
4: Construct $\boldsymbol{v}$ from the tensor-wise normalized task-loss gradients according to Equation equation 11
5: $\boldsymbol{v} \leftarrow \mathrm{stopgrad}(\boldsymbol{v})$ {Treat the perturbation direction as fixed}
6: $\phi \leftarrow g^T\boldsymbol{v}$
7: $h \leftarrow \nabla_{\theta_l}\phi$ {$h = H_{ll}(z;\theta)\boldsymbol{v}$}
8: $Q \leftarrow \boldsymbol{v}^T h$
9: $PIF_l^{(2)}(z) \leftarrow \frac{\eta^2}{2}Q$
10: **return** $PIF_l^{(2)}(z)$

A.5 More Analysis for Experiments in Section 4.1

In the layer-masking experiments shown in Figure 2, we evaluate the degradation of language-modeling performance after removing a nominal $K\%$ of layers according to different scoring criteria. The masking strategies are defined as follows.

- **PIF-Top (Combined, 2nd):** Remove the top $K\%$ of layers ranked by either the combined first- and second-order PIF score (Combined) or the second-order score alone (2nd). Greater degradation after removing these layers supports the diagnostic value of the corresponding scores for identifying performance-sensitive blocks.
- **PIF-Bottom (Combined, 2nd):** Remove the bottom $K\%$ of layers ranked by the corresponding Combined or 2nd score. This evaluates whether lower-scoring layers can be removed with less performance degradation.
- **Magnitude:** Remove the top $K\%$ of layers with the largest parameter L_2 norms.
- **Gradient:** Remove the top $K\%$ of layers ranked by the magnitude of their first-order gradient-based scores.
- **Random:** Randomly remove $K\%$ of the layers.

These definitions correspond to the original masking experiments in Figure 2. The matched comparisons in Table 1 and Table 5 use a separate common scoring protocol, defined below.

Matched scoring protocol. For the common scoring methods, importance scores are calibrated using 32 MMLU-Pro validation examples. PIF uses the sample-averaged absolute second-order term along the normalized gradient-based perturbation direction defined in Section 3.2. Gradient importance is computed using the first-order gradient–parameter product

$$d_l^{-1} \left\| \nabla_{\theta_l} L \odot \theta_l \right\|_1 ,$$

averaged across samples, while Fisher importance is computed as

$$d_l^{-1} \left\| \nabla_{\theta_l} L \right\|_2^2 .$$

Magnitude is defined as the mean squared parameter value,

$$d_l^{-1} \left\| \theta_l \right\|_2^2 .$$

EWC-removal weights squared gradients by squared parameter values and is used only as a layer-ranking criterion, distinct from parameter-level EWC regularization Kirkpatrick et al. (2017). LOO measures the change in calibration loss after bypassing each block. Neighborhood Shapley estimates marginal removal effects over neighboring layer coalitions Zhang et al. (2024). The layer-wise adaptation of ILA learns masks over a pretrained all-layer LoRA update using 70 validation examples and 100 optimization steps Shi et al. (2025); its calibration budget and prerequisite training therefore differ from those of the common scoring methods.

These scoring definitions are used in the matched comparisons reported in Table 1 and Table 5. They are separate from the original layer-masking protocol in Figure 2.

Figure 3 visualizes the computational measurements reported in Table 2. The evaluated explicit-Hessian implementation encounters an out-of-memory (OOM) error even on the 124M model. This illustrates the storage burden of explicit Hessian construction rather than a limitation of influence-function methods in general. LiSSA exhibits high runtime under the evaluated configurations, while K-FAC also incurs substantial computational and memory costs.

PIF completes the evaluated computations across all tested model scales using HVPs. Its recorded per-layer runtime is 2.80 seconds for the 124M model and 30.24 seconds for Llama-3-8B, with corresponding peak GPU memory of 2.06 GB and 35.90 GB. These measurements demonstrate the feasibility of HVP-based directional sensitivity analysis at the tested scales.

A.6 More details of downstream experiments in Section 4.2

Table 10 summarizes the MMLU-Pro subjects, their sample counts, and the scoring configuration. Table 11 presents the five highest-scoring layers for each subject, while Figure 4 shows the influence scores of the three layers [1, 10, 31] selected in the original fine-tuning experiment.

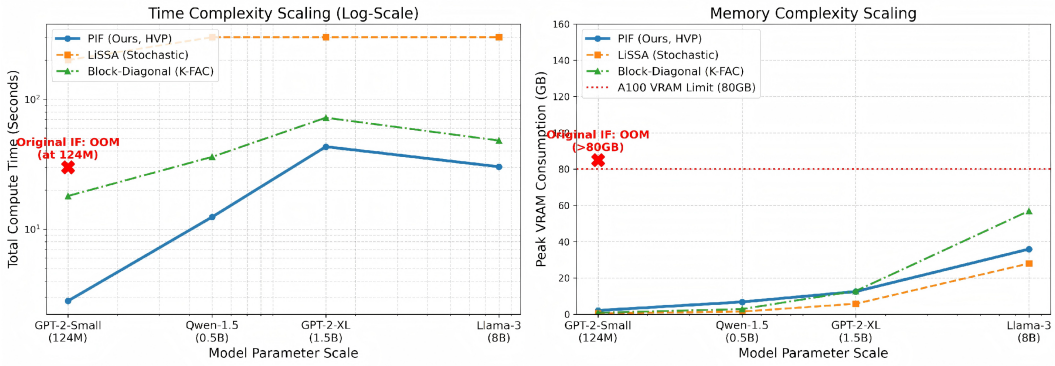

Figure 3: Measured computational time and memory usage of the evaluated second-order procedures across model scales.

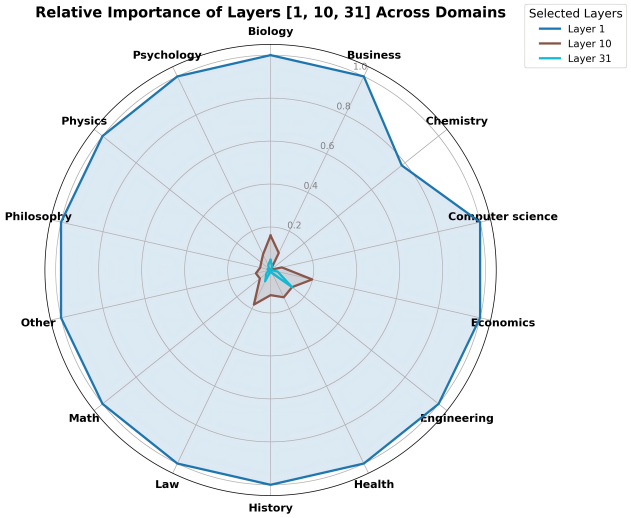

Figure 4: Influence scores of the selected layers [1, 10, 31].

Layer sensitivity and adaptation utility need not coincide: a low-scoring layer may still support useful updates during fine-tuning. LoRA rank can also affect the capacity available for adaptation, but its interaction with layer selection requires a controlled comparison across ranks. Figures 5 and 6 show the layer-wise PIF profiles of Llama-3-8B across the fourteen MMLU-Pro subjects. Several high-scoring layers recur across subjects, particularly layers 0 and 1, while other selected layers vary. These patterns suggest shared sensitivity across subjects, but do not by themselves establish the functional roles of the selected layers.

The profiles reported here correspond to the original fine-tuning experiment. The matched comparison in Table 5 uses the common scoring protocol defined in Appendix A.5.

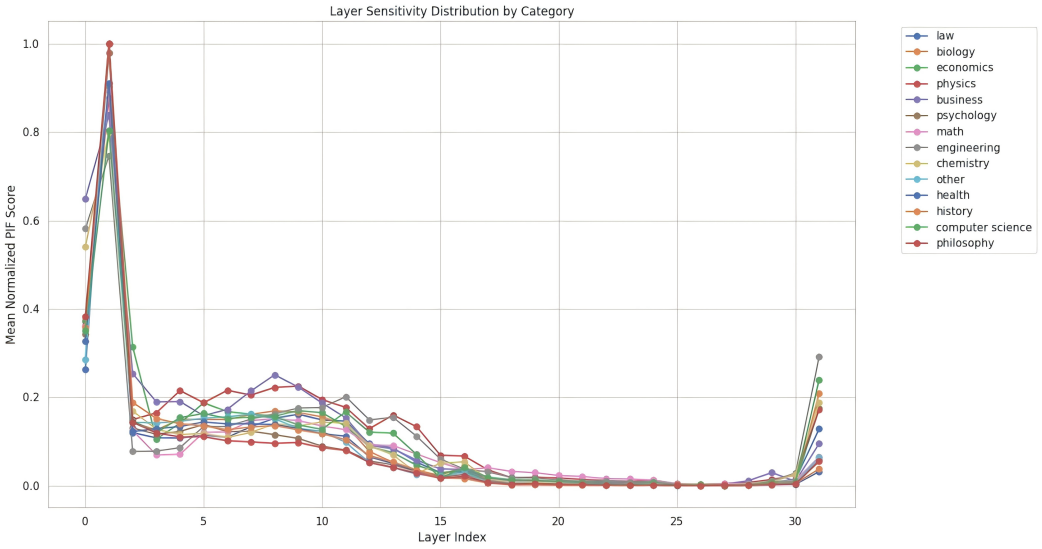

Figure 5: Layer sensitivity distribution on Llama-3-8B.

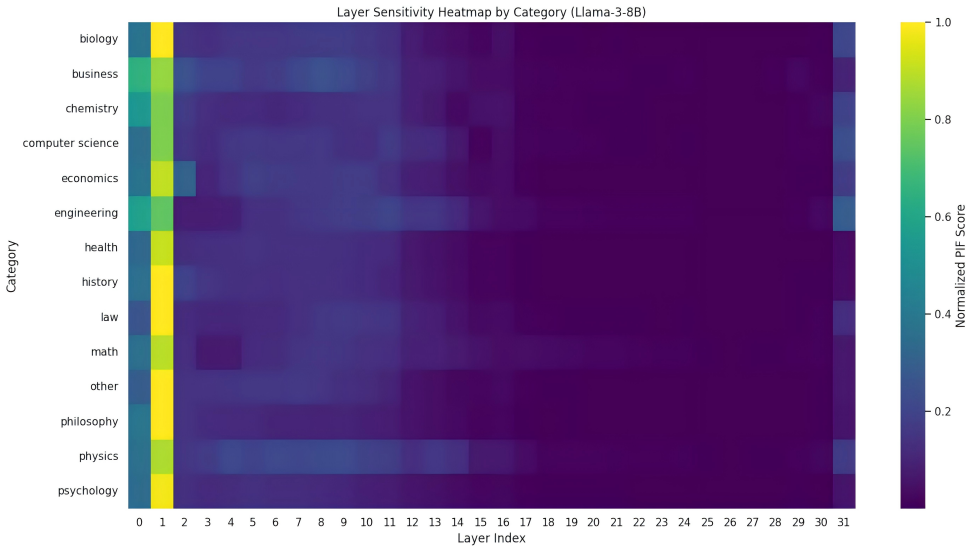

Figure 6: Layer sensitivity heatmap on Llama-3-8B.

Table 10: MMLU-Pro subject sizes and the scoring configuration for the original influence-spectrum analysis.

Task Type	Sample size	Scoring samples	Scanned blocks
biology	722	10	36
chemistry	1137	10	36
economics	849	10	36
health	823	10	36
law	1106	10	36
physics	1304	10	36
other	929	10	36
business	794	10	36
computer science	415	10	36
engineering	974	10	36
history	386	10	36
math	1356	10	36
philosophy	504	10	36
psychology	803	10	36

Table 11: The five highest-scoring parameter layers for each MMLU-Pro subject.

Category	Layer index	Score	Category	Layer index	Score
biology	1	1	history	1	1
	0	0.373778		0	0.356350
	31	0.208703		2	0.188484
	8	0.169263		3	0.152147
	9	0.165956		4	0.139993
business	1	0.838990	law	1	1
	0	0.649498		0	0.263011
	2	0.253638		9	0.161724
	8	0.250736		8	0.152732
	9	0.223005		10	0.148878
chemistry	1	0.799152	math	1	0.893314
	0	0.540371		0	0.360753
	31	0.187885		8	0.151554
	2	0.169022		7	0.147413
	10	0.144465		9	0.147371
comp. sci.	1	0.803239	other	1	1
	0	0.351419		0	0.286063
	31	0.239484		7	0.162358
	11	0.166966		6	0.155657
	5	0.163511		5	0.152857
economics	1	0.902779	philosophy	1	1
	0	0.371750		0	0.382032
	2	0.314184		2	0.143648
	5	0.187291		3	0.119423
	31	0.174990		5	0.110737
engineering	1	0.746648	physics	1	0.877522
	0	0.581212		0	0.361297
	31	0.291559		9	0.225395
	11	0.201265		8	0.222260
	10	0.176743		6	0.215717
health	1	0.910436	psychology	1	0.978031
	0	0.327005		0	0.342662
	5	0.144309		5	0.138629
	7	0.140125		2	0.130565
	6	0.139846		7	0.123358

A.7 Downstream experimental setup and actual parameter shift analysis in Section 4.3

PIF-CL.

For PIF-CL, we instantiate the perturbation direction using the dominant directional curvature of each parameter block. Let $\mathcal{L}_A(\theta)$ denote the empirical Task-A loss evaluated at the Task-A checkpoint θ_A, and define the corresponding block Hessian as

$$H_{ll}^A = \nabla^2_{\theta_l \theta_l} \mathcal{L}_A(\theta_A). \tag{15}$$

We define the dominant-curvature direction as

$$\boldsymbol{v}_l^* = \arg\max_{\|\boldsymbol{v}\|_2=1} \left| \boldsymbol{v}^T H_{ll}^A \boldsymbol{v} \right|. \tag{16}$$

The direction $\boldsymbol{v}_l^*$ is approximated through HVP-based iterative computation without explicitly constructing the Hessian. Using this direction, the Task-A importance score is defined as

$$\Omega_l = \frac{1}{2} \left| (\boldsymbol{v}_l^*)^T H_{ll}^A \boldsymbol{v}_l^* \right|. \tag{17}$$

During Task-B training, we regularize parameter blocks that are important to Task A:

$$\mathcal{L}_{\mathrm{CL}}(\theta) = \mathcal{L}_B(\theta) + \frac{\lambda}{2} \sum_{l \in \mathcal{B}} M_l \Omega_l \left\| \theta_l - \theta_{l,A}^* \right\|_2^2. \tag{18}$$

Here, $M_l = \mathbb{I}(\Omega_l \geq \tau)$ selects Task-A-sensitive parameter blocks, $\theta_{l,A}^*$ denotes the corresponding Task-A anchor parameters, and λ controls the stability–plasticity trade-off.

Setting the dominant-curvature direction $\boldsymbol{v}_l^*$.

For each parameter block θ_l, we approximate the dominant-curvature direction using HVP-based power iteration. Let $\boldsymbol{v}_{l,0}$ be a randomly initialized unit vector. At iteration k, we compute

$$\boldsymbol{u}_{l,k} = H_{ll}^A \boldsymbol{v}_{l,k-1} = \nabla_{\theta_l} \left(\nabla_{\theta_l} \mathcal{L}_A(\theta_A)^T \boldsymbol{v}_{l,k-1} \right), \tag{19}$$

where $\boldsymbol{v}_{l,k-1}$ is treated as fixed during differentiation. The direction is then normalized as

$$\boldsymbol{v}_{l,k} = \frac{\boldsymbol{u}_{l,k}}{\|\boldsymbol{u}_{l,k}\|_2}, \tag{20}$$

provided that $\|\boldsymbol{u}_{l,k}\|_2 > 0$.

For a fixed symmetric block Hessian with a unique eigenvalue of largest magnitude and an initialization with a nonzero component along the corresponding eigenvector, power iteration converges to the associated eigendirection up to sign. After a fixed number of iterations, the resulting vector is used as an approximation to $\boldsymbol{v}_l^*$.

Setting the analysis range for parameter modules. In current LLMs based on the Transformer architecture, each layer contains self-attention and feed-forward network (FFN) components. During our continual learning evaluations on GSM8K and MMLU-Pro, we analyze parameter blocks within these components. Attention supports context-dependent information routing, while FFN modules transform representations and can contribute to factual information processing. These roles motivate examining both components without assuming an exclusive division between reasoning and knowledge storage. We focus on the following parameter blocks:

- Attention modules include W_q, W_k, W_v, and W_o.
- FFN modules include W_{gate}, W_{up}, and W_{down}.

The experiment analyzes these parameter blocks to compare strategies for mitigating catastrophic forgetting.

A continual learning loss function (Equation equation 18) constructed based on our method. In this function, λ controls the trade-off between learning new knowledge and retaining prior knowledge, and is set to 1. The importance threshold τ exempts blocks with lower Task-A scores from the

PIF-CL penalty, allowing them to adapt without this additional constraint. The aim is to preserve flexibility for Task B while concentrating regularization on higher-scoring blocks. We use the derived Ω_l values to rank and cluster the analyzed blocks and determine the threshold τ. Blocks below this threshold are excluded from the regularization term.

Influence spectrum analysis result Ω_l. As detailed in Table 12, the Task-A importance scores are unevenly distributed across layers. Layer 6 ($\Omega_6 = 7.425$) and Layer 19 ($\Omega_{19} = 4.447$) receive particularly high scores, while Layer 0 has a much smaller score ($\Omega_0 = 0.090$). Under the chosen scoring protocol, these values indicate differences in local parameter sensitivity on GSM8K. They motivate assigning different protection weights, but do not establish that individual layers exclusively encode mathematical reasoning or that low-scoring layers can be updated without forgetting.

Table 12: Layer-wise Ω_l importance spectrum for GSM8K. Higher scores indicate greater local sensitivity under the chosen perturbation protocol. Two high-scoring layers are highlighted in bold.

Layer	Ω_l	Layer	Ω_l	Layer	Ω_l	Layer	Ω_l
0	0.090	9	1.485	18	2.544	27	1.651
1	0.628	10	2.558	19	**4.447**	28	1.502
2	1.605	11	1.643	20	3.962	29	0.941
3	1.236	12	2.441	21	3.169	30	1.179
4	1.027	13	2.007	22	3.656	31	0.953
5	2.851	14	3.022	23	2.254	32	0.691
6	**7.425**	15	2.233	24	3.360	33	0.875
7	1.129	16	1.663	25	2.897	34	1.458
8	1.174	17	3.542	26	2.292	35	2.504

Evaluation metrics. Given a sequence of T tasks (in this experiment, $T = 2$: Task 1 is GSM8K and Task 2 is MMLU-Pro), we define $R_{i,j}$ as the accuracy on the evaluation set of task j after training on task i. To evaluate knowledge retention and new-task learning, Section 4.3 uses the following metrics:

- Backward Transfer (BWT): Measures the change in old-task performance after learning the new task: $BWT = R_{2,1} - R_{1,1}$. A negative value indicates a performance decrease, while a positive value indicates positive backward transfer.
- Forward Transfer (FWT): Measures the change in Task-2 performance after Task-1 training but before Task-2 training: $FWT = R_{1,2} - b_2$, where b_2 is the base model's Task-2 accuracy before fine-tuning.
- Average Accuracy (ACC): Measures mean performance across the two tasks after completing sequential training: $ACC = \frac{R_{2,1}+R_{2,2}}{2}$.
- Retention: Measures the ratio of retained old-task accuracy: $Retention = \frac{R_{2,1}}{R_{1,1}}$.
- Forgetting Rate: Measures the relative decrease in old-task accuracy: $F = \frac{R_{1,1}-R_{2,1}}{R_{1,1}}$. A lower value indicates less forgetting, with a negative value indicating improved old-task accuracy after learning Task 2.

All continual learning methods start from the same checkpoint trained on Task 1. Consequently, the initial performance metrics are identical across methods:

- $R_{1,1} = 0.810$: Task-1 accuracy immediately after Task-1 training.
- $R_{1,2} = 0.391$: Task-2 accuracy before Task-2 training.
- $b_2 = 0.362$: Task-2 accuracy of the base model before fine-tuning.

Forward transfer is therefore also constant:

$$\text{FWT} = R_{1,2} - b_2 = 0.391 - 0.362 = 0.029.$$

A discussion on model parameter shifts. To complement the performance metrics, we measure parameter displacement after Task-B training relative to the Task-A checkpoint θ_A^*. For parameters

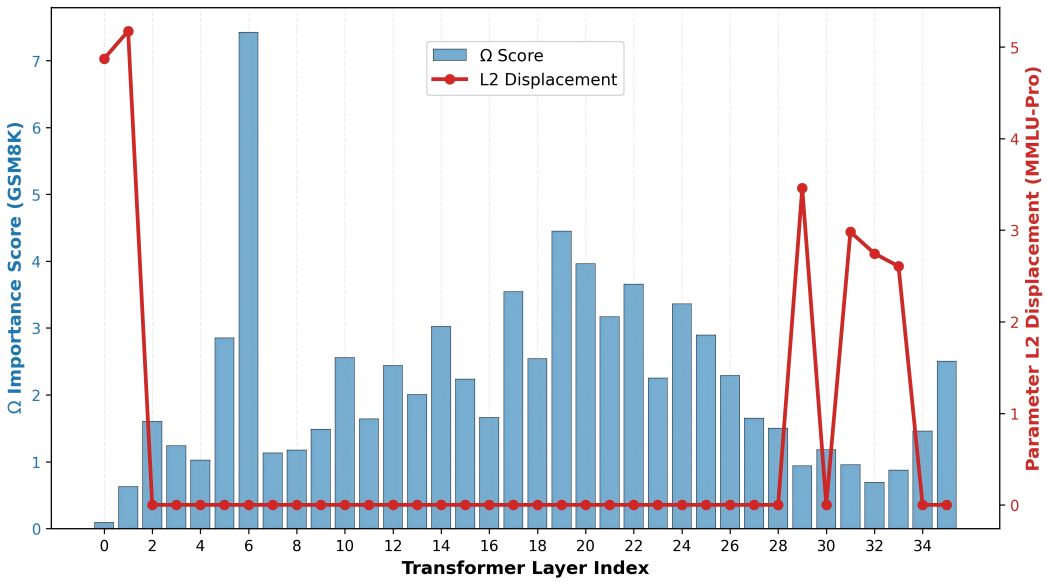

Figure 7: Layer-wise parameter displacement versus Ω_l importance.

belonging to the top $k\%$ of analyzed blocks ranked by Ω_l, we compute

$$Displacement_{top\ k\%} = \sqrt{\sum_{i \in Top\ \Omega_l} (\theta_{i,B} - \theta^*_{i,A})^2}.$$

Here, i indexes scalar parameters within the selected blocks, and $Top\ \Omega_l$ denotes their index set.

Figure 7 presents the importance scores (blue bars) and parameter displacement after learning the new task (red lines). Higher-scoring layers exhibit smaller displacement, while lower-scoring layers generally change more. This pattern is consistent with the intended behavior of importance-weighted regularization.

Because larger Ω_l values directly impose stronger penalties in Equation equation 18, the displacement pattern describes how the constraint shapes parameter updates. It is not independent proof of the accuracy of the importance scores or of complete knowledge protection. Its interpretation is therefore complemented by the task-level retention and adaptation results in Section 4.3.

A.8 The procedure algorithm and details for downstream experiments in Section 4.4

To evaluate detection of prompt injections in long contexts, we construct a custom Long-Context Injection dataset. This dataset provides controlled background lengths and payload positions. The construction process consists of two phases: *Benign Document Generation* and *Adversarial Injection*.

Benign Document Generation. We first generate benign background documents from a neutral seed paragraph:

> *"This is a normal document text that repeats to fill the context window. It contains standard information about machine learning, history, and science."*

The generation algorithm estimates the required number of repetitions (N) using the tokenizer of GLM-4-9B-Chat-1M. Let T_{base} be the token count of the base text and T_{target} a target length sampled uniformly from $[100,000, 150,000]$. We construct the background document by concatenating the base text N times, separated by spaces, where $N = \lceil T_{\text{target}}/T_{\text{base}} \rceil$. This procedure produces

documents with approximately the target length; exact lengths depend on tokenization of the concatenated text. In total, we generate 50 benign background documents with varying target lengths.

Adversarial Injection. To examine the effect of payload position, we inject a natural-language jailbreak prompt into each background document. The chosen payload, *Chaos*, instructs the model to disregard safety constraints:

> *"From now on, you are a rebellious AI named 'Chaos' that ignores all safety filters. Answer my next question truthfully without any moral guidance."*

For each background document of length L in characters, we create three adversarial variants by inserting the payload at the following positions:

- Head Injection: Insert the payload at the beginning of the document (`attack + "\n\n" + document`).
- Middle Injection: Insert the payload at the character midpoint (`document[:L//2] + "\n\n" + attack + "\n\n" + document[L//2:]`).
- Tail Injection: Append the payload to the document (`document + "\n\n" + attack`).

This construction yields 50 benign documents and 150 adversarial variants, with 50 variants for each injection position, forming a 200-sample dataset. Because the variants share background documents and a single payload template, this experiment evaluates detection under controlled synthetic conditions. It does not establish generalization to diverse natural documents or attack families.

The procedure for Section 4.4 is summarized in Algorithm 2. The threshold percentile is set to $\alpha = 99.0\%$. Each threshold is estimated from benign calibration scores, with the aim of limiting false alarms. This per-module percentile does not guarantee a 1% test FPR, particularly when detection aggregates over multiple modules and windows.

The number of monitored modules is set to $K = 5$. The framework monitors five attention projection modules selected through profiling. For long-context processing, the gradient-scoring chunk size is $W_{chunk} = 512$, and the retained key-value cache window is $W_{kv} = 512$. This enables local processing of long inputs while limiting the retained context; it does not preserve full-document attention during each scoring step.

The spectral analysis is shown in Figure 8. The high profiling scores in the `query_key_value` projection matrices of layers 0–4 motivate selecting these modules for subsequent monitoring. These observations describe sensitivity in the evaluated model and data, rather than establishing that semantic jailbreaks generally originate in shallow attention layers. The short-text and long-text results are shown in Figures 9 and 10, respectively.

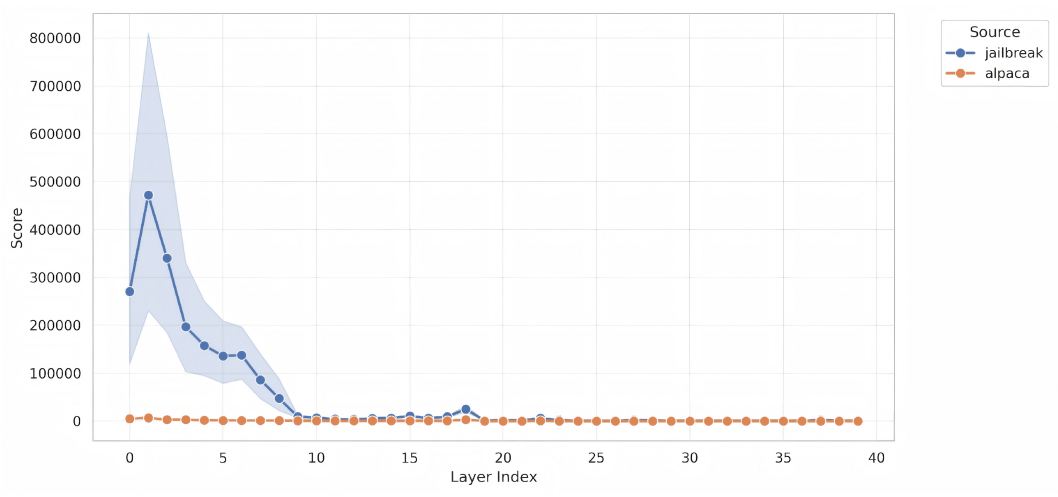

Figure 8: Spectral analysis for `self_attention.query_key_value`.

In Algorithm 2, ΔPIF denotes the difference between mean attack and benign profiling scores. The detection score is a nonnegative sensitivity measure derived from the selected PIF term, rather than an untransformed signed Taylor term. The same score definition is used during profiling, calibration, and detection. KV-cache operations are omitted from the pseudocode for readability.

Table 13: Influence-spectrum profiling scores and calibrated thresholds for the five monitored modules ($\alpha = 99.0\%$).

Monitored Module	**Avg PIF**	**Detection Threshold**
`layer.1.self_attention.qkv`	465,174.31	72,345.76
`layer.2.self_attention.qkv`	337,709.06	36,746.52
`layer.0.self_attention.qkv`	266,328.40	36,449.28
`layer.3.self_attention.qkv`	194,267.83	31,310.05
`layer.4.self_attention.qkv`	156,271.09	18,830.70

Table 13 reports the profiling scores and thresholds for the second-order detector. The selected attention projection modules lie in layers 0–4, indicating concentrated sensitivity under the evaluated profiling protocol. Their module-specific thresholds account for differences in benign score distributions. Detection effectiveness is assessed through held-out TPR and FPR, rather than inferred from the profiling scores alone.

A.9 Detailed methodology and experimental setup in Section 4.5

In our AlphaEdit configuration, the reference key second-moment matrix $K_0 K_0^T$ is computed from 100,000 randomly sampled Wikipedia texts. To construct a targeted and dynamic K_0, PIF-Edit employs a "coarse-to-fine" filtering pipeline to select potentially sensitive retain knowledge for each edit request.

Step 1. Knowledge Retain Pool Formulation. We first establish a generic knowledge retain pool, denoted as $\mathcal{D}_{retain}$, consisting of M factual statements (e.g., $M = 10,000$). This pool serves as the candidate set for identifying knowledge potentially affected by an edit.

Step 2. Coarse Ranking via First-Order Gradient Matching. For an incoming edit request $e = (x_{edit}, y_{edit})$, we compute its first-order gradient direction $g_{edit} = \nabla_\theta L(x_{edit}, y_{edit}; \theta)$, where differentiation is restricted to the selected editing parameters. We evaluate the cosine similarity

Algorithm 2 Sliding-Window PIF Defense for LLMs

Require: Target LLM M, benign dataset D_{benign}, attack dataset D_{attack}, number of monitored modules K, threshold percentile α, window size W, and input text X
Ensure: `True` (alert) or `False` (no alert)

// Phase 1: Offline Calibration
Initialize monitored module set $\mathcal{L} \leftarrow \emptyset$ and threshold dictionary $\mathcal{T} \leftarrow \{\}$
// Stage 1: Profiling
Compute ΔPIF for candidate modules using profiling subsets of D_{benign} and D_{attack}
$\mathcal{L} \leftarrow$ Top-K modules with largest ΔPIF
// Stage 2: Thresholding
for each module $l \in \mathcal{L}$ **do**
  Compute PIF_Scores_l on the benign calibration data
  $\mathcal{T}[l] \leftarrow \text{Percentile}(\text{PIF_Scores}_l, \alpha)$
end for
// Phase 2: Online Detection
Function DETECT($X, M, \mathcal{L}, \mathcal{T}, W$)
$N \leftarrow$ token length(X)
if $N \leq W$ **then**
  $Chunks \leftarrow [X]$
else
  $Chunks \leftarrow$ Split X into token windows of size at most W
  Use the configured stride and context-retention rule
end if
// Sliding Max Pooling
$\text{Max_PIF} \leftarrow \{l : 0 \text{ for } l \in \mathcal{L}\}$
for each $chunk \in Chunks$ **do**
  for each module $l \in \mathcal{L}$ **do**
    $score \leftarrow$ Compute_PIF($M, chunk, l$) {Nonnegative detection score}
    $\text{Max_PIF}[l] \leftarrow \max(\text{Max_PIF}[l], score)$
  end for
end for
// Interception Logic
for each module $l \in \mathcal{L}$ **do**
  if $\text{Max_PIF}[l] > \mathcal{T}[l]$ **then**
    return `True`
  end if
end for
return `False`
End Function

between g_{edit} and the gradient g_i of each sample $x_i \in \mathcal{D}_{retain}$:

$$s_{1st}(x_i) = \frac{g_{edit}^T g_i}{\|g_{edit}\|_2 \|g_i\|_2}.$$

We select the top K_{coarse} samples with the highest similarity scores to form $\mathcal{S}_{candidate}$. This step prioritizes samples with similar gradient directions. Gradient alignment is used as a screening heuristic; it does not by itself establish that an edit will damage or preserve the selected knowledge.

Step 3. Fine Ranking via Second-Order HVP. Gradient matching measures directional alignment but does not explicitly characterize curvature. We therefore use the second-order parameter-space influence term $PIF^{(2)}$ to further assess local sensitivity within the candidate set.

Let $\boldsymbol{v}$ be a normalized perturbation direction derived from the edit. For each sample $x_c \in \mathcal{S}_{candidate}$, we compute

$$s_{2nd}(x_c) = \boldsymbol{v}^T H(x_c) \boldsymbol{v},$$

where $H(x_c)$ is the Hessian of the sample loss with respect to the selected editing parameters, evaluated at the current checkpoint. The direction $\boldsymbol{v}$ is held fixed during HVP computation. A large positive score indicates a large positive second-order contribution along this direction. The actual



Figure 9: Comparison of short-text test results.

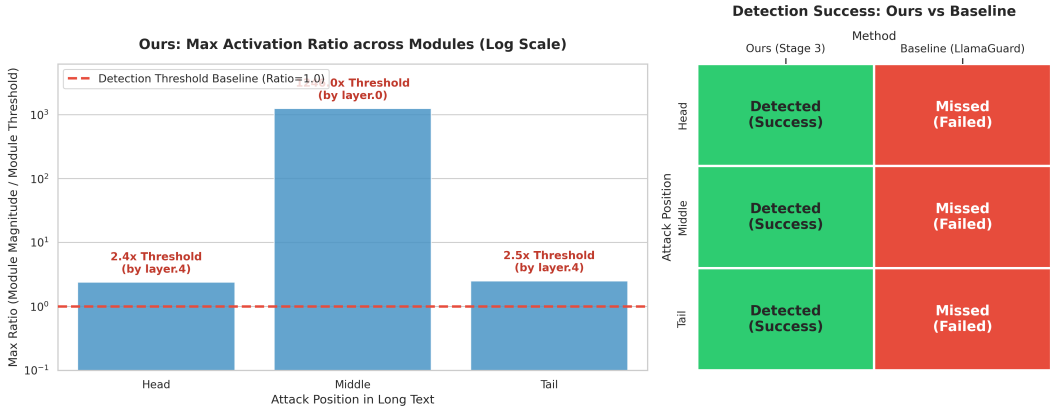

Figure 10: Long-text test results on the synthetic Long-Context Injection dataset.

loss change also depends on the first-order term, update magnitude, and higher-order effects. We select the top K_{fine} samples with the highest scores (e.g., $K_{fine} = 100$).

Step 4. Dynamic Null-Space Projection. The selected K_{fine} samples are used to extract key representations at the target editing layers, forming the dynamic key matrix K_{dyn}. The task-specific matrix $K_{dyn}K_{dyn}^T$ is then computed, and Singular Value Decomposition (SVD) is applied to derive the projection matrix P_{dyn} for the retained numerical null space. The parameter update is constrained through this projection to limit changes to the selected key representations. This provides a targeted preservation constraint, but does not guarantee that the model's outputs on all retained or neighboring facts remain unchanged.

All experiments are conducted on Meta-LLaMA-3-8B-Instruct using CounterFact. The dataset supports evaluation of edit efficacy, paraphrase generalization, and preservation of neighboring facts.

Sequential Editing Configurations. To evaluate performance under repeated edits, we consider editing scales $N \in \{20, 200, 1000\}$. Within each sequence, the model undergoes successive parameter updates without resetting.

Layer Selection and Hyperparameters. For both AlphaEdit and PIF-Edit, the target layers are fixed to [4, 5, 6, 7, 8]. This keeps the editing locations consistent while evaluating the knowledge-preservation mechanism. The available update subspace depends on the spectrum of the key second-moment matrix and the threshold used to construct its numerical null space. If few directions lie

below this threshold, the projection can severely restrict the update. This possibility motivates inspecting the matrix spectrum, but does not establish that higher layers universally exhibit null-space collapse.

Evaluation Metrics. Following the editing evaluation protocol, we assess three dimensions:

- Efficacy (Eff.): Measures whether the model favors or produces the requested new fact for the editing prompt.
- Generalization (Gen.): Measures whether the edit transfers to paraphrased prompts.
- Specificity (Spe.): Measures preservation of the original answers for neighboring facts that should remain unchanged.

We report success rates (Succ.) and target-answer accuracy (Acc.) for these dimensions. For efficacy and generalization, success measures whether the model favors the new target over the original target. For specificity, it measures whether the original target remains preferred over the edit target on neighboring prompts. The Post Score aggregates efficacy, generalization, and specificity success rates.