

Beyond Task Boundaries: Instruction-Level Parameter Isolation for Supervised Fine-Tuning

Zekai Lin^{1*}, Long Xiang^{3*}, Di Liang^{1,2†}, Peiyang Liu⁴,
Zhanghao Hu^{2,5}, Mengze Hong², Xianjie Wu⁶, Minlong Peng^{1,2†}

¹ Tencent Hunyuan, ² Tencent Yuanbao, ³ Sun Yat-sen University

⁴ Peking University, ⁵ King's College London

⁶ Beijing Information Science & Technology University

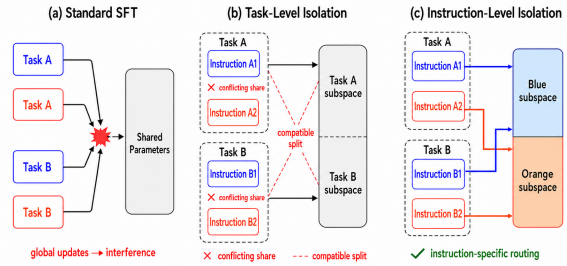
{rcjgdds, islongxiang}@gmail.com; {liangd17, mlpeng16}@fudan.edu.cn

Abstract

Modern supervised fine-tuning (SFT) trains large language models on heterogeneous instruction mixtures, where the effective unit of optimization is an individual instruction-response pair rather than a coarse task label. Existing parameter isolation methods mitigate interference by assigning protected or trainable subspaces at the task or training stage level, but these units are often misaligned with actual parameter usage: instructions from the same task may require conflicting computations, while instructions from different tasks may share latent capabilities. We propose **Instruction-Level Parameter Isolation (IPI)**, a fine-tuning framework that routes each instruction's update to a sparse, capability-conditioned parameter subspace. IPI builds a multi-view instruction representation, predicts importance factors with a lightweight router, and uses an identity-trick reparameterization that preserves dense forward computation while exposing sparse base-model gradients and a task-loss saliency signal for router learning. Across heterogeneous instruction tuning benchmarks and multiple backbone models, IPI consistently improves performance over strong fine-tuning and parameter-isolation baselines. Further analyses show that instruction-conditioned masks reduce destructive gradient interactions and capture optimization structure that task labels miss.

1 Introduction

Supervised fine-tuning (SFT) adapts large language models to instruction following by optimizing on heterogeneous instruction-response pairs. In practice, these instructions often span diverse capability domains, such as reasoning, coding, knowledge use, and open-ended dialogue, while still updating the same shared parameter space. Although such



Interference follows instructions, not task labels.

Figure 1: Motivation of IPI. Full SFT shares all parameters across heterogeneous instructions. Task-level isolation uses fixed task masks, which can miss intra-task heterogeneity and cross-task sharing. IPI predicts an instruction-conditioned sparse update mask.

diversity improves capability coverage, it also creates parameter-level contention: an update that improves one capability may move shared parameters in a direction that harms another. This parameter-level contention can manifest as the “seesaw phenomenon” (Yu et al., 2020) and, especially in sequential or multi-stage fine-tuning, catastrophic forgetting (McCloskey and Cohen, 1989; Kirkpatrick et al., 2017; Xue et al., 2026b).

Prior parameter isolation methods mitigate interference by identifying important parameters and selectively protecting or updating them during fine-tuning. A common strategy is to estimate a fixed mask for each task or training stage, which we refer to as static task-level isolation (Wang et al., 2025; Sorrenti et al., 2023). More recent dynamic isolation methods (Lin et al., 2026) update a global mask over training time. Figure 1 contrasts these two paradigms with instruction-level routing. While effective in reducing coarse interference, both paradigms still rely on task identity or global training-stage statistics as proxies for parameter usage. This assumption is often too coarse for instruction tuning.

The key issue is a mismatch between the unit

¹* Equal Contribution. [†] Corresponding Author.

²This work was completed by Zekai Lin and Long Xiang under Di Liang’s supervision.

of optimization and the unit of isolation. SFT optimizes individual instruction-response pairs, whereas task-level isolation groups many heterogeneous instructions under the same surface task label. A single dataset may contain examples requiring direct arithmetic, symbolic reasoning, factual retrieval, and open-ended generation. Conversely, instructions from different datasets may invoke similar latent computations. Therefore, task labels do not reliably indicate which parameters should be updated or protected for a particular instruction. We argue that interference in heterogeneous instruction tuning is fundamentally **instruction-level**. The relevant question is not merely which dataset an example belongs to, but which latent capability the specific instruction invokes. This perspective suggests that parameter updates should be routed at the instance level: compatible instructions should be allowed to share update regions even across task boundaries, while conflicting instructions within the same task should be separated.

To this end, we propose **Instruction-Level Parameter Isolation (IPI)**, a fine-grained framework that predicts a sparse update mask for each instruction. IPI builds a multi-view capability representation from the instruction query, optional input context, and a short response prefix. A lightweight structured router maps this representation to row and column importance factors for each routed weight matrix, whose outer product defines an implicit per-weight importance score. A hard Top- k mask then selects the sparse update subspace for the current instruction. A central design goal of IPI is to isolate updates without damaging the pre-trained forward computation. Therefore, IPI does not mask the forward model. Instead, it introduces an identity-trick reparameterization that preserves the dense forward pass exactly while defining two backward paths: masked gradients for updating the base model and a first-order task-loss saliency signal for training the router. As a result, IPI performs dense inference-compatible tuning while learning instruction-conditioned sparse update routes.

We evaluate IPI on heterogeneous instruction tuning benchmarks covering mathematical reasoning, logical reasoning, code generation, instruction following, and dialogue. Across multiple backbone models, IPI consistently improves over strong fine-tuning and parameter-isolation baselines. Further analyses show that instruction-conditioned masks reduce destructive gradient interactions and better reflect optimization similarity than surface task

labels.

Contributions. This work makes three contributions. First, we identify a unit mismatch in parameter isolation for heterogeneous SFT: task labels and global training-stage masks are unreliable proxies for the parameter usage of individual instructions. Second, we propose IPI, an instruction-conditioned sparse update routing framework that preserves dense forward computation while learning routing decisions from task-loss saliency through an identity-trick gradient bridge. Third, we empirically show that instruction-level routing improves heterogeneous SFT across model families, reduces destructive gradient interactions, and captures optimization patterns that task-level isolation misses.

2 Method

We propose **Instruction-Level Parameter Isolation (IPI)**, a fine-tuning framework that routes each instruction’s update to a sparse parameter subspace while preserving the dense pretrained forward computation. Figure 2 illustrates the overall pipeline. For each mini-batch, IPI builds instruction-level capability representations, uses a structured low-rank router to predict row-column factors and Top- k update masks, and applies an identity-trick reparameterization so that the forward pass remains dense while backward updates are sparsely routed. The base model receives per-instance masked gradients, the router is trained by a task-loss saliency signal and load-balancing regularization, and the final AdamW (Loshchilov and Hutter, 2017) step is gated to prevent unselected coordinates from being modified. The router is used only during training and is discarded after fine-tuning, leaving a dense model at inference. Detailed gradient derivations, factorized Top- k selection, implementation details, and the complete training algorithm are provided in Appendix C and Appendix C.4.

2.1 Preliminaries and Problem Setup

Let $\theta \in \mathbb{R}^d$ denote the parameters of a pre-trained language model and $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$ a supervised fine-tuning dataset, with per-example loss $\mathcal{L}_i(\theta) = -\log p(\mathbf{y}_i \mid \mathbf{x}_i; \theta)$. Parameter isolation introduces a binary update mask $\mathbf{m} \in \{0, 1\}^d$, where $m_j = 1$ means that coordinate j is allowed to update. Static task-level isolation (Mallya and Lazebnik, 2018; Wang et al., 2025) assigns one fixed mask $\mathbf{m}_t$ to all examples in task t , implicitly assuming that task identity is a reliable proxy for

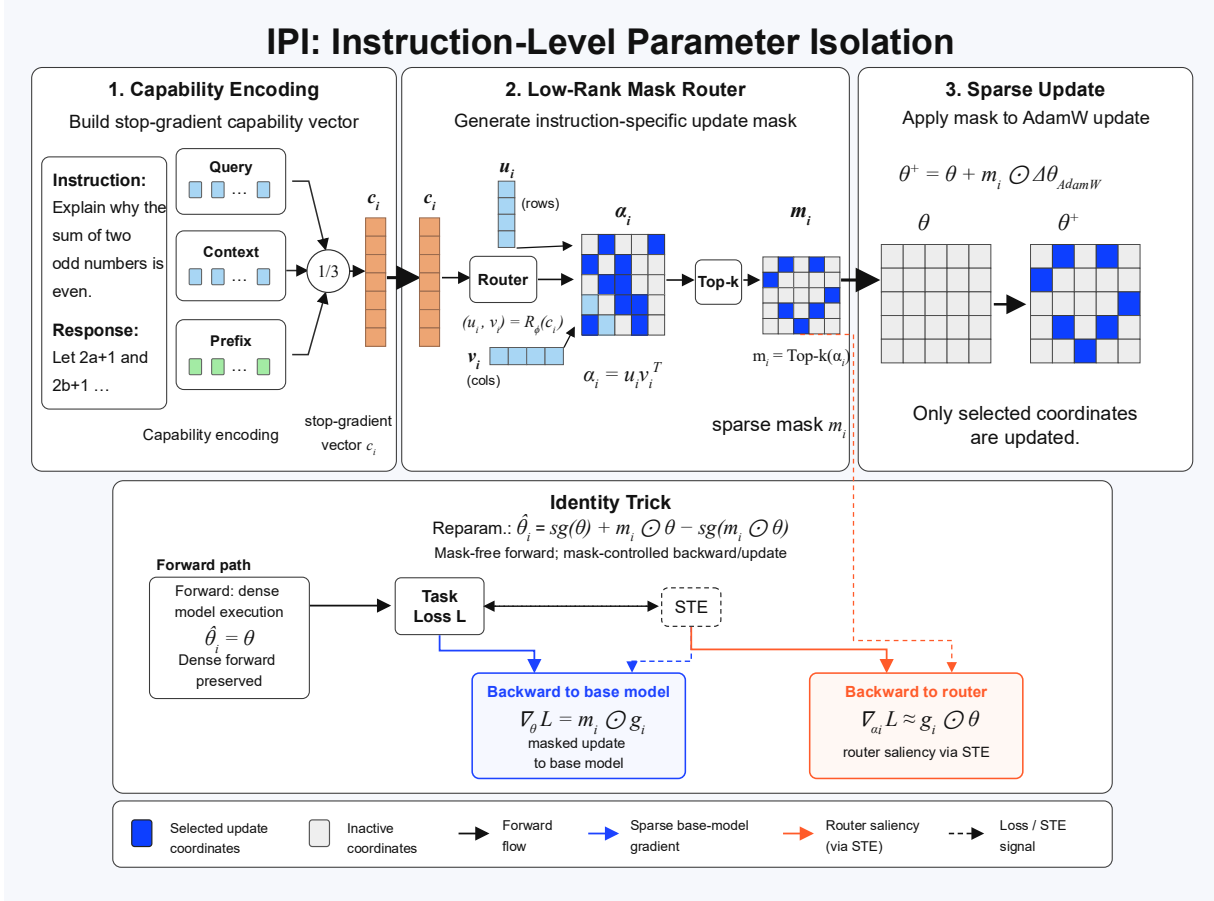


Figure 2: Overview of Instruction-Level Parameter Isolation (IPI). IPI builds an instruction-level capability representation, predicts a sparse update mask with a structured router, and uses an identity-trick reparameterization to keep the forward pass dense while routing backward updates to instruction-conditioned parameter subspaces.

parameter usage. IPI instead predicts an instance-conditioned update mask m_i with a fixed update budget $\|m_i\|_0 = k$:

$$\theta \leftarrow \theta - \eta (m_i \odot \nabla_{\theta} \mathcal{L}_i). \quad (1)$$

In mini-batch training, different instructions may activate different update supports. Therefore, IPI accumulates per-instance masked gradients rather than applying one shared mask to the whole batch.

2.2 Instruction-Level Capability Representation

For each instruction x_i , IPI constructs a dense capability vector $c_i \in \mathbb{R}^h$ from three stop-gradient views:

$$c_i = sg\left(\frac{p_{\text{query}} + p_{\text{input}} + p_{\text{prefix}}}{3}\right), \quad (2)$$

where $sg(\cdot)$ denotes stop-gradient. Here p_{query} summarizes the instruction query, p_{input} summarizes the optional input context and is set to zero

when absent, and p_{prefix} summarizes a short response prefix obtained from a teacher-forced forward pass during training. Each view is computed by averaging last-layer hidden states. Detaching these representations provides a stable routing signal without introducing higher-order interactions between the router and the backbone.

The response-prefix view is used only for training-time update routing. After fine-tuning, the router is discarded and inference uses the dense fine-tuned model alone. Thus teacher-forced prefixes do not alter inference-time inputs. For held-out routing analyses, we use generated prefixes or a prefix-ablated variant to avoid conditioning on gold responses.

2.3 Capability-Conditioned Low-Rank Router

A naive instruction-conditioned importance tensor over a weight matrix $W^{(\ell)} \in \mathbb{R}^{p_{\ell} \times q_{\ell}}$ would require one score for every trainable coordinate. This is impractical for large language models. IPI therefore

predicts importance scores through a structured row-column factorization.

For each routed layer ℓ , the capability vector is first mapped to a low-dimensional routing bottleneck:

$$\mathbf{z}_i^{(\ell)} = \sigma(\mathbf{A}^{(\ell)} \mathbf{c}_i), \quad \mathbf{A}^{(\ell)} \in \mathbb{R}^{r \times h}, \quad (3)$$

where $r \ll h$ is the routing rank. The router then predicts row and column factors:

$$\begin{aligned} \mathbf{u}_i^{(\ell)} &= \sigma(\mathbf{B}_u^{(\ell)} \mathbf{z}_i^{(\ell)}) \in (0, 1)^{p_\ell}, \\ \mathbf{v}_i^{(\ell)} &= \sigma(\mathbf{B}_v^{(\ell)} \mathbf{z}_i^{(\ell)}) \in (0, 1)^{q_\ell}. \end{aligned} \quad (4)$$

The implicit per-weight importance score is

$$\alpha_{i,(a,b)}^{(\ell)} = u_{i,a}^{(\ell)} v_{i,b}^{(\ell)}. \quad (5)$$

This factorization reflects a structural bias: an instruction may preferentially recruit certain output channels and input channels, and their interaction determines the importance of a specific weight. In practice, the full score matrix $\alpha_i^{(\ell)}$ is not materialized; Top- k selection is performed over the factorized scores, with complexity details in Appendix C.5. Section 4.5 ablates this structured parameterization against a coarser block-level router and a tied-factor variant, and shows that routing quality changes little beyond $r = 8$.

2.4 Sparse Update Selection and Identity Trick

For each instruction, IPI converts continuous importance scores into a hard sparse update mask. For layer ℓ , we select k_ℓ coordinate pairs from the implicit outer product:

$$\mathbf{m}_i^{(\ell)} = \text{TopK}(\alpha_i^{(\ell)}, k_\ell), \quad \sum_{\ell=1}^L k_\ell = k. \quad (6)$$

The allocation $k_\ell \propto p_\ell q_\ell$ keeps the update budget comparable across routed layers.

A naive post-hoc application of $\mathbf{m}_i$ to the gradient would make the forward loss independent of the mask, leaving the router without task-loss supervision. Directly masking forward weights is also undesirable under highly sparse update budgets, because it would remove most pretrained computation from the forward pass. IPI therefore separates the dense forward computation from the sparse update gate. Figure 3 illustrates this design: the forward computation remains dense, while the backward

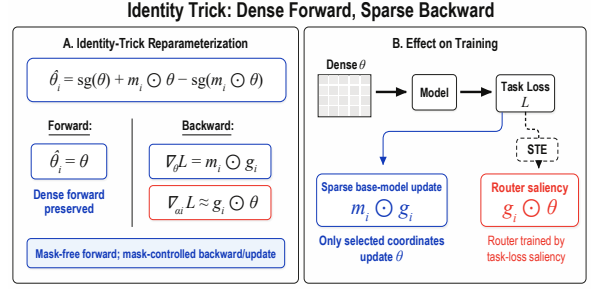


Figure 3: Identity-trick reparameterization: dense forward computation with sparse base-model gradients and router saliency gradients.

pass sends masked gradients to the base model and a task-loss saliency signal to the router. We first define a straight-through estimator (STE) mask for the hard Top- k selection (Bengio et al., 2013; Jang et al., 2016), $\tilde{\mathbf{m}}_i = \text{sg}(\mathbf{m}_i - \alpha_i) + \alpha_i$, so that numerically $\tilde{\mathbf{m}}_i = \mathbf{m}_i$ in the forward pass while gradients are passed to α_i in the backward pass.

For coordinate j , IPI defines the effective weight used by instruction i as

$$\begin{aligned} \hat{\theta}_{i,j} &= \text{sg}(\theta_j) + \text{sg}(m_{i,j})(\theta_j - \text{sg}(\theta_j)) \\ &\quad + \text{sg}(\theta_j)(\tilde{m}_{i,j} - \text{sg}(\tilde{m}_{i,j})). \end{aligned} \quad (7)$$

Writing $g_{i,j} = \partial \mathcal{L}_i(\theta) / \partial \theta_j$, this reparameterization induces three behaviors simultaneously:

$$\begin{aligned} \hat{\theta}_{i,j} &= \theta_j, \quad \frac{\partial \mathcal{L}_i}{\partial \theta_j} = m_{i,j} g_{i,j}, \\ \frac{\partial \mathcal{L}_i}{\partial \alpha_{i,j}} &\approx g_{i,j} \theta_j. \end{aligned} \quad (8)$$

Both residual terms in Eq. (7) are zero in value, so the forward pass is identical to ordinary dense SFT; the base model receives a gradient only on the coordinates selected for instruction i ; and the router receives a task-loss saliency signal. Under gradient descent, the last term promotes coordinates with large descent saliency $-g_{i,j} \theta_j$, which can be interpreted as a movement-style first-order importance signal conditioned on each instruction (Sanh et al., 2020). Therefore, the same loss simultaneously trains the base model with sparse updates and trains the router to select useful update coordinates. The full derivation is deferred to Appendix C.1.

In mini-batch training, routed linear layers accumulate per-instance masked gradients, $\nabla_{\theta} \mathcal{L}_{\text{base}} = B^{-1} \sum_{i \in B} \mathbf{m}_i \odot \mathbf{g}_i$, while the router receives the STE saliency signal through α_i ; the custom backward implementation is described in Appendix C.2.

Finally, masking gradients alone is insufficient under AdamW, because decoupled weight decay can still modify coordinates with zero gradient. We therefore also gate the final optimizer update with the batch-level update gate $\bar{m}_{\mathcal{B},j} = \mathbb{I}(\sum_{i \in \mathcal{B}} m_{i,j} > 0)$:

$$\theta_{t+1} = \theta_t + \bar{\mathbf{m}}_{\mathcal{B}} \odot \text{AdamWStep}(\theta_t, \nabla_{\theta} \mathcal{L}_{\text{total}}). \quad (9)$$

Thus unselected coordinates receive neither gradient updates nor decoupled weight decay.

2.5 Training Objective and Load Balancing

The primary objective is the standard cross-entropy loss evaluated on the identity-trick weights, $\mathcal{L}_{\text{CE}} = B^{-1} \sum_{i=1}^B \mathcal{L}_i(\hat{\theta}_i)$. Forward-wise, this is equivalent to ordinary dense SFT; the dependence on mask scores appears through the backward paths defined above.

Without regularization, the router may collapse to a shared sparse subnetwork for all instructions. To discourage this, we apply a factor-level load-balancing regularizer. For each routed layer, we normalize the predicted row and column factors to unit mass and average them over the mini-batch, yielding $\bar{\mathbf{u}}^{(\ell)}$ and $\bar{\mathbf{v}}^{(\ell)}$ (Appendix C.5). The load-balancing loss maximizes their entropy, and the total objective is

$$\mathcal{L}_{\text{lb}} = - \sum_{\ell=1}^L \left[H(\bar{\mathbf{u}}^{(\ell)}) + H(\bar{\mathbf{v}}^{(\ell)}) \right], \quad (10)$$

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{CE}} + \lambda_{\text{lb}} \mathcal{L}_{\text{lb}}.$$

This regularizer discourages all instructions in a batch from concentrating on the same rows and columns, while still allowing each individual instruction to specialize. We linearly anneal λ_{lb} to zero during training.

3 Experimental Setup

We evaluate IPI in a sequential heterogeneous SFT setting, where instructions from multiple capability domains are fine-tuned in a multi-stage training stream. This setting tests whether instruction-level routing can reduce interference and forgetting under distribution shifts across stages, without relying on coarse task-level masks. Full dataset details, judging protocols, baseline definitions, normalization rules, and hyperparameters are provided in Appendix B.

Datasets and metrics. We use five benchmarks covering complementary capabilities:

GSM8K (Cobbe et al., 2021) for mathematical reasoning, LogiQA (Liu et al., 2020) for logical reasoning, CodeAlpaca (Chaudhary, 2023) for code generation, and Alpaca (Taori et al., 2023) and UltraChat (Ding et al., 2023) for instruction following and dialogue. GSM8K and LogiQA are evaluated by accuracy, CodeAlpaca by CodeBLEU (Ren et al., 2020), and Alpaca/UltraChat by GPT-4 (OpenAI et al., 2024) scoring on a 1–10 scale. We also report Avg. Norm. Score as a compact cross-capability summary.

Baselines. We compare IPI with Full SFT, Random Sparse Fine-Tuning, Heuristic Sparse Fine-Tuning, Static Isolation, and Evolving Parameter Isolation (EPI) (Lin et al., 2026). These baselines separate the effects of full shared optimization, generic sparse updating, fixed task-level masks, and global dynamic isolation. The key comparison is between task- or stage-level isolation and instruction-conditioned routing: EPI maintains a global evolving mask over the training trajectory, whereas IPI predicts a distinct update support for each instruction.

Implementation. We evaluate LLaMA-3.1-8B (Grattafiori et al., 2024), Mistral-7B (Jiang et al., 2023), Qwen3-8B (Yang et al., 2025), and Gemma-3-12B (Team et al., 2025). All methods are trained with AdamW for three epochs under the same task order, data budget, and optimization budget. IPI uses a per-instruction update sparsity of 1%, routing rank $r = 8$, and load-balancing coefficient $\lambda_{\text{lb}} = 0.01$, which is linearly annealed to zero. All experiments use three random seeds.

4 Experimental Results and Analysis

Table 1 reports the main results across four model families and five heterogeneous instruction tuning benchmarks. The comparison is designed to answer three questions: whether instruction-conditioned sparse updates improve over full shared SFT, whether learned routing is more effective than generic sparse fine-tuning, and whether instruction-level masks provide finer control than static or global dynamic isolation.

4.1 Overall Performance Comparison

IPI consistently outperforms all baselines across backbone models and evaluation benchmarks. Compared with Full SFT, IPI improves the Avg. Norm. Score by +0.96 on LLaMA-3.1-8B, +0.86 on Mistral-7B, +0.95 on Qwen3-8B, and +0.95

Base Model	Method	GSM8K	CodeAlpaca	LogiQA	Alpaca	UltraChat	Avg. Norm.
LLaMA-3.1-8B	Full SFT	57.6	29.1	61.4	8.12	8.28	7.60
	Random Sparse FT	58.7	28.8	62.0	8.18	8.36	7.70
	Heuristic Sparse FT	61.3	30.8	62.5	8.06	8.25	7.91
	Static Isolation	60.8	30.3	63.7	8.31	8.55	8.00
	EPI	63.2	32.1	66.1	8.58	8.86	8.32
	IPI (Ours)	64.7	33.0	67.6	8.81	9.08	8.56
Mistral-7B	Full SFT	53.0	26.8	58.7	7.54	7.88	7.04
	Random Sparse FT	54.2	26.4	59.4	7.66	8.00	7.16
	Heuristic Sparse FT	57.1	26.9	61.1	7.43	7.82	7.39
	Static Isolation	56.6	28.0	61.3	7.79	8.13	7.43
	EPI	59.0	29.6	63.0	8.17	8.42	7.73
	IPI (Ours)	60.4	30.6	64.4	8.39	8.64	7.90
Qwen3-8B	Full SFT	59.2	30.0	62.9	8.32	8.47	7.83
	Random Sparse FT	60.0	29.8	63.5	8.43	8.60	7.94
	Heuristic Sparse FT	61.4	32.0	66.0	8.24	8.45	8.09
	Static Isolation	62.3	31.5	65.5	8.58	8.87	8.20
	EPI	64.7	33.0	67.8	8.88	9.18	8.55
	IPI (Ours)	66.1	34.2	69.4	9.07	9.38	8.78
Gemma-3-12B	Full SFT	61.0	31.4	65.1	8.68	8.92	8.15
	Random Sparse FT	62.1	31.0	65.7	8.79	9.02	8.26
	Heuristic Sparse FT	65.0	33.1	67.3	8.61	8.86	8.46
	Static Isolation	64.5	32.7	67.6	8.92	9.19	8.54
	EPI	66.9	34.3	70.1	9.18	9.47	8.87
	IPI (Ours)	68.3	35.5	71.6	9.36	9.66	9.10

Table 1: Main results across heterogeneous instruction tuning benchmarks. All values are averaged over three random seeds; standard deviations are reported in Appendix B.4. Background colors denote the top-3 methods within each model and metric, with darker green indicating higher performance.

on Gemma-3-12B. These gains are not confined to a single benchmark: IPI improves mathematical reasoning, code generation, logical reasoning, instruction following, and dialogue quality. For example, on LLaMA-3.1-8B, IPI increases GSM8K from 57.6 to 64.7, CodeAlpaca from 29.1 to 33.0, and LogiQA from 61.4 to 67.6. This indicates that instruction-conditioned sparse updates can improve specialization while preserving general instruction-following ability.

Generic sparse fine-tuning alone is not sufficient. Random Sparse FT yields only small improvements over Full SFT, showing that sparsity by itself does not explain the gains. Heuristic Sparse FT is more competitive on reasoning- and code-oriented metrics, entering the top three for GSM8K or CodeAlpaca on every backbone, but it is less stable on open-ended instruction-following and dialogue benchmarks and still trails Static Isolation, EPI, and IPI in Avg. Norm. Score. Heuristic sparse selection therefore helps when useful update directions are relatively concentrated, but it cannot reliably adapt to heterogeneous instruction-level

capability demands.

4.2 Comparison with Static and Global Dynamic Isolation

Static Isolation improves over Full SFT by reducing part of the cross-capability contention, especially on open-ended instruction-following and dialogue benchmarks. However, its fixed-mask assumption limits its flexibility: a static support cannot separate conflicting examples within the same task or share update regions between compatible examples from different tasks. Consistent with this, Heuristic Sparse FT surpasses Static Isolation on several reasoning or code metrics, while Static Isolation remains stronger on average and more consistent on Alpaca and UltraChat.

EPI further improves performance by updating a global mask over the training trajectory, which captures temporal changes in parameter importance. However, EPI still applies a shared mask at each stage and therefore cannot adapt to instance-level heterogeneity within a mini-batch. IPI provides an additional level of granularity by conditioning the

update mask on each instruction. Across model families, IPI improves over EPI by +0.17 to +0.24 Avg. Norm. points. This gap is smaller than the gain over Full SFT but is more diagnostic: it isolates the benefit of instruction-conditioned routing beyond global dynamic importance estimation. The consistent improvement over EPI suggests that parameter importance is not only temporally evolving, but also instance-dependent.

4.3 Robustness Across Model Families and Capabilities

The gains are consistent across LLaMA, Mistral, Qwen, and Gemma backbones, so the method is not tied to a specific architecture. They are largest on reasoning-heavy and code-oriented benchmarks, where heterogeneous instructions are most likely to induce conflicting update directions, while Alpaca and UltraChat also improve, indicating that sparse update routing does not degrade open-ended generation. Overall, instruction-level routing provides a more faithful unit of parameter isolation than task labels, heuristic sparsity rules, or globally shared masks.

4.4 Training Regime, Update Budget, and Cost

Mixed-batch SFT. Standard SFT pipelines often mix all capability domains within each mini-batch instead of training in fixed stages. Because IPI predicts a mask independently for each instruction, it applies directly to this setting without modification. Retraining with uniformly mixed batches, Full SFT improves ($7.60 \rightarrow 7.85$ on LLaMA-3.1-8B and $7.83 \rightarrow 8.06$ on Qwen3-8B), because removing the task ordering also removes stage-induced forgetting. EPI loses most of its advantage ($8.32 \rightarrow 8.18$, $8.55 \rightarrow 8.41$), since its single global evolving mask relies on the stage structure that mixing eliminates and cannot specialize within a heterogeneous batch. IPI is essentially unchanged ($8.56 \rightarrow 8.54$, $8.78 \rightarrow 8.75$) and remains the strongest method on both backbones, confirming that it targets instruction-level rather than stage-level interference. Full results are given in Appendix E.3.

Matched update budgets. Table 2 asks whether the gains simply follow from updating a favorable number of parameters. IPI leads Random Sparse FT, Static Isolation, and EPI at every matched budget, so the improvement comes from routing

Budget k/d	Random Sparse	Static Isolation	EPI	IPI
0.5%	7.55	7.84	8.20	8.45
1.0%	7.70	8.00	8.32	8.56
2.0%	7.69	8.03	8.36	8.53
5.0%	7.64	7.96	8.28	8.42
10.0%	7.60	7.89	8.19	8.33

Table 2: Matched-budget sparsity sensitivity on LLaMA-3.1-8B (Avg. Norm. Score). IPI leads at every budget and peaks around the default 1% per-instruction update budget.

Method	Throughput	Memory	Router Params	Inference Cost
Full SFT	$1.00\times$	$1.00\times$	—	$1.00\times$
Static Isolation	$0.91\times$	$1.05\times$	—	$1.00\times$
EPI	$0.88\times$	$1.07\times$	—	$1.00\times$
IPI (Ours)	$0.86\times$	$1.09\times$	$\approx 30M$	$1.00\times$

Table 3: Training efficiency on LLaMA-3.1-8B, measured at batch size 32. Throughput and memory are normalized to Full SFT. The router is discarded after fine-tuning, so all methods share the same inference cost.

selectivity rather than update capacity. Performance peaks around the default 1%: lower budgets provide insufficient plasticity, whereas larger budgets increase support overlap and therefore reintroduce interference. The near-saturation between 1% and 2% indicates that the default is a controlled stability–plasticity trade-off rather than an arbitrary heuristic.

Training cost. IPI adds three sources of training-time overhead: building capability vectors, computing low-rank router factors, and factorized Top- k selection over outer-product scores. As shown in Table 3, this reduces throughput by 14% and increases peak memory by less than 10%, since IPI stores row–column factors and sparse coordinate indices rather than dense per-weight score matrices. The router is lightweight ($\approx 30M$ parameters at $r = 8$, roughly 0.4% of a 7B/8B backbone) and is discarded after fine-tuning, so the deployed model is a standard dense LLM with unchanged inference cost. The relative overhead also does not grow with load: routing costs $O(rh + r(p_\ell + q_\ell))$ per instance and layer against a dense linear cost of $O(p_\ell q_\ell)$, and the measured IPI-over-Full-SFT training-time ratio is $1.18\times$, $1.16\times$, and $1.15\times$ at batch sizes 16, 32, and 64, decreasing slightly as the fixed routing cost is amortized over more instances.

Method	LLaMA-3.1-8B	Qwen3-8B	Avg. Drop
IPI (full)	8.56	8.78	–
<i>Update-routing mechanism</i>			
w/o Identity Trick	7.66	7.86	–0.91
Frozen Router	7.69	7.91	–0.87
Dense Soft Masking	8.12	8.37	–0.43
w/o Load Balancing	8.31	8.54	–0.25
<i>Capability representation</i>			
Query Only	8.42	8.64	–0.14
Query + Prefix (Gold)	8.45	8.67	–0.11
Query + Context	8.47	8.69	–0.09
Query + Context + Prefix (Gen.)	8.50	8.72	–0.06
<i>Conditioning unit</i>			
Task-Conditioned Router	8.35	8.59	–0.20

Table 4: Core ablation results (Avg. Norm. Score). Avg. Drop denotes the mean decrease relative to full IPI across the two backbones. “w/o Identity Trick” replaces the gradient bridge with post-hoc gradient masking, “Frozen Router” keeps the randomly initialized router fixed, and “Dense Soft Masking” replaces hard Top- k gates with continuous update weights. The capability-representation block ablates the three views of Eq. (2): full IPI corresponds to Query + Context + Prefix with a teacher-forced gold prefix, while “Prefix (Gen.)” samples the prefix from the current model instead.

4.5 Ablation Study

We conduct controlled ablations to isolate the core design choices of IPI. We report Avg. Norm. Score on two representative backbones, LLaMA-3.1-8B and Qwen3-8B. All variants use the same sequential heterogeneous training stream, optimizer, sparsity budget, and training schedule as the main experiments; additional diagnostics are provided in Appendix F.

Update-routing mechanism. As shown in Table 4, the identity-trick reparameterization is the most critical component: removing it reduces the two-model average by 0.91 points. This confirms that post-hoc gradient masking is insufficient because it leaves the router without direct task-loss supervision. The frozen-router variant shows a similarly large drop, indicating that sparse update supports must be learned rather than produced by random or untrained routing. Both variants perform close to or below generic sparse fine-tuning, further supporting that IPI’s gains come from task-loss-driven instruction-level routing rather than sparsity alone. Dense soft masking also underperforms, suggesting that hard sparse update gates are important for isolating parameter updates. Removing load balancing degrades performance as well, showing that the router can otherwise collapse to a shared sparse subnetwork; Appendix F.1 confirms that this

Router Variant	Router Params	Rel. Throughput	Avg. Norm.
Block-Level Router	0.12%	1.05×	8.36
Tied Row/Column Factors	0.40%	1.00×	8.43
Outer Product, $r = 4$	0.20%	1.02×	8.48
Outer Product, $r = 8$	0.40%	1.00×	8.56
Outer Product, $r = 16$	0.79%	0.98×	8.59
Outer Product, $r = 32$	1.58%	0.94×	8.60

Table 5: Router architecture and capacity ablation on LLaMA-3.1-8B. Router parameters are reported as a percentage of the backbone parameter count, and throughput is normalized to the default $r = 8$ outer-product router used in Table 3.

drop coincides with a sharp increase in mask overlap and a loss of support coverage.

Capability representation and conditioning unit.

The middle block of Table 4 decomposes the multi-view capability vector. Query-only routing is the weakest variant, and adding either the input context or the response prefix recovers part of the gap, so both context and response-structure cues carry information about the capability an instruction demands; the two views are also complementary, since the full three-view representation is best. Replacing the teacher-forced gold prefix with a short prefix sampled from the current model retains most of the gain (8.50 versus 8.56 on LLaMA-3.1-8B), indicating that the prefix view supplies a signal about expected response structure rather than simply exposing the target answer; since the router is discarded after fine-tuning, no prefix is required at inference. Finally, replacing the instruction-conditioned router with a task-conditioned router causes a clear drop, confirming that task identity is too coarse a conditioning unit for heterogeneous instruction tuning.

Router architecture and capacity. Table 5 ablates the structured row-column router itself, so that its design is not left as an unverified heuristic. A coarser block-level router is cheaper but clearly weaker, indicating that block-granular routing cannot capture sufficiently fine-grained parameter usage, and tying the row and column factors also degrades performance, so asymmetric input/output-channel recruitment is useful. Increasing the bottleneck rank from $r = 4$ to $r = 16$ improves performance, but $r = 32$ adds only 0.01 points while reducing throughput by a further 4%. We therefore adopt $r = 8$ by default as a controlled efficiency–performance trade-off, with $r = 16$ available as a

slightly stronger but more expensive variant.

5 Further Analysis of Instruction-Level Parameter Isolation

We provide two analyses to examine why IPI improves heterogeneous SFT: whether instruction-conditioned routing reduces destructive update interactions, and whether surface task labels are reliable proxies for optimization similarity. Unless otherwise specified, analyses use LLaMA-3.1-8B during the early training phase; additional diagnostics are provided in Appendix E.

5.1 Gradient Interference Reduction

We measure the cosine similarity of effective parameter updates during the first 500 training steps. For IPI, the measured vector is the masked per-instruction update $\mathbf{m}_i \odot \mathbf{g}_i$; for other isolation methods, we apply their corresponding update masks before computing similarity. We group sampled instruction pairs into capability-aligned and capability-conflicting pairs according to benchmark capability categories. Table 6 reports the average cosine similarity after each method’s update rule.

As shown in Table 6, Full SFT exhibits negative similarity for capability-conflicting pairs, indicating destructive interactions in the shared parameter space. Static Isolation reduces this conflict, and EPI further weakens it through a global evolving mask. IPI achieves the strongest alignment for compatible pairs and shifts conflicting pairs away from negative interaction, suggesting that instruction-conditioned supports route updates into more compatible sparse subspaces.

Method	Aligned Pairs	Conflicting Pairs
Full SFT	0.018	−0.034
Static Isolation	0.026	−0.011
EPI	0.034	−0.004
IPI (Ours)	0.052	0.018

Table 6: Average cosine similarity of effective updates. Higher values indicate more constructive update alignment.

5.2 Instruction-Level vs. Task-Level Interference Patterns

We next test whether task identity reliably reflects optimization similarity. We compute pairwise cosine similarity of full-parameter gradients for 2,000 sampled training instructions and group pairs by

Grouping	Mean $\uparrow$	Std.	Neg.	High
Within GSM8K	0.41	0.27	10.8%	34.6%
Within CodeAlpaca	0.38	0.31	14.7%	31.2%
Within UltraChat	0.36	0.29	13.5%	29.4%
Cross-Task	0.05	0.18	38.9%	6.7%

Table 7: Gradient similarity by surface-task grouping. “Neg.” denotes negative-cosine pairs; “High” denotes pairs with cosine similarity above 0.5.

surface task labels. Table 7 summarizes the resulting similarity distribution.

Table 7 shows that within-task pairs have higher mean similarity than random cross-task pairs, but their large variance and non-negligible negative-pair rates reveal substantial intra-task heterogeneity. Conversely, some cross-task pairs are strongly aligned, indicating that compatible update directions can appear across task boundaries. These results support the central motivation of IPI: instruction instances, rather than coarse task labels, provide a more faithful unit for routing sparse parameter updates. Together with the matched-budget comparison in Section 4.4, they indicate that the gains come from learned routing selectivity rather than from simply updating fewer parameters.

6 Conclusion

We argued that the unit of parameter isolation should match the unit of optimization in supervised instruction fine-tuning: the individual instruction-response pair rather than the surface task label. We proposed **Instruction-Level Parameter Isolation (IPI)**, which predicts instruction-conditioned sparse update masks from multi-view capability representations and, through an identity-trick reparameterization, preserves dense pretrained forward computation while learning routing decisions from task-loss saliency and applying sparse AdamW-compatible updates. Across diverse instruction tuning benchmarks and multiple backbone models, IPI consistently improves over full SFT, generic sparse fine-tuning, static task-level isolation, and global dynamic isolation, and further analyses show that instruction-level masks reduce destructive gradient interactions and reveal optimization structure that task labels miss. Since the router is discarded after fine-tuning, IPI is a practical training-time mechanism for interference-aware adaptation that leaves a standard dense model for inference.

7 Limitations

IPI has several limitations. First, its capability representation and sparsity budget are fixed rather than adaptive. Although the query–context–prefix representation and the 1% budget are validated by controlled ablations, learned view aggregation or per-instruction budget allocation may further improve routing quality. Second, IPI introduces training-time overhead from per-instance routing, factorized Top- k selection, and custom backward accumulation of per-instance masked gradients. The router is discarded at inference, so IPI does not directly reduce inference latency unless combined with compression or distillation. Finally, our analysis of the identity trick is an exact structural characterization rather than a convergence theorem; while it links mask overlap to gradient interference, a tighter characterization of joint router–backbone optimization remains open.

References

- Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.
- Sahil Chaudhary. 2023. Code alpaca: An instruction-following llama model for code generation. <https://github.com/sahil280114/codealpaca>.
- Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. 2018. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International conference on machine learning*, pages 794–803. PMLR.
- Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, and 16 others. 2022. *Scaling instruction-finetuned language models*. *Preprint*, arXiv:2210.11416.
- 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. 2021. *Training verifiers to solve math word problems*. *Preprint*, arXiv:2110.14168.
- Chang Dai, Hongyu Shan, Mingyang Song, and Di Liang. 2025. Hope: Hyperbolic rotary positional encoding for stable long-range dependency modeling in large language models. *arXiv preprint arXiv:2509.05218*.
- Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. 2023. *Enhancing chat language models by scaling high-quality instructional conversations*. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3029–3051, Singapore. Association for Computational Linguistics.
- Guanting Dong, Hongyi Yuan, Keming Lu, Chengpeng Li, Mingfeng Xue, Dayiheng Liu, Wei Wang, Zheng Yuan, Chang Zhou, and Jingren Zhou. 2024. *How abilities in large language models are affected by supervised fine-tuning data composition*. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 177–198, Bangkok, Thailand. Association for Computational Linguistics.
- William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res.*, 23(1).
- Zichu Fei, Qi Zhang, Tao Gui, Di Liang, Sirui Wang, Wei Wu, and Xuan-Jing Huang. 2022. Cqg: A simple and effective controlled generation framework for multi-hop question generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6896–6906.
- Ziyuan Gao, Di Liang, Xianjie Wu, Philippe Morel, and Minlong Peng. 2026. Decorl: Decoupling reasoning chains via parallel sub-step generation and cascaded reinforcement for interpretable and scalable rlhf. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 30789–30797.
- 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, and 542 others. 2024. *The llama 3 herd of models*. *Preprint*, arXiv:2407.21783.
- Tao Gui, Qi Zhang, Jingjing Gong, Minlong Peng, Di Liang, Keyu Ding, and Xuanjing Huang. 2018. *Transferring from formal newswire domain with hypernet for Twitter POS tagging*. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2540–2549, Brussels, Belgium. Association for Computational Linguistics.
- Xiaosong Han, Ke Chen, Xindi Dai, Di Liang, Minlong Peng, Wei Pang, Fausto Giunchiglia, Xiaoyue Feng, Yonghao Liu, and Renchu Guan. 2026. Trace: Discovering task-specific parameter via adaptation-aware probing for continual fine-tuning. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pages 1530–1541.

- Mengze Hong, Xia Zeng, Zeyang Lei, Sheng Wang, Chen Jason Zhang, Di Jiang, Taiming Fu, Jinfeng Huang, Mengqiao Liu, Qinghe Chang, Haosheng Zou, Qiongyi Zhou, Sijun He, Simonjmdeng, Haojing Huang, Zijian Li, Lucas Mu Li, Fubao Zhang, Mona Zhou, and 10 others. 2026. [Uxbench: Benchmarking user experience in ai assistants](#). *Preprint*, arXiv:2606.09570.
- Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR.
- Zhanghao Hu, Qinglin Zhu, Runcong Zhao, Di Liang, Hanqi Yan, Yulan He, and Lin Gui. 2026. Beyond rag for agent memory: Retrieval by decoupling and aggregation. *arXiv preprint arXiv:2602.02007*.
- Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.
- Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, L  lio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timoth  e Lacroix, and William El Sayed. 2023. [Mistral 7b](#). *Preprint*, arXiv:2310.06825.
- James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. 2017. [Overcoming catastrophic forgetting in neural networks](#). *Proceedings of the National Academy of Sciences*, 114(13):3521–3526.
- Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich K  ttler, Mike Lewis, Wen-tau Yih, Tim Rock  tschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems, NIPS ’20*, Red Hook, NY, USA. Curran Associates Inc.
- Bo Li, Di Liang, and Zixin Zhang. 2024. Comate-former: Combined attention transformer for semantic sentence matching. In *ECAI 2024: 27th European Conference on Artificial Intelligence, 19–24 October 2024, Santiago de Compostela, Spain–Including 13th Conference on Prestigious Applications of Intelligent Systems (PAIS 2024)*, pages 3677–3684. SAGE Publications Pvt. Ltd 1 Oliver’s Yard, 55 City Road, London, EC1Y 1SP.
- Junchen Li, Liang Xu, Qizhi Chen, Rongzheng Wang, Chao Qi, Shihao He, Di Liang, Haibo Shi, and Shuang Liang. 2026. [When safety becomes a vulnerability: Exploiting llm alignment homogeneity for transferable blocking in rag](#). *Preprint*, arXiv:2603.03919.
- Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. 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)*, pages 4582–4597.
- Zhizhong Li and Derek Hoiem. 2017. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947.
- Di Liang, Fubao Zhang, Qi Zhang, and Xuan-Jing Huang. 2019a. Asynchronous deep interaction network for natural language inference. 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)*, pages 2692–2700.
- Di Liang, Fubao Zhang, Weidong Zhang, Qi Zhang, Jinlan Fu, Minlong Peng, Tao Gui, and Xuanjing Huang. 2019b. Adaptive multi-attention network incorporating answer information for duplicate question detection. In *Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval*, pages 95–104.
- Zekai Lin, Chao Xue, Di Liang, Xingsheng Han, Peiyang Liu, Xianjie Wu, Lei Jiang, Yu Lu, Bob Simons, Shuang Liang, and Minlong Peng. 2026. [Parameter importance is not static: Evolving parameter isolation for supervised fine-tuning](#). In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32846–32864, San Diego, California, United States. Association for Computational Linguistics.
- Zekai Lin and Xu Zheng. 2026. Panoenv: Exploring 3d spatial intelligence in panoramic environments with reinforcement learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9647–9657.
- Jian Liu, Leyang Cui, Hanmeng Liu, Dandan Huang, Yile Wang, and Yue Zhang. 2020. Logiqa: A challenge dataset for machine reading comprehension with logical reasoning. *arXiv preprint arXiv:2007.08124*.
- Peiyang Liu, Zhirui Chen, Xi Wang, Di Liang, Youru Li, Zhi Cai, and Wei Ye. 2026a. Learning from contrasts: Synthesizing reasoning paths from diverse search trajectories. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10947–10969.
- Peiyang Liu, Ziqiang Cui, Di Liang, and Wei Ye. 2025a. Who stole your data? a method for detecting unauthorized rag theft. *arXiv preprint arXiv:2510.07728*.

- Peiyang Liu, Ziqiang Cui, Xi Wang, Di Liang, and Wei Ye. 2026b. [Chain of evidence: Pixel-level visual attribution for iterative retrieval-augmented generation](#). In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '26, page 1141–1151, New York, NY, USA. Association for Computing Machinery.
- Peiyang Liu, Qiang Yan, Ziqiang Cui, Di Liang, Xi Wang, and Wei Ye. 2026c. Beyond semantic relevance: Counterfactual risk minimization for robust retrieval-augmented generation. *arXiv preprint arXiv:2605.01302*.
- Xiaoyu Liu, Xiaoyu Guan, Di Liang, and Xianjie Wu. 2026d. Dpi: Exploiting parameter heterogeneity for interference-free fine-tuning. In *ICASSP 2026-2026 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 19337–19341. IEEE.
- Xiaoyu Liu, Di Liang, Hongyu Shan, Peiyang Liu, Yonghao Liu, Muling Wu, Yuntao Li, Xianjie Wu, Li Miao, Jiangrong Shen, and Minlong Peng. 2025b. [Structural reward model: Enhancing interpretability, efficiency, and scalability in reward modeling](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 672–685, Suzhou (China). Association for Computational Linguistics.
- Yonghao Liu, Mengyu Li, Di Liang, Ximing Li, Fausto Giunchiglia, Lan Huang, Xiaoyue Feng, and Renchu Guan. 2024. Resolving word vagueness with scenario-guided adapter for natural language inference. *arXiv preprint arXiv:2405.12434*.
- David López-Paz and Marc’Aurelio Ranzato. 2017. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.
- Arun Mallya and Svetlana Lazebnik. 2018. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7765–7773.
- Michael McCloskey and Neal J Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier.
- Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. Large language models: A survey. *arXiv preprint arXiv:2402.06196*.
- OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2024. [Gpt-4 technical report](#). *Preprint*, arXiv:2303.08774.
- Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. Adapterfusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th conference of the European chapter of the association for computational linguistics: main volume*, pages 487–503.
- Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. 2020. Codebleu: a method for automatic evaluation of code synthesis. *arXiv preprint arXiv:2009.10297*.
- Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. 2021. Scaling vision with sparse mixture of experts. *Advances in Neural Information Processing Systems*, 34:8583–8595.
- David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. 2019. Experience replay for continual learning. *Advances in neural information processing systems*, 32.
- Victor Sanh, Thomas Wolf, and Alexander Rush. 2020. Movement pruning: Adaptive sparsity by fine-tuning. *Advances in neural information processing systems*, 33:20378–20389.
- Hongyu Shan, Mingyang Song, Chang Dai, Di Liang, and Han Chen. 2025. R-capsule: Compressing high-level plans for efficient large language model reasoning. *arXiv preprint arXiv:2509.22131*.
- Jian Song, Di Liang, Rumei Li, Yuntao Li, Sirui Wang, Minlong Peng, Wei Wu, and Yongxin Yu. 2022. Improving semantic matching through dependency-enhanced pre-trained model with adaptive fusion. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 45–57.
- Amelia Sorrenti, Giovanni Bellitto, Federica Proietto Salanitri, Matteo Pennisi, Concetto Spampinato, and Simone Palazzo. 2023. Selective freezing for efficient continual learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, pages 3550–3559.
- Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca.

- 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, and 197 others. 2025. [Gemma 3 technical report](#). Preprint, arXiv:2503.19786.
- Sirui Wang, Di Liang, Jian Song, Yuntao Li, and Wei Wu. 2022. Dabert: Dual attention enhanced bert for semantic matching. In *Proceedings of the 29th international conference on computational linguistics*, pages 1645–1654.
- Yao Wang, Di Liang, and Minlong Peng. 2025. [Not all parameters are created equal: Smart isolation boosts fine-tuning performance](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 9871–9885, Suzhou, China. Association for Computational Linguistics.
- Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*.
- Muling Wu, Qi Qian, Wenhao Liu, Xiaohua Wang, Zisu Huang, Di Liang, LI Miao, Shihan Dou, Changze Lv, Zhenghua Wang, Zhibo Xu, Lina Chen, Tianlong Li, Xiaoqing Zheng, and Xuanjing Huang. 2025a. [Progressive mastery: Customized curriculum learning with guided prompting for mathematical reasoning](#). Preprint, arXiv:2506.04065.
- Xianjie Wu, Di Liang, Jian Yang, Xianfu Cheng, LinZheng Chai, Tongliang Li, Liqun Yang, and Zhoujun Li. 2025b. Breaking size barrier: Enhancing reasoning for large-size table question answering. In *International Conference on Database Systems for Advanced Applications*, pages 241–256. Springer.
- Xianjie Wu, Xiaohang Xu, Tingyu Jiang, Jian Yang, Di Liang, Xianfu Cheng, Zhenhe Wu, Linzheng Chai, Wei Zhang, Jiaheng Liu, Ge Zhang, Bob Simons, Tongliang Li, and Zhoujun Li. 2026. [Mmtablebench: A multi-level multimodal benchmark for reasoning and layout complexity in table qa](#). In *Proceedings of the ACM Web Conference 2026*, WWW ’26, page 3881–3892, New York, NY, USA. Association for Computing Machinery.
- Xianjie Wu, Jian Yang, Linzheng Chai, Ge Zhang, Jiaheng Liu, Xeron Du, Di Liang, Daixin Shu, Xianfu Cheng, Tianzhen Sun, Tongliang Li, Zhoujun Li, and Guanglin Niu. 2025c. [Tablebench: a comprehensive and complex benchmark for table question answering](#). In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI’25/IAAI’25/EAAI’25. AAAI Press.
- Xianjie Wu, Jian Yang, Tongliang Li, Shiwei Zhang, Yiyang Du, LinZheng Chai, Di Liang, and Zhoujun Li. 2025d. [Unleashing potential of evidence in knowledge-intensive dialogue generation](#). In *ICASSP 2025 - 2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5.
- Chao Xue, Di Liang, Pengfei Wang, and Jing Zhang. 2024. Question calibration and multi-hop modeling for temporal question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19332–19340.
- Chao Xue, Di Liang, Sirui Wang, Jing Zhang, and Wei Wu. 2023. Dual path modeling for semantic matching by perceiving subtle conflicts. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE.
- Chao Xue, Yao Wang, Mengqiao Liu, Di Liang, Xingsheng Han, Peiyang Liu, Xianjie Wu, Chenyao Lu, Lei Jiang, Yu Lu, Haibo Shi, Shuang Liang, Minlong Peng, and Flora D. Salim. 2026a. [Reason only when needed: Efficient generative reward modeling via model-internal uncertainty](#). In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 23302–23319, San Diego, California, United States. Association for Computational Linguistics.
- Chao Xue, Yao Wang, Mengqiao Liu, Di Liang, Xingsheng Han, Peiyang Liu, Xianjie Wu, Chenyao Lu, Lei Jiang, Yu Lu, Haibo Shi, Shuang Liang, Minlong Peng, and Flora D. Salim. 2026b. [Why supervised fine-tuning fails to learn: A systematic study of incomplete learning in large language models](#). In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 30186–30213, San Diego, California, United States. Association for Computational Linguistics.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. [Qwen3 technical report](#). Preprint, arXiv:2505.09388.
- Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient surgery for multi-task learning. *Advances in neural information processing systems*, 33:5824–5836.
- Rui Zheng, Bao Rong, Yuhao Zhou, Di Liang, Sirui Wang, Wei Wu, Tao Gui, Qi Zhang, and Xuan-Jing Huang. 2022. Robust lottery tickets for pre-trained language models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2211–2224.

A Related Work

A.1 Heterogeneous Instruction Tuning and Interference

Supervised fine-tuning (SFT) has become a standard paradigm for adapting pretrained language models to downstream instructions and user-facing tasks (Wei et al., 2021; Chung et al., 2022; Minaee et al., 2024). In practice, SFT is one stage of a broader post-training pipeline that also involves reward modeling and preference optimization (Liu et al., 2025b; Gao et al., 2026; Xue et al., 2026a) as well as curriculum design over the training stream (Wu et al., 2025a). Modern instruction tuning datasets increasingly mix heterogeneous capabilities, including mathematical reasoning, logical reasoning, code generation, factual knowledge use, and open-ended dialogue. While such mixtures improve broad instruction-following ability, they also force diverse optimization objectives to share the same parameter space.

This heterogeneity can induce destructive update interactions. Gradients that improve one capability may conflict with gradients required by another, leading to negative transfer, the “seesaw phenomenon” (Yu et al., 2020), shifts in ability composition under different fine-tuning mixtures (Dong et al., 2024), and catastrophic forgetting in sequential or multi-stage settings (McCloskey and Cohen, 1989; Kirkpatrick et al., 2017; Hu et al., 2026). Unlike conventional multi-task learning, instruction tuning often contains substantial variation even within the same dataset or task label. Therefore, the relevant unit of interference is not always the surface task, but the individual instruction-response pair.

A.2 Mitigating Interference in Shared Parameter Spaces

A large body of work mitigates interference while keeping the model fully shared. Data-centric methods adjust the training curriculum, sampling schedule, or replay distribution to balance task exposure (Wei et al., 2021; Rolnick et al., 2019; Lin and Zheng, 2026; Dong et al., 2024), synthesize training data from diverse search trajectories (Liu et al., 2026a), or construct targeted training instances via controllable generation (Fei et al., 2022). Regularization-based continual learning methods, such as Elastic Weight Consolidation, constrain changes to important parameters to preserve previously learned knowledge (Kirkpatrick

et al., 2017; Li and Hoiem, 2017). Gradient-based approaches, including GEM and PCGrad, modify update directions to reduce conflicts among objectives (López-Paz and Ranzato, 2017; Yu et al., 2020), while gradient normalization methods reweight tasks by balancing gradient magnitudes (Chen et al., 2018). These methods operate in a shared parameter space and usually search for a single update direction or a globally regularized trajectory. However, when different instructions require genuinely different parameter movements, a single shared update can be geometrically insufficient. IPI takes a complementary approach: rather than forcing all instructions to agree on one update direction, it routes their updates to instruction-conditioned sparse parameter subspaces.

A.3 Modularization and Parameter-Efficient Adaptation

Another line of work reduces interference by introducing architectural modularity. Mixture-of-Experts models route examples to different expert modules (Fedus et al., 2022; Riquelme et al., 2021), while adapter-based methods and prefix tuning add lightweight trainable modules on top of a frozen backbone (Houlsby et al., 2019; Pfeiffer et al., 2021; Liu et al., 2024; Li and Liang, 2021). These methods provide a form of separation and can reduce direct contention in the backbone. More broadly, conditioning computation on the individual input has a long tradition in adaptive attention and interaction architectures for semantic matching and question answering (Liang et al., 2019b,a; Wang et al., 2022; Song et al., 2022; Xue et al., 2023; Li et al., 2024; Xue et al., 2024), and in hypernetworks that generate parameters conditioned on the input domain (Gui et al., 2018); IPI applies this instance-conditioned principle to the selection of update coordinates rather than to the forward computation. However, modular approaches often introduce additional parameters, routing complexity, or memory overhead. They may also fragment knowledge across modules, especially when the number of fine-grained capabilities grows. In contrast, IPI does not add inference-time modules or alter the final model architecture. The router is used only during training, and the fine-tuned model remains a dense backbone at inference. Orthogonal architectural and systems advances, such as stable long-context positional encodings (Dai et al., 2025) and compressed reasoning representations (Shan et al., 2025), improve backbone efficiency but do

not address update interference; likewise, retrieval-augmented generation sidesteps parameter updates by attaching non-parametric memory (Lewis et al., 2020; Liu et al., 2026c,b), with growing attention to its reliability and security (Liu et al., 2025a; Li et al., 2026), and thus complements rather than replaces interference-aware fine-tuning.

A.4 Parameter-Level Isolation

Parameter-level isolation aims to mitigate interference by partitioning or protecting subsets of the original backbone parameters. Early mask-based continual learning methods allocate task-specific parameter subsets through pruning or binary masks (Mallya and Lazebnik, 2018). Recent work further studies important parameters in language-model fine-tuning and freezes or isolates task-critical coordinates based on update statistics, importance estimates, or adaptation-aware probing (Sorrenti et al., 2023; Wang et al., 2025; Liu et al., 2026d; Han et al., 2026). Dynamic isolation methods improve upon static masks by updating parameter importance over the training trajectory (Lin et al., 2026). Despite their effectiveness, existing isolation methods usually define masks at the task or training-stage level. Static isolation assumes that one fixed mask is suitable for all examples in a task, while global dynamic isolation updates a shared mask over time. Both assumptions are too coarse for heterogeneous instruction tuning: examples within the same task can require different computations, and examples from different tasks can share latent capabilities. IPI addresses this unit mismatch by predicting a separate update mask for each instruction.

A.5 Sparse Mask Learning and Straight-Through Estimation

Sparse fine-tuning and pruning methods select a subset of parameters to update or preserve, often using magnitude, gradient, or movement-based importance signals (Sanh et al., 2020), and lottery-ticket analyses show that pretrained language models contain sparse subnetworks that retain full performance (Zheng et al., 2022). These approaches show that sparse parameter changes can be effective, but they do not directly decide which sparse support should be assigned to each instruction. Learning hard sparse masks also introduces a differentiability challenge because Top- k selection is discrete. Straight-through estimators provide a practical way to pass gradients through hard or dis-

crete decisions (Bengio et al., 2013; Jang et al., 2016). IPI builds on this idea with an identity-trick reparameterization tailored to instruction tuning. The reparameterization preserves the dense forward computation, sends masked gradients to the base model, and exposes a task-loss saliency signal to the router. This allows IPI to learn instruction-conditioned sparse update masks without damaging the pretrained forward pass or adding inference-time parameters.

B Detailed Experimental Setup

This appendix provides the full experimental details for Section 3.

B.1 Datasets

We evaluate IPI on heterogeneous instruction tuning benchmarks covering mathematical reasoning, logical reasoning, code generation, instruction following, and conversational generation. This benchmark composition intentionally introduces capability diversity to expose parameter contention and interference effects. Complementary capability-oriented suites, such as table-reasoning and user-experience benchmarks (Wu et al., 2025c,b, 2026; Hong et al., 2026), illustrate the breadth of skills that instruction-tuned models are expected to cover; we adopt five widely used benchmarks to enable controlled comparison.

Mathematical Reasoning. We use GSM8K (Cobbe et al., 2021), a widely adopted benchmark for multi-step arithmetic reasoning. Performance is evaluated using exact-match accuracy.

Logical Reasoning. We adopt LogiQA (Liu et al., 2020), which requires deductive reasoning over short passages. Performance is measured using accuracy.

Code Generation. We use CodeAlpaca (Chaudhary, 2023) following prior heterogeneous instruction tuning settings. Generation quality is evaluated using CodeBLEU (Ren et al., 2020).

Instruction Following and Dialogue. We evaluate open-ended generation ability using Alpaca (Taori et al., 2023) and UltraChat (Ding et al., 2023); dialogue generation is a knowledge-intensive, open-ended capability (Wu et al., 2025d). Responses are evaluated using GPT-4 (OpenAI et al., 2024) scoring on a 1–10 scale according to correctness, helpfulness, completeness, coherence, and instruction adherence.

Each dataset is evaluated using its standard metric, and all task-specific scores are reported in the main table. We additionally report Avg. Norm. Score as a compact cross-capability summary. Because the benchmarks use heterogeneous native metrics, Avg. Norm. Score is computed by applying a fixed benchmark-level normalization in the evaluation script before macro-averaging across benchmarks. The same normalization constants are used for all methods, models, and random seeds; no method-specific or seed-specific rescaling is applied.

GPT-4 judge protocol. For Alpaca and UltraChat, all model responses are anonymized and randomly shuffled before scoring. The same judge prompt, decoding setting, and evaluation subset are used for every method and every seed.

B.2 Baselines

Full SFT. Standard supervised fine-tuning updates all model parameters throughout the sequential heterogeneous training stream without explicit interference mitigation. This setting represents the dominant instruction tuning paradigm and serves as the primary reference point.

Random Sparse Fine-Tuning. To isolate the effect of capability-conditioned routing, we include a sparse fine-tuning baseline that randomly selects trainable parameter subsets under the same sparsity budget as IPI.

Heuristic Sparse Fine-Tuning. We compare against a non-learned sparse baseline that selects trainable coordinates using a magnitude-based criterion under the same sparsity budget. This baseline controls for whether the gains come merely from sparse updating rather than instruction-conditioned routing.

Static Isolation. We implement static isolation following prior parameter selection methods (Sorrenti et al., 2023; Wang et al., 2025). Each task or training stage is assigned a fixed parameter mask estimated from task-level importance statistics. This baseline directly evaluates whether coarse task identity provides an effective proxy for parameter usage.

Evolving Parameter Isolation (EPI). We further compare against dynamic parameter isolation methods that periodically update parameter importance masks during optimization (Lin et al., 2026). In our sequential heterogeneous SFT setting, EPI maintains one global evolving mask over the training trajectory. Therefore, EPI captures temporal changes

in global parameter importance, whereas IPI predicts a distinct update mask for each instruction within the same batch.

B.3 Implementation Details

We evaluate all methods using strong open-weight language models spanning multiple architectural families: LLaMA-3.1-8B (Grattafiori et al., 2024), Mistral-7B (Jiang et al., 2023), Qwen3-8B (Yang et al., 2025), and Gemma-3-12B (Team et al., 2025).

All models are fine-tuned using AdamW (Loshchilov and Hutter, 2017) with learning rate 1×10^{-5} , batch size 64, and cosine learning-rate decay with 3% warmup. Unless otherwise stated, supervised fine-tuning is conducted for three epochs. All methods use the same task order, data budget, optimization budget, and evaluation subsets.

For IPI, the routing rank is set to $r = 8$ by default. Instruction representations are extracted using last-layer hidden states as described in Section 2.2. Sparse routing uses Top- k parameter selection with a default per-instruction update sparsity of 1%. The load-balancing coefficient is set to $\lambda_{lb} = 0.01$ and linearly annealed to zero. All experiments are conducted on eight NVIDIA A100 GPUs with 80GB memory. Each experiment is repeated using three independent random seeds, and we report average performance throughout.

B.4 Statistical Robustness

Table 8 reports representative mean $\pm$ standard deviation results over three random seeds. The mean values are aligned with the main results in Table 1. The small standard deviations indicate that the observed improvements are stable rather than artifacts of a favorable seed.

C Method Details

This appendix provides supplementary details for the IPI method described in Section 2. We first analyze the complexity of the structured low-rank router and factorized Top- k selection. We then derive the gradients induced by the identity-trick reparameterization and describe how per-instance masked gradients are accumulated in mini-batch training. Finally, we provide the complete training algorithm.

Base Model	Method	GSM8K	CodeAlpaca	LogiQA	Alpaca	UltraChat	Avg. Norm.
LLaMA-3.1-8B	Full SFT	57.6 ± 0.4	29.1 ± 0.5	61.4 ± 0.5	8.12 ± 0.07	8.28 ± 0.08	7.60 ± 0.05
	EPI	63.2 ± 0.3	32.1 ± 0.4	66.1 ± 0.5	8.58 ± 0.06	8.86 ± 0.07	8.32 ± 0.04
	IPI (Ours)	64.7 ± 0.3	33.0 ± 0.3	67.6 ± 0.4	8.81 ± 0.05	9.08 ± 0.06	8.56 ± 0.03
Qwen3-8B	Full SFT	59.2 ± 0.4	30.0 ± 0.5	62.9 ± 0.5	8.32 ± 0.07	8.47 ± 0.08	7.83 ± 0.05
	EPI	64.7 ± 0.3	33.0 ± 0.4	67.8 ± 0.4	8.88 ± 0.06	9.18 ± 0.07	8.55 ± 0.04
	IPI (Ours)	66.1 ± 0.3	34.2 ± 0.3	69.4 ± 0.4	9.07 ± 0.05	9.38 ± 0.06	8.78 ± 0.03

Table 8: Representative statistical robustness results over three random seeds.

C.1 Gradient Derivation of the Identity Trick

We show that the identity-trick reparameterization preserves the dense forward computation while inducing sparse base-model gradients and a router-learning signal.

Recall the STE mask:

$$\tilde{m}_{i,j} = \text{sg}(m_{i,j} - \alpha_{i,j}) + \alpha_{i,j}, \quad (11)$$

where $\text{sg}(\cdot)$ denotes stop-gradient. The effective weight for instruction i and coordinate j is

$$\hat{\theta}_{i,j} = \text{sg}(\theta_j) + \text{sg}(m_{i,j})(\theta_j - \text{sg}(\theta_j)) + \text{sg}(\theta_j)(\tilde{m}_{i,j} - \text{sg}(\tilde{m}_{i,j})). \quad (12)$$

Dense forward identity. Forward-wise, the second term is zero because $\theta_j - \text{sg}(\theta_j)$ has value zero. The third term is also zero because $\tilde{m}_{i,j} - \text{sg}(\tilde{m}_{i,j})$ has value zero. Therefore,

$$\hat{\theta}_{i,j} = \theta_j \quad (13)$$

in the forward pass. IPI therefore preserves the same dense forward computation as ordinary SFT.

Sparse base-model gradient. For the base model, only the second term in Eq. (12) exposes θ_j to autograd. Hence

$$\frac{\partial \hat{\theta}_{i,j}}{\partial \theta_j} = m_{i,j}. \quad (14)$$

Let

$$g_{i,j} = \frac{\partial \mathcal{L}_i(\theta)}{\partial \theta_j}. \quad (15)$$

Then the gradient received by the base model from instruction i is

$$\frac{\partial \mathcal{L}_i}{\partial \theta_j} = m_{i,j} g_{i,j}. \quad (16)$$

Thus the base model is updated only on coordinates selected by the instruction-level mask.

Router saliency signal. For the router, only the third term in Eq. (12) exposes $\tilde{m}_{i,j}$. Therefore,

$$\frac{\partial \hat{\theta}_{i,j}}{\partial \tilde{m}_{i,j}} = \theta_j, \quad (17)$$

and

$$\frac{\partial \mathcal{L}_i}{\partial \tilde{m}_{i,j}} = g_{i,j} \theta_j. \quad (18)$$

Since the STE in Eq. (11) passes gradients from $\tilde{m}_{i,j}$ to $\alpha_{i,j}$, the router receives

$$\frac{\partial \mathcal{L}_i}{\partial \alpha_{i,j}} \approx g_{i,j} \theta_j. \quad (19)$$

Under gradient descent, this promotes coordinates with large descent saliency

$$s_{i,j} = -g_{i,j} \theta_j. \quad (20)$$

This is a movement-style first-order importance signal computed separately for each instruction rather than accumulated globally.

Summary. The identity trick defines three behaviors simultaneously:

$$\hat{\theta}_{i,j} = \theta_j, \quad \frac{\partial \mathcal{L}_i}{\partial \theta_j} = m_{i,j} g_{i,j}, \quad \frac{\partial \mathcal{L}_i}{\partial \alpha_{i,j}} \approx g_{i,j} \theta_j. \quad (21)$$

This mechanism keeps the pretrained forward computation dense, applies sparse base-model updates, and trains the router directly from task-loss saliency.

C.2 Per-Instance Masked Gradient Accumulation

In ordinary batched linear layers, weight gradients are reduced over the batch before the optimizer update. IPI requires per-instance masks, because each instruction may select a different update support. Since the training loss is the batch mean,

$$\mathcal{L}_{\text{CE}} = \frac{1}{B} \sum_{i \in \mathcal{B}} \mathcal{L}_i, \quad (22)$$

the masked base-model gradient is

$$\nabla_{\theta} \mathcal{L}_{\text{base}} = \frac{1}{B} \sum_{i \in \mathcal{B}} \mathbf{m}_i \odot \mathbf{g}_i. \quad (23)$$

If the implementation uses a summed loss rather than a mean loss, the factor $1/B$ is absorbed into the learning rate.

This can be implemented with a custom autograd function for routed linear layers. The forward pass remains the standard dense linear transformation. In the backward pass, each example's weight-gradient contribution is first masked by its instruction-specific update mask, and the masked contributions are then averaged across the mini-batch. This is mathematically equivalent to applying the identity-trick reparameterization independently to each instruction and then accumulating the resulting sparse gradients. For debugging or small-scale experiments, per-sample gradient utilities such as functorch can be used to verify the same behavior, although a custom autograd implementation is more efficient for large models.

C.3 Support-Overlap View of Interference

IPI reduces interference by controlling where different instructions are allowed to update. For two instructions i and j , let

$$S_i = \{t : m_{i,t} = 1\}, \quad S_j = \{t : m_{j,t} = 1\} \quad (24)$$

denote their selected supports. Their effective update interaction is

$$\mathcal{I}(i, j) = \langle \mathbf{m}_i \odot \mathbf{g}_i, \mathbf{m}_j \odot \mathbf{g}_j \rangle = \sum_{t \in S_i \cap S_j} g_{i,t} g_{j,t}. \quad (25)$$

Thus destructive cross-terms can occur only on coordinates selected by both instructions. By Cauchy–Schwarz,

$$|\mathcal{I}(i, j)| \leq \left\| \mathbf{g}_i^{S_i \cap S_j} \right\|_2 \left\| \mathbf{g}_j^{S_i \cap S_j} \right\|_2. \quad (26)$$

A looser but more explicit support-overlap bound is

$$|\mathcal{I}(i, j)| \leq |S_i \cap S_j| \|\mathbf{g}_i\|_{\infty} \|\mathbf{g}_j\|_{\infty}. \quad (27)$$

This bound does not imply that overlap alone determines interference, since gradient energy may be concentrated on a small number of coordinates. Rather, it shows that reducing unnecessary overlap removes coordinates on which destructive interactions can occur. IPI uses identity-trick saliency to select coordinates useful for the current instruction and load balancing to discourage all instructions from collapsing to the same support.

C.4 Training Algorithm

Algorithm 1 summarizes the full IPI procedure. The notation $\text{AdamWDelta}(\cdot)$ denotes the additive AdamW parameter delta, including the decoupled weight-decay term. IPI applies the final update gate only to the base-model delta; router parameters are updated normally and are discarded after fine-tuning.

Algorithm 1 Instruction-Level Parameter Isolation (IPI)

Input: Pre-trained model θ , router Φ , dataset $\mathcal{D}$, layer-wise sparsity budgets $\{k_{\ell}\}_{\ell=1}^L$, load-balancing weight λ_{lb}

Output: Fine-tuned dense model θ ; router Φ is discarded after training

- 1: Initialize router Φ with near-uniform row and column factors
- 2: **for** each training step **do**
- 3: Sample mini-batch $\mathcal{B} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^B$
- 4: Build stop-gradient capability vectors $\{\mathbf{c}_i\}_{i \in \mathcal{B}}$ using Eq. (2)
- 5: **for** each routed layer ℓ **do**
- 6: Compute row/column factors $\{\mathbf{u}_i^{(\ell)}, \mathbf{v}_i^{(\ell)}\}_{i \in \mathcal{B}}$ using Eq. (4)
- 7: Select instruction masks $\mathbf{m}_i^{(\ell)} \leftarrow \text{FactorizedTopK}(\mathbf{u}_i^{(\ell)}, \mathbf{v}_i^{(\ell)}, k_{\ell})$
- 8: Build STE masks $\tilde{\mathbf{m}}_i^{(\ell)} \leftarrow \text{sg}(\mathbf{m}_i^{(\ell)} - \alpha_i^{(\ell)}) + \alpha_i^{(\ell)}$
- 9: **end for**
- 10: Run dense forward computation with identity-trick routed layers
- 11: $\mathcal{L}_{\text{CE}} \leftarrow B^{-1} \sum_{i \in \mathcal{B}} \mathcal{L}_i$
- 12: $\mathcal{L}_{\text{total}} \leftarrow \mathcal{L}_{\text{CE}} + \lambda_{\text{lb}} \mathcal{L}_{\text{lb}}$
- 13: Backpropagate through the identity trick
- 14: Accumulate base gradient $\mathbf{G}_{\theta} \leftarrow B^{-1} \sum_{i \in \mathcal{B}} \mathbf{m}_i \odot \mathbf{g}_i$
- 15: Compute router gradient $\mathbf{G}_{\Phi} \leftarrow \nabla_{\Phi} \mathcal{L}_{\text{total}}$
- 16: Compute AdamW delta $\Delta \theta_t \leftarrow \text{AdamWDelta}(\theta_t, \mathbf{G}_{\theta})$
- 17: Build batch update gate $\tilde{\mathbf{m}}_{\mathcal{B}, j} \leftarrow \mathbb{I}(\sum_{i \in \mathcal{B}} m_{i,j} > 0)$
- 18: Update base model $\theta_{t+1} \leftarrow \theta_t + \tilde{\mathbf{m}}_{\mathcal{B}} \odot \Delta \theta_t$
- 19: Update router $\Phi_{t+1} \leftarrow \text{AdamWUpdate}(\Phi_t, \mathbf{G}_{\Phi})$
- 20: Anneal λ_{lb}
- 21: **end for**

The algorithm preserves the dense forward pass while enforcing instruction-conditioned sparse updates. The final-update gate is important under AdamW because decoupled weight decay can otherwise modify coordinates with zero masked gradient. In implementation, the gate is applied to the base-model parameter delta, while router parameters are optimized normally from the task-loss saliency and load-balancing gradients.

C.5 Complexity Analysis

Router parameter count. Under the shared-bottleneck parameterization in Eq. (3)–(4), the router for layer ℓ contains

$$|\Phi^{(\ell)}| = rh + r(p_{\ell} + q_{\ell}) \quad (28)$$

parameters, where h is the dimension of the capability vector, r is the routing rank, and (p_ℓ, q_ℓ) are the output and input dimensions of the routed weight matrix. If separate row and column bottlenecks are used, the count becomes $2rh + r(p_\ell + q_\ell)$. We use the shared-bottleneck form by default. For LLaMA-style 7B/8B backbones with routed attention and MLP projections and $r = 8$, this gives roughly 3×10^7 router parameters, around 0.4% of a 7B model. The router is therefore small relative to the base model and is discarded after fine-tuning.

Per-instance memory. For each instance and routed layer, the router outputs only $\mathbf{u}_i^{(\ell)} \in \mathbb{R}^{p_\ell}$ and $\mathbf{v}_i^{(\ell)} \in \mathbb{R}^{q_\ell}$. The full per-weight tensor $\alpha_i^{(\ell)} \in \mathbb{R}^{p_\ell \times q_\ell}$ is not stored. Thus the per-instance routing state scales as $O(\sum_\ell (p_\ell + q_\ell))$ rather than $O(\sum_\ell p_\ell q_\ell)$.

Computational overhead. Computing the bottleneck and factor projections costs $O(rh + r(p_\ell + q_\ell))$ per instance and layer, which is small compared with the base linear transformation cost $O(p_\ell q_\ell)$. The main additional cost comes from factorized Top- k selection over outer-product scores and from passing router gradients through the factorized scores. Since the router is used only during training, IPI introduces no inference-time routing cost.

Factorized Top- k selection. For each routed matrix $\mathbf{W}^{(\ell)} \in \mathbb{R}^{p_\ell \times q_\ell}$, the router defines non-negative scores through the outer product

$$\alpha_{i,(a,b)}^{(\ell)} = u_{i,a}^{(\ell)} v_{i,b}^{(\ell)}. \quad (29)$$

The dense score matrix is never materialized. Instead, we sort the row and column factors and enumerate large products with a heap-based search over index pairs. This returns the largest k_ℓ coordinate pairs while avoiding the $O(p_\ell q_\ell)$ memory cost of dense score construction. After sorting, the selection cost is approximately

$$O((p_\ell + q_\ell) \log(p_\ell + q_\ell) + k_\ell \log k_\ell). \quad (30)$$

For very large layers, candidate-pruned variants can also be used, but all experiments avoid materializing the full $p_\ell q_\ell$ score matrix.

Factor-level load balancing. For each routed layer, the predicted factors are first normalized to unit mass,

$$\tilde{\mathbf{u}}_i^{(\ell)} = \frac{\mathbf{u}_i^{(\ell)}}{\mathbf{1}^\top \mathbf{u}_i^{(\ell)}}, \quad \tilde{\mathbf{v}}_i^{(\ell)} = \frac{\mathbf{v}_i^{(\ell)}}{\mathbf{1}^\top \mathbf{v}_i^{(\ell)}}, \quad (31)$$

and then averaged over the mini-batch,

$$\bar{\mathbf{u}}^{(\ell)} = \frac{1}{B} \sum_i \tilde{\mathbf{u}}_i^{(\ell)}, \quad \bar{\mathbf{v}}^{(\ell)} = \frac{1}{B} \sum_i \tilde{\mathbf{v}}_i^{(\ell)}. \quad (32)$$

The resulting regularizer

$$\mathcal{L}_{\text{lb}} = - \sum_\ell \left[H(\bar{\mathbf{u}}^{(\ell)}) + H(\bar{\mathbf{v}}^{(\ell)}) \right] \quad (33)$$

depends only on batch-averaged row and column factors. It is computed in $O(B \sum_\ell (p_\ell + q_\ell))$ time and does not require coordinate-level entropy over the full outer product.

D Formal Analysis of Instruction-Level Parameter Isolation

We provide a formal perspective on IPI, focusing on three questions: (i) how sparse masks affect gradient interference, (ii) why instruction-level masks are more expressive than task-level masks, and (iii) why the identity-trick reparameterization provides a well-defined router learning signal. The goal of this analysis is not to prove universal dominance over task-level isolation, but to clarify the optimization structure induced by instruction-conditioned sparse updates.

D.1 Notation and Setup

Let $\theta \in \mathbb{R}^d$ denote the model parameters. For each example $(\mathbf{x}_i, \mathbf{y}_i)$, define the per-example loss

$$\ell_i(\theta) = -\log p(\mathbf{y}_i | \mathbf{x}_i; \theta), \quad (34)$$

and its gradient

$$\mathbf{g}_i = \nabla_\theta \ell_i(\theta) \in \mathbb{R}^d. \quad (35)$$

IPI predicts an instance-conditioned binary update mask $\mathbf{m}_i \in \{0, 1\}^d$ with $\|\mathbf{m}_i\|_0 = k$. The corresponding soft importance scores are denoted by $\alpha_i \in (0, 1)^d$, whose factorized form for layer ℓ is

$$\alpha_{i,(a,b)}^{(\ell)} = u_{i,a}^{(\ell)} v_{i,b}^{(\ell)}. \quad (36)$$

Ignoring optimizer-specific preconditioning for clarity, the masked SGD-style batch update induced by IPI is

$$\Delta \theta = -\eta \frac{1}{B} \sum_{i \in \mathcal{B}} \mathbf{m}_i \odot \mathbf{g}_i. \quad (37)$$

In the actual implementation, this masked gradient is used inside AdamW, and the final AdamW parameter delta is further gated to prevent unselected coordinates from being changed by decoupled weight decay.

D.2 Gradient Interference and Mask Overlap

For two examples i and j , define their masked update interaction as

$$\mathcal{I}(i, j) = \langle \mathbf{m}_i \odot \mathbf{g}_i, \mathbf{m}_j \odot \mathbf{g}_j \rangle. \quad (38)$$

Let

$$S_i = \{t : m_{i,t} = 1\}, \quad S_j = \{t : m_{j,t} = 1\} \quad (39)$$

be the selected supports. Then

$$\mathcal{I}(i, j) = \sum_{t \in S_i \cap S_j} g_{i,t} g_{j,t}. \quad (40)$$

This expression shows that masked interference can occur only on coordinates selected by both instructions.

Without assuming independence between masks and gradients, the following deterministic bound holds:

$$\begin{aligned} |\mathcal{I}(i, j)| &\leq \left\| \mathbf{g}_i^{S_i \cap S_j} \right\|_2 \left\| \mathbf{g}_j^{S_i \cap S_j} \right\|_2 \\ &\leq |S_i \cap S_j| \|\mathbf{g}_i\|_\infty \|\mathbf{g}_j\|_\infty. \end{aligned} \quad (41)$$

This bound is deliberately conservative. It does not claim that interference is determined only by support overlap, since gradient energy may be concentrated on a small number of coordinates. Instead, it shows a structural property: reducing unnecessary overlap removes coordinates on which destructive cross-terms can occur. This is the formal basis for using instruction-conditioned masks to control update interactions.

D.3 Task-Level versus Instruction-Level Isolation

Task-level isolation assigns a single mask $\mathbf{m}_T$ to all examples from task T . For two examples $i, j \in T$, the effective interaction is

$$\mathcal{I}^{\text{task}}(i, j) = \langle \mathbf{m}_T \odot \mathbf{g}_i, \mathbf{m}_T \odot \mathbf{g}_j \rangle. \quad (42)$$

Because all examples in the same task share the same support, task-level isolation cannot adapt the overlap pattern for different instruction pairs within that task. This is restrictive when a task contains both compatible and conflicting examples.

IPI introduces an instance-pair-specific degree of freedom by predicting $\mathbf{m}_i$ and $\mathbf{m}_j$ separately from their instruction representations. If two instructions are routed to disjoint supports, their masked interaction is exactly zero:

$$S_i \cap S_j = \emptyset \quad \Rightarrow \quad \mathcal{I}(i, j) = 0. \quad (43)$$

If two instructions require compatible computations, the router can still assign overlapping supports. Thus instruction-level isolation strictly generalizes task-level isolation in representational flexibility: task-level masking is the special case where all instructions in the same task receive the same mask.

This does not imply that IPI always produces lower interference for every pair, because the learned masks depend on the router, the data distribution, and the gradient energy on selected coordinates. Rather, it shows why instruction-level routing can represent both within-task separation and cross-task sharing, while task-level masks cannot.

D.4 Sparse Routing as a Constrained Optimization View

For the masked batch update

$$\mathbf{u} = \frac{1}{B} \sum_{i=1}^B \mathbf{m}_i \odot \mathbf{g}_i, \quad (44)$$

the squared update norm expands as

$$\|\mathbf{u}\|_2^2 = \frac{1}{B^2} \left[\sum_i \|\mathbf{m}_i \odot \mathbf{g}_i\|_2^2 + \sum_{i \neq j} \mathcal{I}(i, j) \right]. \quad (45)$$

An ideal router should preserve useful per-instruction descent while avoiding unnecessary destructive cross-terms. Directly optimizing this objective under the hard constraint $\|\mathbf{m}_i\|_0 = k$ is combinatorial. IPI approximates this objective using two tractable signals.

First, the identity trick provides a movement-style task-loss signal:

$$\frac{\partial \mathcal{L}_i}{\partial \alpha_{i,j}} \approx g_{i,j} \theta_j. \quad (46)$$

Under gradient descent, this increases routing scores on coordinates with large descent saliency

$$s_{i,j} = -g_{i,j} \theta_j. \quad (47)$$

Second, the load-balancing loss discourages all instructions in a batch from collapsing to the same row and column factors. Together, these two signals form a practical relaxation of instruction-conditioned sparse update routing: the router is encouraged to select useful coordinates for each instruction while avoiding excessive shared support.

D.5 Well-Definedness of the Router Signal

A central concern is that the forward pass remains dense, so the cross-entropy loss may appear independent of the hard mask and hence independent of α_i . The identity-trick reparameterization resolves this by keeping the forward value identical to θ while defining separate backward paths for the base model and the router.

Proposition 1 (Gradients induced by the identity trick). *Let*

$$\tilde{m}_{i,j} = \text{sg}(m_{i,j} - \alpha_{i,j}) + \alpha_{i,j}, \quad (48)$$

and define

$$\hat{\theta}_{i,j} = \text{sg}(\theta_j) + \text{sg}(m_{i,j})(\theta_j - \text{sg}(\theta_j)) + \text{sg}(\theta_j)(\tilde{m}_{i,j} - \text{sg}(\tilde{m}_{i,j})). \quad (49)$$

For $\mathcal{L}_i = \mathcal{L}_i(\hat{\theta}_i)$ and $g_{i,j} = \partial \mathcal{L}_i(\theta) / \partial \theta_j$, any autograd system respecting stop-gradient gives:

$$\hat{\theta}_{i,j} = \theta_j \quad \text{in the forward pass}, \quad (50)$$

$$\frac{\partial \mathcal{L}_i}{\partial \theta_j} = m_{i,j} g_{i,j}, \quad (51)$$

$$\frac{\partial \mathcal{L}_i}{\partial \alpha_{i,j}} \approx g_{i,j} \theta_j. \quad (52)$$

Sketch. The first claim follows because both residual terms have zero forward value. For the base model, only the second term exposes θ_j to autograd, giving derivative $m_{i,j}$. For the router, only the third term exposes $\tilde{m}_{i,j}$, giving $\partial \mathcal{L}_i / \partial \tilde{m}_{i,j} = g_{i,j} \theta_j$; the STE bridge then passes this gradient to $\alpha_{i,j}$. $\square$

Saliency interpretation. The router gradient is $g_{i,j} \theta_j$, but gradient descent updates the router in the negative gradient direction. Therefore, the relevant descent saliency is

$$s_{i,j} = -g_{i,j} \theta_j. \quad (53)$$

A large positive $s_{i,j}$ means that increasing the routing score for coordinate j is locally aligned with reducing the current instruction loss. This is a movement-pruning-style signal delivered per instruction rather than accumulated globally.

D.6 Optimization Remarks

The analysis above should be interpreted as a structural explanation rather than a convergence theorem. Jointly optimizing the backbone and the router is a non-convex and non-smooth problem due to

deep-network parameterization, hard Top- k selection, and the use of STE. Moreover, the routing decision depends on instruction representations extracted from the evolving model, even though these representations are detached from higher-order gradients.

Nevertheless, for a fixed router and fixed capability extractor, the base-model update at coordinate j accumulates gradients only from examples routed to that coordinate:

$$\Delta \theta_j = -\eta \frac{1}{B} \sum_{i \in \mathcal{B}: m_{i,j}=1} g_{i,j}. \quad (54)$$

Thus IPI can be viewed as optimizing a sparse, data-dependent update field over the original dense parameter space. The identity trick ensures that this sparse update field is learned without changing the dense forward computation, while the final AdamW gate ensures that unselected coordinates are not modified by decoupled weight decay.

E Additional Analyses

This appendix provides additional analyses of IPI beyond the main results. We study optimization dynamics, mask-induced gradient interactions, capability-level mask structure, the mixed-batch training regime, robustness under distribution shift, sparsity-performance trade-offs, and representative case patterns.

E.1 Training Dynamics

Section 4.4 reports throughput, peak memory, and batch-size scalability. Here we additionally examine validation loss trajectories and gradient-conflict diagnostics. Unless otherwise specified, measurements are conducted on LLaMA-3.1-8B using the same sequential heterogeneous SFT setting as the main experiments.

E.1.1 Validation Loss Dynamics

We track validation loss across training epochs to examine whether instruction-level routing destabilizes optimization. Table 9 reports loss trajectories on Qwen3-8B.

IPI achieves the lowest validation loss at all measured checkpoints while maintaining a comparable convergence rate. This suggests that instruction-level routing does not introduce optimization instability. The identity-trick reparameterization preserves dense forward computation, while sparse instruction-conditioned updates reduce unnecessary update sharing. The load-balancing coefficient

Method	Epoch 1	Epoch 2	Epoch 3	Δ
Full SFT	2.41	2.18	2.12	0.29
Static Isolation	2.35	2.09	2.01	0.34
EPI	2.22	1.98	1.91	0.31
IPI (Ours)	2.15	1.89	1.83	0.32

Table 9: Validation loss across training epochs on Qwen3-8B. Δ denotes the reduction from epoch 1 to epoch 3.

Method	Mean Pairwise Cos.	Conflict Rate
Full SFT	−0.03	31.2%
Static Isolation	−0.01	24.7%
EPI	+0.05	19.3%
IPI (Ours)	+0.12	11.6%

Table 10: Additional gradient conflict diagnostics on LLaMA-3.1-8B. Conflict rate is the fraction of instruction pairs whose masked updates have cosine similarity below −0.15; the same definition is used in Table 18.

is annealed to zero during training, encouraging early routing diversity without imposing a persistent entropy constraint.

E.1.2 Additional Gradient Conflict Diagnostics

The main paper reports aligned and conflicting update-pair analysis. Here we provide an additional diagnostic based on pairwise cosine similarity over random mini-batches. For each instance pair, we compare the cosine similarity of the effective masked updates. Conflict rate denotes the fraction of pairs with cosine similarity below −0.15.

Table 10 shows that IPI reduces the fraction of strongly conflicting instance pairs. This is consistent with the support-overlap view in Appendix C.3: destructive cross-terms can occur only on coordinates selected by both instructions. IPI does not eliminate overlap entirely; rather, it learns to preserve overlap for compatible examples while reducing unnecessary overlap among conflicting ones.

E.2 Capability Structure of Learned Masks

A central motivation of IPI is that parameter usage is better explained by instruction-level capability requirements than by coarse task labels. We therefore analyze whether the learned masks recover capability-level structure.

Method	Task NMI	Capability NMI
Random Mask	0.02	0.03
Static Isolation	0.61	0.42
EPI	0.55	0.48
IPI (Ours)	0.58	0.72

Table 11: Mask clustering under task and capability labelings.

	GSM8K	LogiQA	Code	Alpaca	UChat
GSM8K	0.21	0.28	0.24	0.12	0.11
LogiQA	0.28	0.25	0.39	0.14	0.13
Code	0.24	0.39	0.22	0.10	0.09
Alpaca	0.12	0.14	0.10	0.18	0.30
UChat	0.11	0.13	0.09	0.30	0.20

Table 12: Mean Jaccard similarity of IPI masks across task categories.

E.2.1 Mask Clustering Analysis

For held-out instructions, we collect the binary masks produced by the trained router and compute compact mask embeddings using layer-wise Jaccard sketches. We then apply k -means clustering and measure alignment with two label sets: surface task identity and broader capability groups. The capability groups merge tasks that share core computational demands, such as mathematical/logical reasoning, code generation, and dialogue. Clustering quality is measured by Normalized Mutual Information (NMI).

IPI achieves the highest capability NMI, while Static Isolation achieves the highest task NMI. This pattern supports the intended behavior of IPI: the router does not merely reproduce dataset boundaries, but organizes masks around finer-grained computational structure. EPI lies between the two because it updates a global mask over time but does not condition masks on individual instructions.

E.2.2 Task-Pair Mask Similarity

We further compute mean Jaccard similarity between top- k masks for instruction pairs sampled from different task categories. Results are reported in Table 12.

The diagonal values are modest, indicating substantial intra-task heterogeneity. Meanwhile, some cross-task pairs, such as LogiQA–CodeAlpaca, show relatively high overlap. This supports the claim that task identity is an imperfect proxy for parameter usage: some examples within a task re-

Setting	Method	LLaMA-3.1-8B	Qwen3-8B
Sequential	Full SFT	7.60	7.83
	EPI	8.32	8.55
	IPI (Ours)	8.56	8.78
Mixed-Batch	Full SFT	7.85	8.06
	EPI	8.18	8.41
	IPI (Ours)	8.54	8.75

Table 13: Avg. Norm. Score under sequential multi-stage and mixed-batch SFT. IPI routes at the instruction level and is therefore largely insensitive to whether domains are separated into stages.

Method	Seen	Unseen	Drop
Full SFT	7.83	6.92	−0.91
Static Isolation	8.20	7.17	−1.03
EPI	8.55	7.65	−0.90
IPI (Ours)	8.78	8.00	−0.78

Table 14: Generalization to held-out instruction distributions.

quire different update supports, while examples from different tasks may share latent symbolic capabilities.

E.3 Mixed-Batch SFT

Table 13 reports the full mixed-batch results summarized in Section 4.4. All capability domains are uniformly mixed within mini-batches, under the same data budget, optimizer, and evaluation protocol as the sequential setting.

Mixing removes the ordering that drives stage-induced forgetting, so Full SFT improves. EPI, whose single global evolving mask relies on that stage structure, loses most of its advantage and cannot specialize within a heterogeneous batch. IPI is essentially unchanged and still leads on both backbones, showing that instruction-conditioned routing applies beyond sequential training.

E.4 Out-of-Distribution Generalization

We evaluate whether instruction-conditioned routing improves robustness to distribution shifts in the fine-tuning mixture. We construct a held-out instruction setting by excluding one dialogue-oriented distribution and a subset of instruction templates during training. At test time, we evaluate on both seen-style examples and held-out/paraphrased examples. Table 14 reports Avg. Norm. Score.

IPI shows the smallest drop under held-out

Method	Per-Instruction Budget	Avg. Norm.
Full SFT	100%	7.83
Static Isolation	1.0%	8.20
EPI	1.0%	8.55
IPI (0.5%)	0.5%	8.61
IPI (1.0%)	1.0%	8.78
IPI (2.0%)	2.0%	8.79

Table 15: Sparsity–performance trade-off on Qwen3-8B.

instruction distributions. This suggests that instruction-conditioned routing is less tied to fixed task boundaries than Static Isolation. The result should be interpreted as an empirical robustness diagnostic rather than evidence of universal OOD generalization, since the difficulty of the held-out split depends on the specific excluded templates and distributions.

E.5 Sparsity–Performance Trade-off

Section 4.4 reports matched-budget comparisons on LLaMA-3.1-8B. Here we replicate the trade-off on Qwen3-8B and additionally include the dense Full SFT reference point.

IPI achieves strong performance under very sparse update budgets. Increasing the budget from 0.5% to 1.0% improves performance, while further increasing it to 2.0% yields only marginal gains. This saturation suggests that routing quality, rather than simply the number of updated parameters, is the main driver of performance.

E.6 Case Study

We provide representative case patterns to illustrate how instruction-level routing differs from task-level masking. The goal is not to claim that every example follows the same pattern, but to show concrete cases where task identity misrepresents parameter usage.

E.6.1 Intra-Task Variation and Cross-Task Sharing

We select two contrasting instruction pairs. Pair A contains two GSM8K examples with different reasoning demands: one direct arithmetic example and one multi-step rate/time reasoning example. Pair B contains one LogiQA example and one CodeAlpaca example that both require symbolic structure tracking. We report Jaccard similarity of top- k masks averaged over routed linear layers.

Pair	Instruction Summary	Task	IPI	Static
A1	<i>Direct arithmetic reasoning sample</i>	GSM8K	0.17	1.00
A2	<i>Multi-step rate/time reasoning sample</i>	GSM8K		
B1	<i>Deductive consistency reasoning sample</i>	LogiQA	0.39	0.00
B2	<i>Algorithmic invariant/code-generation sample</i>	CodeAlpaca		

Table 16: Representative mask-overlap cases. IPI can separate heterogeneous examples within a task and preserve sharing across compatible tasks.

Method	Mean Cos.	Median Cos.	% Cos. < -0.3
Full SFT	-0.02	-0.01	18.7%
Static Isolation	0.11	0.13	9.1%
IPI (Ours)	0.21	0.24	4.3%

Table 17: Gradient conflict statistics over sampled instruction pairs.

Pair A shows low IPI overlap despite identical task labels, suggesting intra-task heterogeneity. Pair B shows higher IPI overlap despite different task labels, suggesting cross-task sharing of symbolic computation. Static task-level masking cannot express either pattern because its overlap is determined by task identity.

E.6.2 Case-Level Gradient Conflict

We also compare gradient conflict statistics over sampled instance pairs. For each pair, we compute cosine similarity after applying the method-specific effective update mask. Table 17 shows that IPI shifts the conflict distribution toward more aligned effective updates and reduces the fraction of strongly negative pairs. Together with the mask-overlap cases above, this supports the claim that instruction-conditioned routing captures a finer parameter-usage structure than rigid task boundaries.

F Additional Ablations

This appendix provides additional ablations and diagnostics for the IPI design. Unless otherwise specified, all experiments use LLaMA-3.1-8B and the same sequential heterogeneous training stream as the main experiments. The goal is to further verify whether IPI’s gains come from learned instruction-level routing, rather than from sparsity, mask diversity, or router capacity alone.

F.1 Mask-Collapse Diagnostics

The main ablation shows that removing load balancing degrades downstream performance. We further

Method	Avg. Norm.	Mask Jac. ↓	Coverage ↑	Entropy ↑	Conflict ↓
IPI (full)	8.56	0.19	18.4%	0.91	11.6%
w/o Load Balancing	8.31	0.46	7.1%	0.58	18.7%
Per-Instance Entropy	8.22	0.39	8.6%	0.62	17.4%
MI-Style Balance	8.52	0.22	17.2%	0.88	12.5%
Random Instance Mask	7.69	0.01	38.2%	0.99	28.5%

Table 18: Mask-collapse diagnostics. “Mask Jac.” denotes average pairwise Jaccard similarity between instruction masks, and “Coverage” denotes the fraction of parameters selected by at least one instruction in a batch. Removing load balancing sharply increases mask overlap and reduces support coverage, indicating collapse toward a small shared subnetwork. Random masks achieve low overlap and high coverage but still perform poorly, showing that diversity alone is insufficient without learned saliency.

diagnose whether this degradation is associated with mask collapse. We report four mask-level statistics: (i) average pairwise Jaccard similarity between instruction masks, (ii) union coverage, defined as the fraction of parameters selected by at least one instruction in a batch, (iii) normalized factor entropy of the batch-averaged row and column factors, and (iv) the effective gradient-conflict rate after masking, using the same definition as Appendix E.1.2, i.e. the fraction of instruction pairs whose masked updates have cosine similarity below -0.15. As shown in Table 18, removing load balancing increases average mask Jaccard similarity from 0.19 to 0.46, reduces union coverage from 18.4% to 7.1%, and raises the conflict rate from 11.6% to 18.7%. This confirms that the performance drop is accompanied by mask collapse. Per-instance entropy also underperforms because it encourages each individual mask distribution to become diffuse, which weakens specialization. The MI-style balance variant achieves comparable performance but introduces additional estimation cost. Random instance masks yield very low overlap and high coverage, yet suffer from high conflict and low downstream performance, confirming that learned instruction-conditioned saliency is essential.

Variant	Conditioning Unit	Mask Source	Avg. Norm.
Random Instance Mask	Instruction	Random	7.69
Task-Conditioned Router	Task	Learned	8.35
Task-Averaged IPI Mask	Task	Averaged IPI	8.27
IPI (full)	Instruction	Learned	8.56

Table 19: Routing-granularity ablation: learned instruction-level routing outperforms random instruction-level and task-level compressed IPI masks.

Masking Variant	Top- k Support	Binary Gate	Avg. Norm.
Dense Soft Masking	No	No	8.12
Sparse Soft Top- k	Yes	No	8.32
Hard Top- k IPI	Yes	Yes	8.56

Table 20: Hard-vs-soft masking ablation. Sparse support improves over dense soft weighting, but hard binary update gates produce the strongest isolation effect.

F.2 Granularity of Routing

We further test whether IPI’s advantage comes from instruction-level routing rather than from learning a better static support. Table 19 reports the comparison. Task-conditioned routing improves over random instance masks but remains below full IPI. Compressing learned IPI masks into a single mask per task also loses a substantial fraction of the gain. This supports the claim that instruction-level routing is necessary to capture intra-task heterogeneity and cross-task sharing.

F.3 Hard and Soft Masking

To distinguish the effect of sparsity from the effect of binary isolation, we compare dense soft masks, sparse soft masks, and hard binary masks. Table 20 reports the results. Dense soft masking updates all parameters on every example and therefore weakens the isolation mechanism. Sparse soft Top- k improves over dense soft masking by restricting the support, but binary IPI performs best, suggesting that hard update gates are more effective for preventing conflicting instructions from sharing sensitive coordinates.