Monkey Jump: MoE-Style PEFT for Efficient Multi-Task Learning

Anonymous ACL submission

Abstract

Mixture-of-experts variants of parameter-efficient fine-tuning enable per-token specialization, but they introduce additional trainable routers and expert parameters, increasing memory and training costs. This undermines the core goal of parameter efficient finetuning. We propose *Monkey Jump*[1], which brings MoE-style specialization to PEFT without adding extra trainable parameters for experts and routers. Instead of introducing new PEFT adapters as experts, Monkey Jump treats the PEFT adapters already present in each Transformer block (e.g., query, key, value, up, and down projections) as implicit experts and routes tokens among them. Routing is performed via k-means clustering with EMA-updated centers—no gradients, no learned parameters. We theoretically show that token-wise routing increases expressivity and can outperform shared adapters by avoiding cancellation effects. In multi-task experiments spanning 14 text, 14 image, and 19 video benchmarks, Monkey Jump achieves competitive performance with MoE-PEFT methods while using 7–29$\times$ fewer trainable parameters, up to 48% lower memory, and 1.5–2$\times$ faster training. Monkey Jump is architecture-agnostic and can be applied to any adapter-based PEFT method. Code is available at *Anonymous*.

1 Introduction

Large language models achieve remarkable performance across many tasks, but fine-tuning all their parameters is expensive (Prottasha et al., 2025). Parameter-efficient fine-tuning (PEFT) methods like LoRA (Hu et al., 2022) address this by freezing the pretrained weights and training only small adapter modules. This reduces memory usage and training time while maintaining strong performance. PEFT has become the standard approach for adapting large models to downstream tasks.

However, standard PEFT applies the same adapters uniformly to all inputs. Every token receives the same transformation, regardless of its content. This uniformity limits the model's ability to specialize for different input types, which becomes problematic in multi-task learning where diverse tasks require different adaptations (Ma et al., 2025; Luo et al., 2024).

Recent work combines mixture-of-experts (MoE) with PEFT to address this limitation (Li et al., 2024b; Luo et al., 2024). These methods create multiple adapter experts per layer and use a learned router to select which experts to apply for each input. This enables specialization—different inputs activate different experts—and improves performance on many benchmarks. However, MoE-PEFT methods introduce significant overhead compared to standard PEFT: (i) **more parameters**—multiple experts of a layer multiply the adapter count by $N\times$; (ii) **learned routers**—routing networks add $O(Nd)$ trainable parameters per layer; (iii) **higher memory**—evaluating multiple experts increases activation memory; and (iv) **slower training**—multiple activated experts participate in gradient updates. These costs conflict with the core goal of PEFT: efficient adaptation under resource constraints.

Motivation: *When PEFT is applied to all projections in a Transformer block, it creates separate PEFT adapters. Rather than adding new experts in each projection, Monkey Jump routes tokens among these existing adapters—achieving MoE-style specialization with zero additional expert parameters.*

We propose **Monkey Jump (MJ)**, a method that brings MoE-style specialization to PEFT while preserving its parameter efficiency (Figure 1(c)). Standard PEFT attaches an adapter (e.g., for LoRA, $\Delta\mathbf{W} = \mathbf{BA}$) to each projection (Q, K, V, O, up, gate, down), applying all of them uniformly to every token. MJ routes each token to a subset of these projection adapters based on representation similarity, enabling natural specialization.

[1]Named for the selective activation pattern: adapters "jump" on for some projections and off for others.

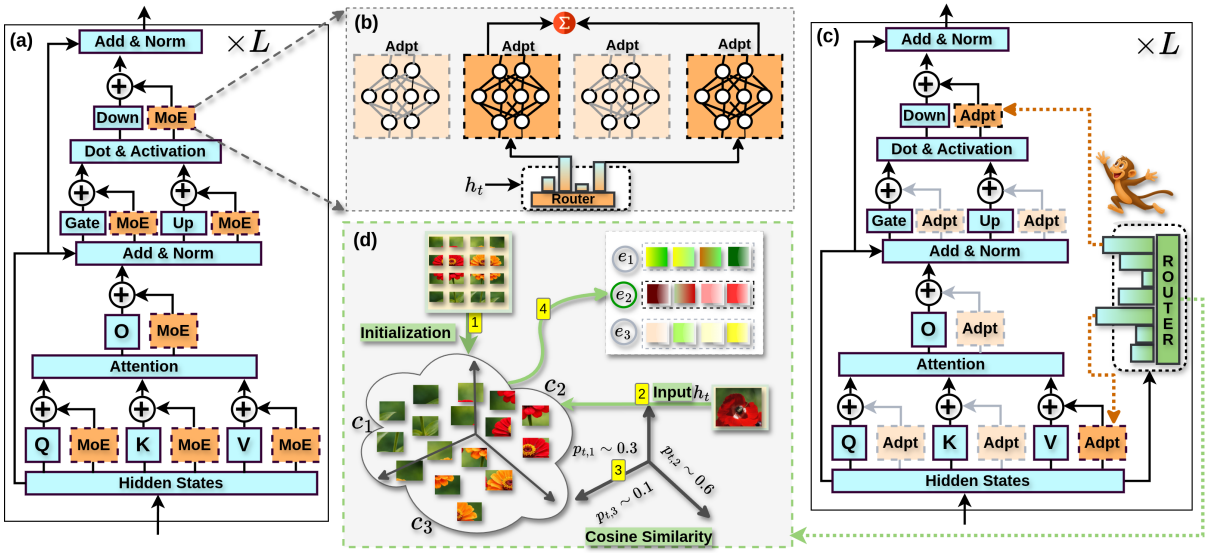

Figure 1: Overview of MJ compared to MoE-PEFT. **(a)** MoE-PEFT architecture: each projection (Q, K, V, O, Gate, Up, Down) has multiple expert adapters with a learned router. **(b)** MoE-PEFT routing: a trainable router selects among N expert adapters, and outputs are summed (Σ). **(c)** MJ architecture: each projection has a single adapter (same as standard PEFT). Here, V and Down adapters are activated (k=2); the rest are skipped.For each token h_t, inactive projections ($m_{t,e} = 0$) apply only the frozen weight: $y_{t,e} = \mathbf{W}_e h_t$. Active projections ($m_{t,e} > 0$) apply $y_{t,e} = \mathbf{W}_e h_t + m_{t,e} \cdot \mathbf{\Delta W}_e h_t$. **(d)** MJ routing mechanism: ① Initialize cluster centers $\mathbf{C}$ via k-means before training; ② For input token h_t, ③ compute cosine similarity to each center; ④ Select top-k experts based on similarity (here e_2 is activated, e_1 and e_3 are skipped).

MJ works in three stages. **(i) Before training:** We run k-means on token representations from a data subset (§5.2) to initialize cluster centers—one per projection. **(ii) During training:** Each token computes cosine similarity to all centers (§D.1). The centers are updated gradually via EMA to track how token patterns change—no gradients needed (§D.3). **(iii) Expert selection:** Based on similarity scores, each token activates the top-k most similar adapters and skips the rest. Only activated adapters contribute to the output. This lets tokens "jump" between adapters based on content, hence the name.

The trainable parameter count remains *exactly the same* as standard PEFT. This is because MJ introduces no new trainable components: (i) existing adapters serve as implicit experts—no additional expert parameters are added; (ii) routing centers are non-trainable buffers updated via EMA, not learned routers optimized by gradients. The only additions are the cluster centers ($O(Ed)$ per block), which are negligible compared to model size and do not participate in backpropagation. This results in comparable model size, memory, and trainable parameters to standard PEFT (§D.13).

To motivate why routing helps, we conduct a preliminary experiment comparing two settings using SmolLM-360M with LoRA (r=1) applied to Q, K, V projections. In the **shared adapter** setting,

Setting	Shared Adapters			Task-Specific Adapters		
Task	SST-2	CoLA	MRPC	SST-2	CoLA	MRPC
Accuracy	88.28	60.21	84.62	**89.47**	**60.82**	**85.83**

Table 1: Shared vs. task-specific adapters on GLUE tasks (SST2, CoLA, MRPC). **Shared**: one adapter trained on all tasks jointly. **Task-specific**: separate adapters per task (same total parameters). Specialization improves accuracy without adding parameters.

we train a single set of adapters on all three GLUE tasks jointly—every task uses the same adapter weights, similar to standard process of LoRA. In the **task-specific adapter** setting, we train separate adapters for each task—each task gets its own dedicated adapter weights, but the total parameter count remains identical (we partition the adapters across tasks). As shown in Table 1, task-specific adapters consistently outperform shared adapters across all three tasks (+1.19% on SST-2, +0.61% on CoLA, +1.21% on MRPC). This suggests that different tasks benefit from different adapter configurations, and routing tokens to specialized adapters—rather than applying the same adapter uniformly—can improve performance without adding parameters.

Contributions. **(i)** We propose **Monkey Jump**, a gradient-free routing mechanism that achieves MoE-style specialization by treating existing PEFT adapters as implicit experts. The routing uses k-

means initialization and EMA-updated centers—no learned routers, no additional parameters. **(ii)** We provide **theoretical analysis** showing that routing increases expressivity (Theorem 1) and that last-token routing is optimal for causal Transformers (Theorem 2). **(iii)** We demonstrate **competitive accuracy with superior efficiency** across 47 benchmarks: MJ matches MoE-PEFT performance using 7–29× fewer parameters, 48% lower memory, and 2× faster training. **(iv)** We show that MJ is a **general recipe**: any adapter-based PEFT method can be converted to a mixture-of-experts model by simply adding trainable projections-free routing.

2 Preliminaries

We consider a Transformer with L blocks. Each block receives a sequence of token representations $H = [h_1, \ldots, h_T] \in \mathbb{R}^{d \times T}$ and transforms them through a set of linear projections $\mathcal{S} = \{\mathtt{q}, \mathtt{k}, \mathtt{v}, \mathtt{o}, \mathtt{up}, \mathtt{gate}, \mathtt{down}\}$, corresponding to the attention projections (query, key, value, output) and feed-forward layers (up, gate, down). Each projection $s \in \mathcal{S}$ applies a linear transformation $\mathbf{W_s} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ to its input.

PEFT. PEFT adapts pretrained models by freezing the original weights $\mathbf{W_s}$ and introducing a small trainable adapter $\mathbf{\Delta W_s}$ for each projection. For a token h_t, the transformation becomes

$$y_t = \mathbf{W_s}\, h_t + \mathbf{\Delta W_s}\, h_t. \tag{1}$$

In LoRA (Hu et al., 2022), the adapter is parameterized as a low-rank product $\mathbf{\Delta W_s} = \mathbf{B_s A_s}$ with $\mathbf{A_s} \in \mathbb{R}^{r \times d_{\text{in}}}$ and $\mathbf{B_s} \in \mathbb{R}^{d_{\text{out}} \times r}$, where $r \ll \min(d_{\text{in}}, d_{\text{out}})$ keeps the parameter count small. Standard PEFT applies every adapter uniformly—each token h_t receives the same correction $\mathbf{\Delta W_s}\, h_t$ regardless of its content.

MoE. Mixture-of-experts architectures achieve input-dependent computation by maintaining multiple expert networks $\{f_1, \ldots, f_N\}$ and a router $g(\cdot)$ that selects which experts to apply. The output is a weighted combination $y = \sum_{n=1}^{N} g_n(x) \cdot f_n(x)$, where the router weights satisfy $\sum_n g_n(x) = 1$ and are typically sparse via top-k selection. While MoE enables specialization, it multiplies parameters by N and requires a learned router with $O(Nd)$ additional parameters.

MoE-PEFT. Recent work combines these ideas by instantiating N adapter experts per projection (Figure 1(a)). For example, MoE-LoRA maintains N adapter pairs $\{(\mathbf{A_s^{(n)}}, \mathbf{B_s^{(n)}})\}_{n=1}^{N}$ and computes

$$y_{t,s} = \mathbf{W_s}\, h_t + \sum_{n=1}^{N} g_n(h_t) \cdot \mathbf{B_s^{(n)} A_s^{(n)}}\, h_t. \tag{2}$$

This achieves token-wise specialization but increases trainable parameters by a factor of N and adds a learned router—undermining the efficiency that motivated PEFT.

Notation. We use color-coded notation to distinguish parameter types: $\mathbf{W}$ (frozen), $\mathbf{\Delta W}$ (trainable via gradient descent), and $\mathbf{C}$ (updated via EMA). We denote element-wise multiplication by $\odot$ and ℓ_2 norm by $\|\cdot\|$.

3 Monkey Jump

Monkey Jump (MJ) brings MoE-style specialization to PEFT while preserving the parameter budget of standard PEFT. The core idea is simple: each Transformer block already contains multiple adapters—one per projection (query, key, value, etc.)—and these adapters can serve as *implicit experts*. Rather than applying all adapters uniformly to every token, MJ lets each token select a subset of adapters based on its content.

Concretely, consider a block with $E = |\mathcal{S}|$ adapter projections, indexed by $e \in \{1, \ldots, E\}$. In standard PEFT, every token h_t receives the full contribution from every adapter. In MJ, we introduce a routing coefficient $m_{t,e} \geq 0$ for each token-projection pair that modulates the adapter's contribution:

$$y_{t,e} = \mathbf{W_e}\, h_t + m_{t,e} \cdot \mathbf{\Delta W_e}\, h_t. \tag{3}$$

When $m_{t,e} = 0$, adapter e has no effect on token t; when $m_{t,e} > 0$, it contributes proportionally. The frozen base transformation $\mathbf{W_e}\, h_t$ is always applied—only the adapter contribution is gated. To maintain sparsity, each token activates at most k out of E adapters.

3.1 Routing

Question: *How can we route tokens to adapters without adding trainable parameters?*

In standard MoE, a learned router network maps inputs to expert weights, introducing $O(Ed)$ trainable parameters per block. This conflicts with our goal of parameter efficiency. Instead, we observe that tokens requiring similar adaptations tend to have similar representations (Bal and Sengupta, 2025), motivating a *clustering-based* approach: we group tokens by representation similarity and route each group to the same adapters.

Each block maintains E routing centers $\mathbf{C} =$

$[\mathbf{c}_1, \ldots, \mathbf{c}_E] \in \mathbb{R}^{E \times d}$, one per projection. These centers are non-trainable buffers updated via EMA—no gradients flow through them (Cai et al., 2021). For each token h_t, we compute cosine similarity to each center, convert to probabilities via softmax, and select the top-k projections:

$$z_{t,e} = \frac{1}{\tau} \left\langle \frac{h_t}{\|h_t\|_2}, \frac{\mathbf{c}_e}{\|\mathbf{c}_e\|_2} \right\rangle, \quad p_{t,e} = \frac{\exp(z_{t,e})}{\sum_{e'=1}^{E} \exp(z_{t,e'})}, \quad m_{t,e} = p_{t,e} \cdot \mathbb{I}[e \in \mathrm{TopK}(p_t, k)], \tag{4}$$

where $\tau > 0$ is a temperature controlling routing sharpness. This mechanism routes similar tokens to the same adapters without any learned parameters.

3.2 Center Initialization and Online Updates

The effectiveness of routing depends critically on the quality of centers $\mathbf{C}$. Poor initialization leads to arbitrary routing decisions (Figure 11(c)) that provide no benefit over uniform application (Fedus et al., 2022). We address this in two stages.

Initialization. Before training, we initialize centers using k-means clustering (Arthur and Vassilvitskii, 2006). We randomly sample a subset of training examples, perform a forward pass through the frozen backbone to collect token representations, and run k-means with cosine similarity on the L_2-normalized representations:

$$\mathbf{C} \leftarrow \mathrm{KMeans}\left(\left\{ \frac{h_t}{\|h_t\|_2} \right\}_{t \in \mathcal{D}_{\text{init}}}, E \right), \tag{5}$$

where $\mathcal{D}_{\text{init}}$ is the initialization subset. This ensures that each center corresponds to a distinct cluster in the representation space from the first training iteration (§5.2, §D.11).

Online updates. During training, centers are updated via EMA, entirely outside the gradient path:

$$\mathbf{c}_e \leftarrow \beta\, \mathbf{c}_e + (1-\beta)\, \bar{h}_e, \quad \bar{h}_e = \frac{1}{|\mathcal{B}_e|} \sum_{t \in \mathcal{B}_e} h_t, \tag{6}$$

where $\mathcal{B}_e$ is the set of tokens routed to adapter e in the current batch and $\beta \in (0, 1)$ is the momentum coefficient. If $|\mathcal{B}_e| = 0$, the center remains unchanged. This allows centers to track the evolving token distribution as adapters are trained.

Important: *K-means initialization is crucial for stable training. Random initialization causes ∼3% performance drop (§ D.3).*

3.3 Routing Variants

The routing mechanism can be instantiated at different levels of granularity, trading off specialization against computational cost.

Token-wise routing. Each token h_t is routed independently based on its own representation, as described in Equation 4. This provides fine-grained, per-token specialization and is the default mode.

Sequence-wise routing. Sequence-wise routing follows the same mechanism as token-wise routing, with one difference: instead of routing each token independently, all tokens in a sequence share the same routing decision. The routing is computed using the last token representation h_T, and the resulting coefficients m_e are applied uniformly to all tokens in the sequence.

Finding: *In causal Transformers, the last token h_T contains more mutual information about the full sequence than pooled representations (mean/max), because it has attended to all previous tokens. See Theorem 2.*

Shared adapter. Optionally, multiple projections can be designated as always-active ($m_{t,e^*} = 1$ for all t), providing stable global adaptation alongside routed adapters. These shared adapters contribute to every token regardless of routing decisions. We find that FFN projections work best as shared adapters—in our experiments, we use O and gate as shared adapters while Q, K, V participate in routing (§D.8).

3.4 Computational Cost

MJ exactly preserves the trainable parameter count of the underlying PEFT method—no new learned weights are added. The only additions are: **(i) Routing centers:** non-trainable buffers of size $O(Ed)$ per block. **(ii) Routing computation:** $O(TEd)$ operations for similarity and top-k selection. Both are negligible compared to the adapter forward pass ($O(TEdr)$) or attention ($O(T^2d)$) (§ 5.1, § D.13).

Monkey Jump = Standard PEFT adapters (ΔW) + Trainable Parameter-free routing ($\mathbf{C}$). *Same trainable parameters, better specialization.*

4 Theoretical Analysis

We provide theoretical justifications for two key design choices in MJ: (1) why token-wise routing increases expressivity over uniform adapter application, and (2) why last-token representations are optimal for sequence-wise routing in causal Transformers.

Expressivity of Token-wise Routing. Why does routing help? Intuitively, when all adapters are applied uniformly, their effects are summed for

every token. If two adapters make opposing corrections along some dimension, these corrections cancel, reducing expressivity. Routing avoids this: by sending different tokens through different adapters, each adapter's full effect is preserved for the tokens that need it.

We formalize this by comparing the *output rank* of MJ versus standard PEFT—a measure of how many independent directions the adapter outputs can span. For each adapter ΔW_e, let $\mathcal{C}_e := \mathrm{Col}(\Delta W_e)$ denote its column space, and let $\mathcal{C}_{\text{all}} := \sum_{e=1}^{E} \mathcal{C}_e$ denote the sum of all column spaces.

Standard PEFT applies all projection adapters to all tokens. To analyze expressivity, we consider the aggregate adapter contribution: $U^{\text{PEFT}} = (\sum_e \Delta W_e)H$. This lies in $\mathrm{Col}(\sum_e \Delta W_e)$—the column space of the *summed* matrix—which can be strictly smaller than $\mathcal{C}_{\text{all}}$ due to cancellation. MJ with hard routing (top-1) assigns each token to exactly one adapter, yielding $U^{\text{MJ}} = [\Delta W_1 H_1 \ \cdots \ \Delta W_E H_E]$ where H_e contains the tokens routed to adapter e. This is a horizontal concatenation, so its column space is $\sum_e \mathrm{Col}(\Delta W_e H_e)$—potentially much larger.

Theorem 1 (Expressivity of MJ)**.** *Under hard routing, if all adapters are activated and receive sufficiently diverse inputs (i.e.,* $\mathrm{rank}(\Delta W_e H_e) = \mathrm{rank}(\Delta W_e)$ *for all* e*), then*

$$\mathrm{rank}(U^{\text{MJ}}) \geq \mathrm{rank}(U^{\text{PEFT}}),$$

with strict inequality whenever $\mathrm{Col}(\sum_e \Delta W_e) \subsetneq \sum_e \mathcal{C}_e$.

The proof is in Appendix A; extension to soft top-k routing is in Appendix B.

Example. Consider two rank-1 adapters:

$$\Delta W_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \qquad \Delta W_2 = \begin{bmatrix} 0 & 0 \\ -1 & 0 \end{bmatrix}.$$

Adapter 1 produces outputs along $e_1 = (1, 0)^\top$; adapter 2 along $e_2 = (0, 1)^\top$. Together, $\mathcal{C}_1 + \mathcal{C}_2 = \mathbb{R}^2$. However, their sum $\Delta W_1 + \Delta W_2 = \left[\begin{smallmatrix} 1 & 0 \\ -1 & 0 \end{smallmatrix}\right]$ produces outputs only along $(1, -1)^\top$—a 1D subspace. With input $H = [e_1 \ e_1]$, standard PEFT gives $U^{\text{PEFT}} = \left[\begin{smallmatrix} 1 & 1 \\ -1 & -1 \end{smallmatrix}\right]$ (rank 1), while MJ gives $U^{\text{MJ}} = \left[\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\right]$ (rank 2). Routing doubles the effective rank by avoiding cancellation.

Key insight: *Routing increases expressivity by avoiding cancellation between adapters—each adapter contributes its full column space independently.*

Optimality of Last-Token Routing. For sequence-wise routing, all tokens share the same routing decision based on a single sequence representation. Common choices include mean pooling ($\bar{h} = \frac{1}{T}\sum_t h_t$), max pooling, or the last token (h_T). We show that in causal Transformers, the last token is theoretically optimal.

In causal attention, each token h_t can only attend to positions $1, \ldots, t$. This creates an information asymmetry: early tokens have limited context, while later tokens accumulate information from the entire prefix. We formalize this using mutual information.

Theorem 2 (Information Maximality)**.** *Let* $X = (x_1, \ldots, x_T)$ *be an input sequence and* $h_1, \ldots, h_T$ *be the hidden representations at any layer of a causal Transformer. Under the Information Preservation and Attention Aggregation properties:* ***(i) Monotonicity:*** $I(h_t; X) \leq I(h_{t+1}; X)$ *for all* $t < T$. ***(ii) Maximality:*** $I(h_T; X) \geq I(h_t; X)$ *for all* $t \leq T$. ***(iii) Dominance over pooling:***

$$I(h_T; X) \geq I(\bar{h}; X), \quad \text{where } \bar{h} = \tfrac{1}{T}\textstyle\sum_{t=1}^{T} h_t.$$

Proof sketch. (i) Since $h_t = f_t(x_1, \ldots, x_t)$ is a deterministic function of the causal prefix, $I(h_t; X) = I(h_t; X_{1:t}) \leq H(X_{1:t})$ by the data processing inequality. The Information Preservation property bounds the representation loss $\epsilon_t := H(X_{1:t}) - I(h_t; X_{1:t})$ so that added context in x_{t+1} is not offset by extra loss, giving $I(h_t; X) \leq I(h_{t+1}; X)$. (ii) Follows from (i) by induction. (iii) $\bar{h}$ is a deterministic function of $(h_1, \ldots, h_T)$, so $I(\bar{h}; X) \leq I(h_1, \ldots, h_T; X)$ by DPI; the Attention Aggregation property gives $I(h_1, \ldots, h_T; X) = I(h_T; X)$ (Full proof in § C and empirical evidence D.6). □

Finding: *Last-token routing is information-theoretically optimal for causal Transformers—not just a heuristic. Mean/max pooling dilutes information from early, context-poor tokens (Table 3).*

5 Experiments

We evaluate MJ on multi-task benchmarks spanning text, image, and video. Our experiments address: (i) Is MJ competitive with MoE-PEFT methods while using fewer parameters? (ii) Does MJ improve efficiency? (iii) How do design choices affect performance?

We evaluate on 47 benchmarks across three modalities: **Text** (14 tasks, 98K samples), **Image** (14 tasks, 42K samples), and **Video** (19 tasks, 13K samples). Details are in §G and Table 15. For image/video tasks, we use LLaVA-OneVision-Qwen2-7B (Li et al., 2024a); for text tasks, we use Llama-3-8B-Instruct (Grattafiori et al., 2024). We apply PEFT or MoE-PEFT to projections Q, K, V, O, and gate. Ablations use LLaVA-OneVision-Qwen2-0.5B with adapters applied only to attention projections (Q, K, V, O). We compare against standard

PEFT methods (LoRA (Hu et al., 2022), LoRA-FA (Zhang et al., 2023a), AdaLoRA (Zhang et al., 2023b), Propulsion (Kowsher et al., 2025)) and MoE-PEFT methods (MoELoRA (Luo et al., 2024), HydraLoRA (Tian et al., 2024), MoLA (Gao et al., 2024), MoRE (Zhang et al., 2025), MoA (Cao et al., 2025)). We implement four MJ variants: MJLoRA, MJLoRAFA, MJAdaLoRA, and MJPropulsion. Full hyperparameters are in Appendix H. Table 2 summarizes average performance across task families. Per-task results are in Appendix E. MJ variants achieve competitive performance with MoE-PEFT methods while using 7–29× fewer trainable parameters. On GLUE, MJLoRA achieves the best score (89.91%), edging out HydraLoRA (89.75%) and MoLA (89.74%). On CS&QA, MJLoRAFA achieves the highest accuracy (71.95%), outperforming HydraLoRA (71.73%) and MoRE (71.62%). For Action&Object reasoning, MJAdaLoRA leads all methods (48.06%), ahead of HydraLoRA (47.85%) and MoRE (47.81%). On Motion understanding, MJLoRA achieves the best score (55.02%), while MJPropulsion shows strong results on High-Level Reasoning (59.02%), trailing only MoLA (59.10%) and HydraLoRA (59.05%). Compared to their base PEFT methods, MJ variants show consistent improvements: MJLoRA outperforms LoRA by +0.33% on GLUE, +0.38% on CS&QA, +1.17% on ActObj, +2.34% on Motion, and +1.31% on HighLvl—demonstrating that gradient-free routing provides meaningful specialization at no additional parameter cost.

5.1 Efficiency Analysis

A core goal of MJ is achieving MoE-style specialization without sacrificing PEFT efficiency. We evaluate using LLaVA-OneVision-Qwen2-0.5B with rank 2, applying MoE-based PEFT to attention projections (Q, K, V, O). For fair comparison, all methods use the same environment: H100 GPU, Transformers library, PyTorch, batch size 8, and gradient accumulation 2. Figure 2 compares MJ variants against MoE-PEFT baselines across six metrics.

Parameter efficiency. MJ variants use significantly fewer trainable parameters. MJ-Propulsion requires only 49K parameters—7× fewer than MixLoRA (364K), 19× fewer than HydraLoRA (909K), and 29× fewer than MoELoRA (1,425K). MJ-LoRAFA (98K) and MJ-LoRA (270K) also remain well below all MoE-PEFT baselines. Despite this, total model size remains nearly identical (1,705MB for MJ vs 1,706–1,709MB for MoE-PEFT), as MJ reuses existing adapters rather than adding new experts.

Memory efficiency. MJ reduces peak GPU memory by up to 48%. MJ-Propulsion uses only 12.0GB compared to 23.2GB for MoEAdaLoRA and 22.8GB for MoELoRA. Even the highest-memory MJ variant (MJ-AdaLoRA, 15.4GB) uses 33% less than MoEAdaLoRA. This reduction comes from top-k sparse routing—MJ evaluates fewer adapter branches per forward pass.

Training speed. MJ achieves 1.5–2× faster training. MJ-Propulsion reaches 5.94 it/s throughput and completes training in 5.0 minutes, compared to 3.02–3.83 it/s and 7.7–9.4 minutes for MoE-PEFT methods. All MJ variants exceed 4.8 it/s, while no MoE-PEFT method exceeds 3.9 it/s.

Inference speed. MJ maintains efficiency at inference. Across GLUE tasks, MJ-Propulsion achieves 15.8 it/s on SST-2 versus 9.4–12.8 it/s for MoE-PEFT. On average, MJ variants achieve 10–25% higher inference throughput.

Key result: *MJ achieves 7–29× fewer trainable parameters, up to 48% lower peak memory, and 1.5–2× faster training—while maintaining comparable accuracy to MoE-PEFT. (Table 2).*

5.2 Ablation Study

MJ introduces several design choices: how to initialize routing centers, how often to update them, and how many layers to equip with routing.

Initialization method. Figure 3(a) compares five center initialization strategies: random vectors, normalized random, random token sampling, k-means clustering, and trainable router. K-means achieves the best performance among training-free methods and matches the trainable router, confirming that clustering captures meaningful structure without learned parameters.

K-means sample size. Figure 3(b) varies the number of samples for k-means initialization from 100 to 30K randomly. Performance saturates at 5K–10K samples, indicating that a modest sample size suffices for effective initialization.

Cluster update coverage. Figure 3(c) varies the percentage of training steps where EMA updates are applied. Performance plateaus at 50–70% coverage, suggesting that centers need regular but not

Method	GLUE	CS & QA	ImgCls	VLQA	ActObj	Motion	HighLvl
LoRA (Hu et al., 2022)	$89.58_{\pm 0.56}$	$71.35_{\pm 1.21}$	$94.09_{\pm 0.48}$	$71.52_{\pm 1.97}$	$45.82_{\pm 1.17}$	$52.68_{\pm 1.16}$	$56.63_{\pm 1.22}$
AdaLoRA (Zhang et al., 2023b)	$89.15_{\pm 0.68}$	$71.01_{\pm 1.06}$	$93.53_{\pm 0.61}$	$71.04_{\pm 1.93}$	$45.27_{\pm 1.21}$	$51.89_{\pm 1.17}$	$55.83_{\pm 1.24}$
Propulsion (Kowsher et al., 2025)	$88.97_{\pm 0.53}$	$70.97_{\pm 1.16}$	$93.63_{\pm 0.47}$	$71.24_{\pm 1.85}$	$45.10_{\pm 1.19}$	$51.59_{\pm 1.17}$	$55.55_{\pm 1.14}$
LoRAFA (Zhang et al., 2023a)	$89.06_{\pm 0.50}$	$70.86_{\pm 1.12}$	$93.62_{\pm 0.49}$	$71.06_{\pm 1.91}$	$45.13_{\pm 1.22}$	$51.70_{\pm 1.14}$	$55.73_{\pm 1.29}$
MoELoRA (Luo et al., 2024)	$89.64_{\pm 0.70}$	$71.31_{\pm 1.13}$	$94.30_{\pm 0.53}$	$71.79_{\pm 1.78}$	$45.91_{\pm 1.20}$	$52.88_{\pm 1.07}$	$56.83_{\pm 1.14}$
MixLoRA (Li et al., 2024b)	$89.38_{\pm 0.53}$	$71.59_{\pm 1.07}$	$94.64_{\pm 0.54}$	$72.41_{\pm 1.69}$	$47.31_{\pm 1.11}$	$54.17_{\pm 1.15}$	$58.51_{\pm 1.17}$
HydraLoRA (Tian et al., 2024)	$89.75_{\pm 0.46}$	$71.73_{\pm 1.09}$	$\mathbf{94.86}_{\pm 0.48}$	$71.85_{\pm 1.80}$	$47.85_{\pm 1.15}$	$54.96_{\pm 1.21}$	$59.05_{\pm 1.14}$
MoLA (Gao et al., 2024)	$89.74_{\pm 0.59}$	$71.30_{\pm 1.07}$	$94.03_{\pm 0.53}$	$\mathbf{72.54}_{\pm 1.73}$	$45.78_{\pm 1.20}$	$52.49_{\pm 1.08}$	$\mathbf{59.10}_{\pm 1.12}$
MoRE (Zhang et al., 2025)	$89.72_{\pm 0.53}$	$71.62_{\pm 1.12}$	$94.74_{\pm 0.51}$	$72.47_{\pm 1.69}$	$47.81_{\pm 1.13}$	$54.64_{\pm 1.12}$	$58.49_{\pm 1.23}$
MoA (Cao et al., 2025)	$89.57_{\pm 0.55}$	$71.46_{\pm 1.08}$	$94.46_{\pm 0.39}$	$71.68_{\pm 1.75}$	$46.35_{\pm 1.13}$	$53.23_{\pm 1.17}$	$57.16_{\pm 1.12}$
MJLoRA	$\mathbf{89.91}_{\pm 0.50}$	$71.73_{\pm 0.98}$	$93.39_{\pm 0.44}$	$72.01_{\pm 1.83}$	$46.99_{\pm 1.11}$	$\mathbf{55.02}_{\pm 1.22}$	$57.94_{\pm 1.10}$
MJAdaLoRA	$89.78_{\pm 0.56}$	$71.46_{\pm 1.03}$	$93.84_{\pm 0.47}$	$71.44_{\pm 1.83}$	$\mathbf{48.06}_{\pm 1.17}$	$54.49_{\pm 1.19}$	$58.67_{\pm 1.13}$
MJPropulsion	$89.80_{\pm 0.56}$	$71.40_{\pm 1.04}$	$93.51_{\pm 0.67}$	$72.48_{\pm 1.79}$	$47.46_{\pm 1.21}$	$54.83_{\pm 1.17}$	$59.02_{\pm 1.13}$
MJLoRAFA	$89.90_{\pm 0.52}$	$\mathbf{71.95}_{\pm 1.05}$	$94.03_{\pm 0.65}$	$71.38_{\pm 1.83}$	$47.42_{\pm 1.19}$	$54.07_{\pm 1.15}$	$56.37_{\pm 1.18}$

Table 2: Average performance across task families (mean $\pm$ std over 5 runs). Columns: GLUE, Commonsense & QA, Image Classification, Vision–Language QA, Action & Object-Centric Reasoning, Motion & Scene Understanding, and High-Level Reasoning. Highlighted rows denote MJ variants. Per-task results in Tables 5, 6, 7, 8, 9, 10, 11.

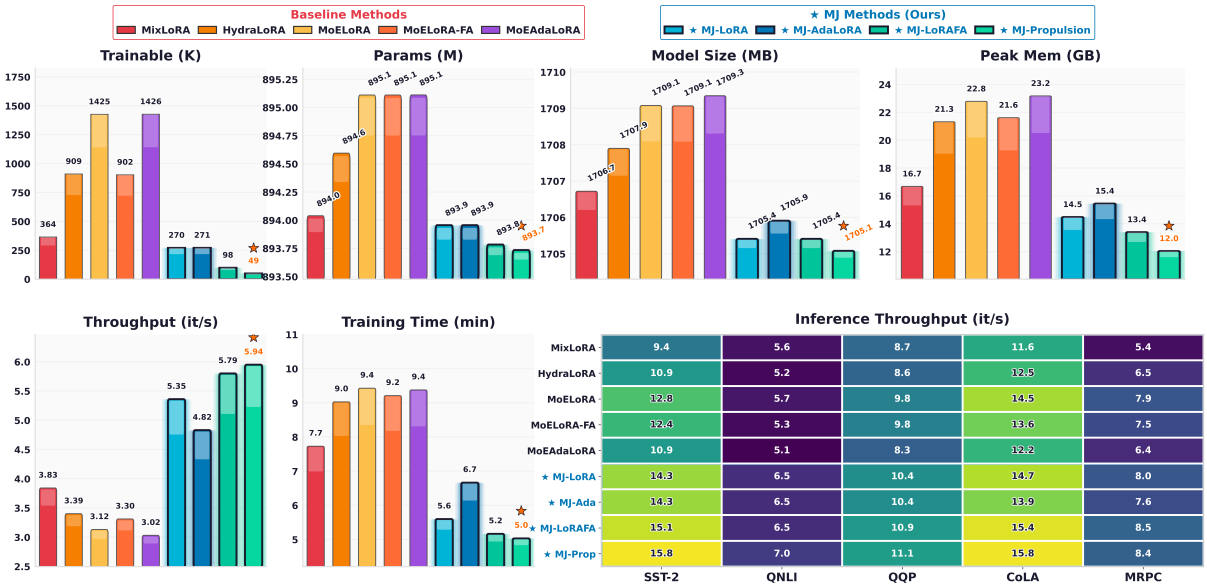

Figure 2: Efficiency comparison. **Top row:** Trainable parameters (K), total parameters (M), model size (MB), and peak GPU memory (GB). **Bottom row:** Training throughput (it/s = iterations per second), training time (min), and inference throughput across GLUE tasks.

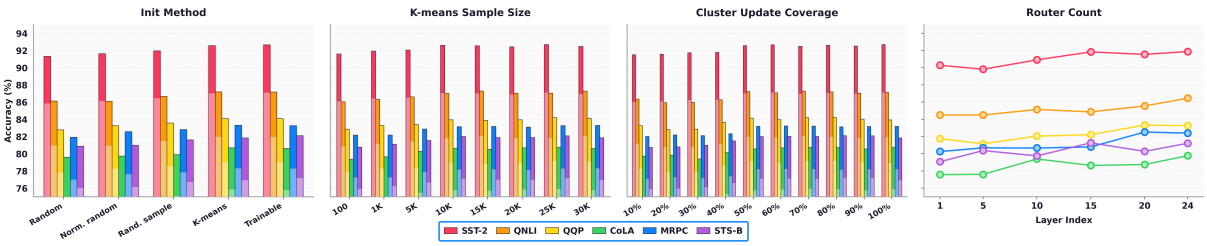

Figure 3: (a) K-means initialization matches trainable routers. (b) More samples improve initialization, saturating at 5K–10K. (c) EMA update coverage of 50–70% suffices. (d) More routing layers improve performance.

continuous updates to track the evolving token distribution.

Router count. Figure 3(d) varies the number of Transformer blocks equipped with MJ routing (1 to 24), with remaining blocks using standard PEFT. Performance improves as more blocks use routing, with 20–24 blocks achieving the best results.

Routing granularity. Table 3 compares three routing strategies using Llama-3-8B-Instruct on QA benchmarks. **Token-wise** and **sentence-wise**

Method	Granularity	BoolQ	PIQA	SIQA	H.Sw.	W.Gra	ARC-e	ARC-c	OBQA
MJLoRA	Token	$71.11_{\pm 0.67}$	$73.73_{\pm 0.53}$	$64.73_{\pm 1.41}$	$51.68_{\pm 2.20}$	$77.93_{\pm 0.30}$	$85.13_{\pm 0.66}$	$71.42_{\pm 1.19}$	$78.14_{\pm 0.89}$
	Sentence	$71.00_{\pm 0.88}$	$73.24_{\pm 0.74}$	$65.22_{\pm 1.69}$	$51.22_{\pm 2.52}$	$77.27_{\pm 0.61}$	$84.73_{\pm 0.93}$	$71.45_{\pm 1.36}$	$77.42_{\pm 1.18}$
	Task	$71.60_{\pm 0.41}$	$74.03_{\pm 0.65}$	$\mathbf{65.70}_{\pm 1.29}$	$51.50_{\pm 2.31}$	$78.90_{\pm 0.48}$	$86.19_{\pm 0.72}$	$71.48_{\pm 1.27}$	$77.89_{\pm 1.06}$
MJPropulsion	Token	$71.76_{\pm 0.65}$	$74.32_{\pm 0.53}$	$64.02_{\pm 1.41}$	$50.52_{\pm 2.33}$	$77.18_{\pm 0.18}$	$85.03_{\pm 0.80}$	$70.91_{\pm 1.53}$	$77.46_{\pm 0.92}$
	Sentence	$71.20_{\pm 0.91}$	$74.53_{\pm 0.88}$	$63.73_{\pm 1.72}$	$50.49_{\pm 2.66}$	$77.34_{\pm 0.47}$	$84.92_{\pm 1.04}$	$71.10_{\pm 1.79}$	$77.39_{\pm 1.21}$
	Task	$72.26_{\pm 0.37}$	$\mathbf{74.62}_{\pm 0.79}$	$64.47_{\pm 1.31}$	$51.03_{\pm 2.42}$	$77.50_{\pm 0.52}$	$84.95_{\pm 0.65}$	$71.33_{\pm 1.33}$	$77.99_{\pm 1.14}$
MJAdaLoRA	Token	$71.52_{\pm 0.22}$	$73.14_{\pm 0.58}$	$64.15_{\pm 1.51}$	$50.12_{\pm 2.55}$	$78.79_{\pm 0.31}$	$85.08_{\pm 0.45}$	$71.98_{\pm 1.67}$	$76.91_{\pm 0.92}$
	Sentence	$71.11_{\pm 0.63}$	$73.67_{\pm 0.79}$	$64.17_{\pm 1.83}$	$50.31_{\pm 2.71}$	$79.37_{\pm 0.56}$	$84.71_{\pm 0.88}$	$71.83_{\pm 1.92}$	$79.57_{\pm 1.24}$
	Task	$\mathbf{72.84}_{\pm 0.29}$	$73.74_{\pm 0.74}$	$65.26_{\pm 1.37}$	$49.94_{\pm 2.51}$	$\mathbf{79.73}_{\pm 0.61}$	$\mathbf{86.41}_{\pm 0.67}$	$72.64_{\pm 1.39}$	$\mathbf{79.73}_{\pm 1.15}$
MJLoRAFA	Token	$71.38_{\pm 0.27}$	$74.04_{\pm 0.62}$	$64.94_{\pm 1.46}$	$51.55_{\pm 2.64}$	$78.51_{\pm 0.35}$	$85.86_{\pm 0.61}$	$71.39_{\pm 1.58}$	$77.92_{\pm 0.90}$
	Sentence	$71.34_{\pm 0.71}$	$73.62_{\pm 0.91}$	$64.82_{\pm 1.77}$	$51.70_{\pm 2.85}$	$78.20_{\pm 0.68}$	$84.87_{\pm 0.94}$	$\mathbf{72.94}_{\pm 1.96}$	$77.77_{\pm 1.19}$
	Task	$71.69_{\pm 0.34}$	$74.46_{\pm 0.76}$	$64.64_{\pm 1.40}$	$\mathbf{52.50}_{\pm 2.55}$	$78.91_{\pm 0.59}$	$86.13_{\pm 0.71}$	$71.70_{\pm 1.42}$	$77.70_{\pm 1.08}$

Table 3: **Token/Sentence**: unsupervised routing via representation similarity. **Task**: supervised routing using dataset ID (oracle). For task-specific routing, ARC-e and ARC-c share one expert (7 experts, 8 datasets).

routing are unsupervised (representation-based), while **task-specific** routing uses known dataset IDs as an oracle. With 7 experts and 8 datasets, ARC-e and ARC-c share an expert (science/factual knowledge), while each remaining dataset receives a dedicated expert. We follow the same experimental setting as Appendix H.

Task-specific routing performs best on average because each group receives a dedicated expert that specializes without interference. Token-wise routing closely matches the oracle with an average gap of only 0.53% across the four MJ variants, despite having no task labels—and even outperforms task routing on individual benchmarks (e.g., MJLoRAFA on SIQA, MJAdaLoRA on H.Sw.). Sentence-wise routing is generally weaker than token-wise routing because a single routing decision cannot capture intra-sequence variation—different tokens (e.g., question vs answer) may benefit from different adapters—though the gap is small and sentence routing wins on some configurations (e.g., MJAdaLoRA OBQA and W.Gra). Overall, this shows that task-relevant structure emerges naturally in token representations, and MJ discovers it through clustering alone.

Finding: *Unsupervised token-wise routing achieves near-oracle performance, demonstrating that MJ learns meaningful specialization from representations without task supervision.*

Extended ablations in Appendix D: **(i)** Similarity function (§D.1); **(ii)** Routing temperature τ (§D.2); **(iii)** EMA smoothing β (§D.3); **(iv)** Update schedule (§D.4); **(v)** Projection specialization (§D.5); **(vi)** Linear probing validation (§D.6); **(vii)** Expert permutation (§D.7); **(viii)** Shared expert (§D.8); **(ix)** Rank sensitivity (§D.9); **(x)** Expert combinations (§D.10); **(xi)** Self-balancing (§D.12); **(xii)** K-means impact (§D.11); **(xiii)** Complexity and Parameter Analysis (§D.13); **(xiv)** Layer-wise visualization (§D.14).

6 Related Work

We provide extended discussion of related work in Appendix F; here we summarize the key connections.

PEFT methods adapt frozen LLMs using lightweight modules such as low-rank adapters (Hu et al., 2022; Houlsby et al., 2019; Zaken et al., 2022; Lester et al., 2021), but apply the same adapters uniformly to all inputs. MoE-PEFT methods (Dou et al., 2024; Luo et al., 2024; Li et al., 2024b; Gou et al., 2023; Liao et al., 2025) introduce learned routing for input-dependent specialization, but add trainable router parameters and memory overhead from multi-expert evaluation. MJ bridges these approaches: it achieves MoE-style specialization while preserving the exact parameter budget of standard PEFT through gradient-free clustering-based routing, avoiding the overhead of learned routers and multi-expert execution.

7 Conclusion

We presented Monkey Jump, a method that achieves MoE-style specialization in PEFT without adding extra trainable parameters for experts and routing. By treating existing adapters as implicit experts and routing tokens via gradient-free clustering, MJ achieves comparable accuracy to MoE-PEFT while using 7–29× fewer trainable parameters, up to 48% lower memory, and 1.5–2× faster training and inference. Across 47 benchmarks spanning text, image, and video tasks, MJ consistently improves over standard PEFT methods. MJ is architecture-agnostic—we demonstrated gains with LoRA, LoRA-FA, AdaLoRA, and Propulsion, showing that any adapter-based PEFT method can benefit from gradient-free routing.

8 Limitations

While Monkey Jump achieves strong results, several limitations remain:

Fixed expert capacity. MJ treats each projection adapter as an implicit expert, so the number of experts is fixed at the number of projections (typically 7). This limits specialization compared to standard MoE, which can scale to hundreds of experts. Increasing expert capacity would require adding multiple adapters per projection, reintroducing the parameter overhead MJ is designed to avoid. This trade-off is intentional: MJ prioritizes parameter efficiency over expert scalability.

Clustering assumptions. MJ assumes that token representations naturally cluster in meaningful ways that correspond to different adaptation needs. While this holds for the benchmarks we tested, highly complex or heterogeneous data distributions may benefit from learned routing that can capture more nuanced patterns beyond what k-means clustering provides.

Initialization overhead. MJ requires a k-means initialization step before training, involving a forward pass through a data subset. While this adds only a few minutes of overhead, it introduces an extra pipeline stage compared to standard PEFT methods.

Hyperparameter sensitivity. Although our ablations show MJ is robust to most hyperparameters (§D), performance depends on choices like top-k value, EMA momentum β, and update schedule. Suboptimal settings can degrade performance, particularly on small datasets.

References

Amit Agarwal, Srikant Panda, Angeline Charles, Bhargava Kumar, Hitesh Patel, Priyaranjan Pattnayak, Taki Hasan Rafi, Tejaswini Kumar, and Dong-Kyu Chae. 2025. Mvtamperbench: Evaluating robustness of vision-language models. *Preprint*, arXiv:2412.19794.

David Arthur and Sergei Vassilvitskii. 2006. k-means++: The advantages of careful seeding. Technical report, Stanford.

Malyaban Bal and Abhronil Sengupta. 2025. Grasp: Grouped activation shared parameterization for parameter-efficient fine-tuning and robust inference of transformers. *arXiv preprint arXiv:2512.04296*.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, and 1 others. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, 05, pages 7432–7439.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Zhaowei Cai, Avinash Ravichandran, Subhransu Maji, Charless Fowlkes, Zhuowen Tu, and Stefano Soatto. 2021. Exponential moving average normalization for self-supervised and semi-supervised learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 194–203.

Jie Cao, Tianwei Lin, Hongyang He, Rolan Yan, Wenqiao Zhang, Juncheng Li, Dongping Zhang, Siliang Tang, and Yueting Zhuang. 2025. Moa: Heterogeneous mixture of adapters for parameter-efficient fine-tuning of large language models. *arXiv preprint arXiv:2506.05928*.

Shaoxiang Chen, Zequn Jie, and Lin Ma. 2024. Llava-mole: Sparse mixture of lora experts for mitigating data conflicts in instruction finetuning mllms. *arXiv preprint arXiv:2401.16160*.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Yu Wu, and 1 others. 2024. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186.

Shihan Dou, Enyu Zhou, Yan Liu, Songyang Gao, Wei Shen, Limao Xiong, Yuhao Zhou, Xiao Wang, Zhiheng Xi, Xiaoran Fan, and 1 others. 2024. Loramoe: Alleviating world knowledge forgetting in large language models via moe-style plugin. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1932–1945.

Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, and 1 others. 2022. Glam: Efficient scaling of language models with mixture-of-experts. In *International conference on machine learning*, pages 5547–5569. PMLR.

Chengyang Fang, Jiangnan Li, Liang Li, Can Ma, and Dayong Hu. 2023. Separate and locate: Rethink the text in text-based visual question answering. In *Proceedings of the 31st ACM International Conference on Multimedia*, pages 4378–4388.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39.

Chongyang Gao, Kezhen Chen, Jinmeng Rao, Baochen Sun, Ruibo Liu, Daiyi Peng, Yawen Zhang, Xiaoyuan Guo, Jie Yang, and VS Subrahmanian. 2024. Higher layers need more lora experts. *arXiv preprint arXiv:2402.08562*.

Yunhao Gou, Zhili Liu, Kai Chen, Lanqing Hong, Hang Xu, Aoxue Li, Dit-Yan Yeung, James T Kwok, and Yu Zhang. 2023. Mixture of cluster-conditional lora experts for vision-language instruction tuning. *arXiv preprint arXiv:2312.12379*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Yongxin Guo, Zhenglin Cheng, Xiaoying Tang, Zhaopeng Tu, and Tao Lin. 2024. Dynamic mixture of experts: An auto-tuning approach for efficient transformer models. *arXiv preprint arXiv:2405.14297*.

Danna Gurari, Qing Li, Abigale J Stangl, Anhong Guo, Chi Lin, Kristen Grauman, Jiebo Luo, and Jeffrey P Bigham. 2018. Vizwiz grand challenge: Answering visual questions from blind people. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3608–3617.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. 2024. Lora+: Efficient low rank adaptation of large models. *arXiv preprint arXiv:2402.12354*.

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.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Lee. 2023. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 5254–5276.

Quzhe Huang, Zhenwei An, Nan Zhuang, Mingxu Tao, Chen Zhang, Yang Jin, Kun Xu, Liwei Chen, Songfang Huang, and Yansong Feng. 2024. Harder tasks need more experts: Dynamic routing in moe models. *arXiv preprint arXiv:2403.07652*.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. 1991. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, and 1 others. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

Dawid J Kopiczko, Tijmen Blankevoort, and Yuki M Asano. 2023. Vera: Vector-based random matrix adaptation. *arXiv preprint arXiv:2310.11454*.

Md Kowsher, Nusrat Jahan Prottasha, and Prakash Bhat. 2025. Propulsion: Steering llm with tiny fine-tuning. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 7569–7597.

Jason J Lau, Soumya Gayen, Asma Ben Abacha, and Dina Demner-Fushman. 2018. A dataset of clinically generated visual questions and answers about radiology images. *Scientific data*, 5(1):1–10.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Bo Li, Yuanhan Zhang, Dong Guo, Renrui Zhang, Feng Li, Hao Zhang, Kaichen Zhang, Peiyuan Zhang, Yanwei Li, Ziwei Liu, and 1 others. 2024a. Llava-onevision: Easy visual task transfer. *arXiv preprint arXiv:2408.03326*.

Bohao Li, Rui Wang, Guangzhi Wang, Yuying Ge, Yixiao Ge, and Ying Shan. 2023. Seed-bench: Benchmarking multimodal llms with generative comprehension. *arXiv preprint arXiv:2307.16125*.

Dengchun Li, Yingzi Ma, Naizheng Wang, Zhengmao Ye, Zhiyuan Cheng, Yinghao Tang, Yan Zhang, Lei Duan, Jie Zuo, Cal Yang, and 1 others. 2024b. Mixlora: Enhancing large language models fine-tuning with lora-based mixture of experts. *arXiv preprint arXiv:2404.15159*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

Mengqi Liao, Wei Chen, Junfeng Shen, Shengnan Guo, and Huaiyu Wan. 2025. Hmora: Making llms more effective with hierarchical mixture of lora experts. In *The Thirteenth International Conference on Learning Representations*.

Pan Lu, Swaroop Mishra, Tanglin Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. 2022. Learn to explain: Multimodal reasoning via thought chains for science question answering. *Advances in Neural Information Processing Systems*, 35:2507–2521.

Tongxu Luo, Jiahe Lei, Fangyu Lei, Weihao Liu, Shizhu He, Jun Zhao, and Kang Liu. 2024. Moelora: Contrastive learning guided mixture of experts on parameter-efficient fine-tuning for large language models. *arXiv preprint arXiv:2402.12851*.

Dabiao Ma, Ziming Dai, Zhimin Xin, Shu Wang, Ye Wang, and Haojun Fei. 2025. Ts-peft: Token-selective parameter-efficient fine-tuning with learnable threshold gating. *arXiv preprint arXiv:2511.16147*.

Kenneth Marino, Mohammad Rastegari, Ali Farhadi, and Roozbeh Mottaghi. 2019. Ok-vqa: A visual question answering benchmark requiring external knowledge. In *Proceedings of the IEEE/cvf conference on computer vision and pattern recognition*, pages 3195–3204.

Ahmed Masry, Xuan Long Do, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. 2022. Chartqa: A benchmark for question answering about charts with visual and logical reasoning. In *Findings of the association for computational linguistics: ACL 2022*, pages 2263–2279.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, and 1 others. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Nusrat Jahan Prottasha, Upama Roy Chowdhury, Shetu Mohanto, Tasfia Nuzhat, Abdullah As Sami, Md Shamol Ali, Md Shohanur Islam Sobuj, Hafijur Raman, Md Kowsher, and Ozlem Ozmen Garibay. 2025. Peft a2z: Parameter-efficient fine-tuning survey for large language and vision models. *arXiv preprint arXiv:2504.14117*.

Joan Puigcerver, Carlos Riquelme, Basil Mustafa, and Neil Houlsby. 2023. From sparse to soft mixtures of experts. *arXiv preprint arXiv:2308.00951*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. 2019. Socialiqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.

Chunlin Tian, Zhan Shi, Zhijiang Guo, Li Li, and Cheng-Zhong Xu. 2024. Hydralora: An asymmetric lora architecture for efficient fine-tuning. *Advances in Neural Information Processing Systems*, 37:9565–9584.

Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. 2023. Dylora: Parameter-efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 3274–3287.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP workshop BlackboxNLP: Analyzing and interpreting neural networks for NLP*, pages 353–355.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and 1 others. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Shu Yang, Muhammad Asif Ali, Cheng-Long Wang, Lijie Hu, and Di Wang. 2024. Moral: Moe augmented lora for llms' lifelong learning. *arXiv preprint arXiv:2402.11260*.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In

Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pages 1–9.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

Xiaohua Zhai, Joan Puigcerver, Alexander Kolesnikov, Pierre Ruyssen, Carlos Riquelme, Mario Lucic, Josip Djolonga, Andre Susano Pinto, Maxim Neumann, Alexey Dosovitskiy, and 1 others. 2019. A large-scale study of representation learning with the visual task adaptation benchmark. *arXiv preprint arXiv:1910.04867*.

Dacao Zhang, Kun Zhang, Shimao Chu, Le Wu, Xin Li, and Si Wei. 2025. More: A mixture of low-rank experts for adaptive multi-task learning. *arXiv preprint arXiv:2505.22694*.

Longteng Zhang, Lin Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. 2023a. Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning. *arXiv preprint arXiv:2308.03303*.

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

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M. Dai, Quoc V. Le, James Laudon, and 1 others. 2022. Mixture-of-experts with expert-choice routing. In *Advances in Neural Information Processing Systems*, volume 35, pages 7103–7114.

Yuan Zhuang, Yi Shen, Yuexin Bian, Qing Su, Shihao Ji, Yuanyuan Shi, and Fei Miao. 2025. Ld-mole: Learnable dynamic routing for mixture of lora experts. *arXiv preprint arXiv:2509.25684*.

Contents

1 Introduction **1**

2 Preliminaries **3**

3 Monkey Jump **3**
- 3.1 Routing . . . 3
- 3.2 Center Initialization and Online Updates . . . 4
- 3.3 Routing Variants . . . 4
- 3.4 Computational Cost . . . 4

4 Theoretical Analysis **4**

5 Experiments **5**
- 5.1 Efficiency Analysis . . . 6
- 5.2 Ablation Study . . . 6

6 Related Work **8**

7 Conclusion **8**

8 Limitations **9**

A Proof of Theorem 1 **15**

B Extension to Soft Top-k Routing **17**

C Proof of Theorem 2: Information Maximality **18**
- C.1 Definitions . . . 18
- C.2 Assumptions . . . 18
- C.3 Main Result . . . 18

D Extended Ablation Studies **23**
- D.1 Similarity Function . . . 23
- D.2 Routing Temperature . . . 23
- D.3 EMA Smoothing Factor . . . 23
- D.4 Update Schedule . . . 23
- D.5 Projection Specialization . . . 23
- D.6 Linear Probing for Last-Token Routing . . . 24
- D.7 Expert Permutation Analysis . . . 25
- D.8 Shared Expert Selection . . . 25
- D.9 Rank Sensitivity . . . 25
- D.10 Expert Combination Analysis . . . 26
- D.11 Impact of K-means Initialization . . . 27
- D.12 Expert Usage and Self-Balancing . . . 29
- D.13 Complexity and Parameter Analysis . . . 29
- D.14 Layer-wise Cluster Visualization . . . 30

E Detailed Results **32**
- E.1 Statistical Significance Analysis . . . 36

F Extended Related Work **37**
F.1 Parameter-Efficient Fine-Tuning 37
F.2 Sparse Mixture-of-Experts 38
F.3 MoE-Enhanced PEFT 38
F.4 Positioning of Monkey Jump 38

G Datasets **39**

H Implementation Details **41**

I Use of AI Assistants **43**

A Proof of Theorem 1

We analyze a single Transformer block with E adapter projections and drop layer indices for clarity. Let $H = [h_1, \dots, h_T] \in \mathbb{R}^{d \times T}$ denote the input token representations. Each projection $e \in \{1, \dots, E\}$ has a frozen weight $\mathbf{W_e}$ and a trainable adapter $\mathbf{\Delta W_e}$. We assume hard routing (top-1) throughout: each token is assigned to exactly one expert.

Shared PEFT applies all adapters uniformly:

$$U^{\text{PEFT}} = \sum_{e=1}^{E} \mathbf{\Delta W_e} H = \left(\sum_{e=1}^{E} \mathbf{\Delta W_e} \right) H.$$

Monkey Jump uses token-specific routing:

$$U^{\text{MJ}} = \sum_{e=1}^{E} \mathbf{\Delta W_e} H D_e,$$
$$D_e = \text{diag}(m_{1,e}, \dots, m_{T,e}).$$

Let $\mathcal{E}_t = \{e : m_{t,e} > 0\}$ denote the set of active experts for token t, and define the activated expert set as $\mathcal{A} = \bigcup_{t=1}^{T} \mathcal{E}_t$.

Proof. We begin by defining the relevant column spaces. For each expert e, define

$$\mathcal{C}_e := \text{Col}(\mathbf{\Delta W_e}) = \{\mathbf{\Delta W_e} x : x \in \mathbb{R}^d\},$$

which is the set of all output vectors that adapter e can produce. By definition, $\dim(\mathcal{C}_e) = \text{rank}(\mathbf{\Delta W_e}) = r_e$. The sum of column spaces is

$$\mathcal{C}_{\text{all}} := \sum_{e=1}^{E} \mathcal{C}_e,$$

which contains every vector of the form $v_1 + \cdots + v_E$ with $v_e \in \mathcal{C}_e$.

We first establish the upper bound for Shared PEFT. The output is

$$U^{\text{PEFT}} = \left(\sum_{e=1}^{E} \mathbf{\Delta W_e} \right) H.$$

Since $\text{Col}(AB) \subseteq \text{Col}(A)$ for any matrix product,

$$\text{Col}(U^{\text{PEFT}}) \subseteq \text{Col}\left(\sum_{e=1}^{E} \mathbf{\Delta W_e} \right).$$

Any column of $\sum_e \mathbf{\Delta W_e}$ is the sum of corresponding columns from each $\mathbf{\Delta W_e}$, each lying in $\mathcal{C}_e$, so

$$\text{Col}\left(\sum_{e=1}^{E} \mathbf{\Delta W_e} \right) \subseteq \sum_{e=1}^{E} \mathcal{C}_e = \mathcal{C}_{\text{all}}.$$

This inclusion can be strict: directions in individual column spaces may cancel in the matrix sum. Combining these,

$$\text{rank}(U^{\text{PEFT}}) \leq \text{rank}\left(\sum_{e=1}^{E} \mathbf{\Delta W_e} \right) \leq \dim(\mathcal{C}_{\text{all}}).$$

Next, we derive the upper bound for Monkey Jump. The output is

$$U^{\text{MJ}} = \sum_{e=1}^{E} \mathbf{\Delta W_e} H D_e.$$

The t-th column is $u_t^{\text{MJ}} = \sum_{e=1}^{E} m_{t,e} \mathbf{\Delta W_e} h_t$. Each nonzero term $m_{t,e} \mathbf{\Delta W_e} h_t$ lies in $\mathcal{C}_e$, so

$$u_t^{\text{MJ}} \in \sum_{e \in \mathcal{E}_t} \mathcal{C}_e \subseteq \sum_{e \in \mathcal{A}} \mathcal{C}_e \subseteq \mathcal{C}_{\text{all}},$$

where $\mathcal{E}_t = \{e : m_{t,e} > 0\}$ and $\mathcal{A} = \bigcup_t \mathcal{E}_t$ is the set of all activated experts. Therefore,

$$\text{Col}(U^{\text{MJ}}) \subseteq \sum_{e \in \mathcal{A}} \mathcal{C}_e,$$

giving

$$\text{rank}(U^{\text{MJ}}) \leq \dim\left(\sum_{e \in \mathcal{A}} \mathcal{C}_e \right) \leq \dim(\mathcal{C}_{\text{all}}).$$

We now show that Monkey Jump can achieve this upper bound under favorable conditions. Suppose the routing activates all experts, i.e., $\mathcal{A} = \{1, \dots, E\}$. For each expert e, let $\mathcal{T}_e = \{t : e \in \mathcal{E}_t\}$ be the set of tokens routed to expert e, and let H_e be the corresponding submatrix of H.

Under hard routing, each token activates exactly one expert, so $\{\mathcal{T}_e\}$ partitions $\{1, \dots, T\}$. The output becomes

$$U^{\text{MJ}} = \left[\mathbf{\Delta W_1} H_1 \;\; \mathbf{\Delta W_2} H_2 \;\; \cdots \;\; \mathbf{\Delta W_E} H_E \right].$$

The column space of a horizontal concatenation equals the sum of column spaces:

$$\text{Col}(U^{\text{MJ}}) = \sum_{e=1}^{E} \text{Col}(\mathbf{\Delta W_e} H_e).$$

If the inputs routed to each expert span the row space of that adapter, then $\text{Col}(\mathbf{\Delta W_e} H_e) = \mathcal{C}_e$, and

$$\text{Col}(U^{\text{MJ}}) = \sum_{e=1}^{E} \mathcal{C}_e = \mathcal{C}_{\text{all}}.$$

Thus $\mathrm{rank}(U^{\mathrm{MJ}}) = \dim(\mathcal{C}_{\mathrm{all}})$, achieving the upper bound.

Combining these results establishes the theorem. Under full activation with diverse inputs, $\mathrm{rank}(U^{\mathrm{MJ}}) = \dim(\mathcal{C}_{\mathrm{all}})$ while $\mathrm{rank}(U^{\mathrm{PEFT}}) \leq \dim(\mathcal{C}_{\mathrm{all}})$. Therefore $\mathrm{rank}(U^{\mathrm{MJ}}) \geq \mathrm{rank}(U^{\mathrm{PEFT}})$.

The inequality is strict when

$$\mathrm{Col}\left(\sum_{e=1}^{E} \mathbf{\Delta W_e}\right) \subsetneq \mathcal{C}_{\mathrm{all}}.$$

This occurs when directions in individual column spaces cancel in the matrix sum.

As a concrete example, consider $E = 2$ with

$$\mathbf{\Delta W_1} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \qquad \mathbf{\Delta W_2} = \begin{bmatrix} 0 & 0 \\ -1 & 0 \end{bmatrix}.$$

Then $\mathcal{C}_1 = \mathrm{span}\{e_1\}$, $\mathcal{C}_2 = \mathrm{span}\{e_2\}$, and $\mathcal{C}_{\mathrm{all}} = \mathbb{R}^2$. The matrix sum is

$$\mathbf{\Delta W_1} + \mathbf{\Delta W_2} = \begin{bmatrix} 1 & 0 \\ -1 & 0 \end{bmatrix},$$

with $\mathrm{Col}(\mathbf{\Delta W_1} + \mathbf{\Delta W_2}) = \mathrm{span}\{(1, -1)^\top\}$, a one-dimensional subspace strictly contained in $\mathcal{C}_{\mathrm{all}} = \mathbb{R}^2$.

With $H = [e_1 \; e_1]$ and hard routing (token 1 to expert 1, token 2 to expert 2):

$$U^{\mathrm{MJ}} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \qquad U^{\mathrm{PEFT}} = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}.$$

Thus $\mathrm{rank}(U^{\mathrm{MJ}}) = 2 > 1 = \mathrm{rank}(U^{\mathrm{PEFT}})$. □

Remark (Routing coverage). The theorem requires that all experts are activated. In practice, Monkey Jump uses k-means clustering with EMA updates, which encourages but does not guarantee full coverage. If some experts are never activated ($\mathcal{A} \subsetneq \{1, \ldots, E\}$), then Monkey Jump's achievable rank is limited to $\dim(\sum_{e \in \mathcal{A}} \mathcal{C}_e)$, which may be smaller than what Shared PEFT achieves. This motivates the use of load balancing or auxiliary losses to ensure diverse routing.

B Extension to Soft Top-k Routing

The main theorem assumes hard routing (top-1) for simplicity. Here we extend the analysis to soft top-k routing, where each token activates up to k experts with weighted coefficients.

Under soft top-k routing, each token t selects k experts $\mathcal{E}_t = \mathrm{TopK}(\{p_{t,e}\}, k)$ and assigns weights $m_{t,e} = p_{t,e} \cdot \mathbb{I}[e \in \mathcal{E}_t]$. The effective adapter for token t is

$$\mathbf{\Delta W_t^{eff}} = \sum_{e \in \mathcal{E}_t} m_{t,e} \mathbf{\Delta W_e},$$

and the Monkey Jump output column for token t is

$$u_t^{\mathrm{MJ}} = \mathbf{\Delta W_t^{eff}} h_t = \sum_{e \in \mathcal{E}_t} m_{t,e} \mathbf{\Delta W_e} h_t.$$

Each output u_t^{MJ} lies in $\sum_{e \in \mathcal{E}_t} \mathcal{C}_e$, since it is a linear combination of vectors from the selected adapters' column spaces. The full output satisfies

$$\mathrm{Col}(U^{\mathrm{MJ}}) \subseteq \sum_{e \in \mathcal{A}} \mathcal{C}_e,$$

where $\mathcal{A} = \bigcup_t \mathcal{E}_t$ is the set of all activated experts. This upper bound is identical to the hard routing case.

The key difference from hard routing is that the output U^{MJ} is no longer a simple concatenation of per-expert blocks. Instead, each column u_t^{MJ} is a weighted combination of up to k adapter outputs.

Let $\mathcal{P} = \{\mathcal{E}_t : t = 1, \ldots, T\}$ denote the set of distinct routing patterns. For each pattern $P \in \mathcal{P}$, define the effective adapter

$$\mathbf{\Delta W_P} = \sum_{e \in P} \bar{m}_e \mathbf{\Delta W_e},$$

where $\bar{m}_e$ represents the average routing weight for expert e within pattern P. The achievable rank depends on how many distinct effective adapters arise from different routing patterns.

Proposition 1 (Soft Routing Expressivity)**.** *Under soft top-k routing:*

(i) The upper bound remains $\mathrm{rank}(U^{\mathrm{MJ}}) \leq \dim\left(\sum_{e \in \mathcal{A}} \mathcal{C}_e\right)$.

(ii) If the routing patterns are diverse (different tokens select different expert subsets), Monkey Jump can still achieve higher rank than Shared PEFT.

(iii) The maximum achievable rank is

$$\mathrm{rank}(U^{\mathrm{MJ}}) \leq \min\Big(|\mathcal{P}| \cdot \max_P \mathrm{rank}(\Delta W_P), \dim(\mathcal{C}_{\mathrm{all}})\Big).$$

Hard routing ($k = 1$) maximizes the diversity of effective adapters: each token uses a pure adapter $\mathbf{\Delta W_e}$ rather than a blend. This makes achieving the full rank $\dim(\mathcal{C}_{\mathrm{all}})$ straightforward.

Soft routing ($k > 1$) introduces blending, which can reduce diversity. However, if the routing patterns are sufficiently varied and the blending coefficients differ across tokens, soft routing can still achieve high expressivity. In practice, the temperature parameter τ controls the sharpness of routing: lower τ yields sharper (more hard-like) routing, while higher τ yields softer blending.

The theoretical analysis suggests three practical guidelines:

- **Lower k increases expressivity:** Fewer experts per token means more distinct routing patterns, closer to the hard routing ideal.
- **Lower τ sharpens routing:** Concentrating probability mass on fewer experts mimics hard routing benefits.
- **Diverse routing is key:** The expressivity advantage of Monkey Jump depends on tokens being routed to different experts, which is encouraged by the k-means clustering and EMA updates.

C Proof of Theorem 2: Information Maximality

We establish that in causal Transformers, the last token's hidden representation is theoretically optimal for sequence-wise routing decisions.

C.1 Definitions

Definition 1 (Entropy)**.** For a discrete random variable X with probability mass function $p(x)$:

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x)$$

Definition 2 (Conditional Entropy)**.** For random variables X and Y:

$$H(Y \mid X) = -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log p(y \mid x) \quad (7)$$

Definition 3 (Mutual Information)**.** For random variables X and Y:

$$\begin{aligned} I(X; Y) &= H(X) - H(X \mid Y) \\ &= H(Y) - H(Y \mid X) \end{aligned} \quad (8)$$

Definition 4 (Conditional Mutual Information)**.** For random variables X, Y, and Z:

$$I(X; Y \mid Z) = H(X \mid Z) - H(X \mid Y, Z)$$

Definition 5 (Kullback-Leibler Divergence)**.** For two probability distributions P and Q over the same sample space $\mathcal{X}$:

$$D_{\mathrm{KL}}(P \| Q) = \sum_{x \in \mathcal{X}} P(x) \log \frac{P(x)}{Q(x)}$$

Definition 6 (Causal Hidden Representation)**.** In a causal Transformer, the hidden representation h_t at position t is a deterministic function of the prefix tokens:

$$h_t = f_t(X_{1:t}) = f_t(x_1, x_2, \ldots, x_t)$$

where f_t is determined by the model architecture and parameters.

C.2 Assumptions

Assumption 1 (Information Preservation Property)**.** *A causal Transformer satisfies the* Information Preservation Property *if the information loss* $\epsilon_t := H(X_{1:t}) - I(h_t; X_{1:t})$ *satisfies:*

$$\epsilon_{t+1} - \epsilon_t \leq H(x_{t+1} \mid X_{1:t}) \quad \textit{for all } t < T.$$

This holds when additional context tokens contribute more information than is lost through the representation bottleneck.

Assumption 2 (Attention Aggregation Property)**.** *A causal Transformer satisfies the* Attention Aggregation Property *if the final hidden state captures the information from all positions:*

$$I(h_1, \ldots, h_{T-1}; X \mid h_T) = 0.$$

This is satisfied when attention weights $\alpha_{T,s}^{(\ell)} > 0$ *for all* $s \in \{1, \ldots, T\}$ *across layers and the model has sufficient capacity.*

C.3 Main Result

Theorem 3 (Information Maximality)**.** *Let* $X = (x_1, \ldots, x_T)$ *be an input sequence and* $h_1, \ldots, h_T$ *be the hidden representations at any layer of a causal Transformer. Then:*

(i) Context Monotonicity: $I(X_{1:t}; X) \leq I(X_{1:t+1}; X)$ *for all* $t < T$.

(ii) Representation Monotonicity: *Under Assumption 1,* $I(h_t; X) \leq I(h_{t+1}; X)$ *for all* $t < T$.

(iii) Dominance over pooling: *Under Assumption 2,*

$$I(h_T; X) \geq I(\bar{h}; X), \quad \textit{where } \bar{h} = \frac{1}{T} \sum_{t=1}^{T} h_t.$$

Proof. We prove all three parts by analyzing the information-theoretic properties of causal hidden representations.

Proof of part (i). We establish monotonicity for the cumulative token contexts $X_{1:t} = (x_1, \ldots, x_t)$.

Since $X_{1:t}$ is a sub-tuple of $X = (x_1, \ldots, x_T)$, knowing X completely determines $X_{1:t}$. Formally, for any realization $x = (x_1, \ldots, x_T)$ of X, there is exactly one corresponding realization $x_{1:t} = (x_1, \ldots, x_t)$ of $X_{1:t}$.

This means the conditional probability satisfies:

$$p(x_{1:t} \mid x) = \begin{cases} 1 & \text{if } x_{1:t} = (x_1, \ldots, x_t) \\ 0 & \text{otherwise} \end{cases}$$

By Definition 2, the conditional entropy is:

$$\begin{aligned} &H(X_{1:t} \mid X) \\ &= -\sum_{x \in \mathcal{X}} \sum_{x_{1:t} \in \mathcal{X}_{1:t}} p(x, x_{1:t}) \log p(x_{1:t} \mid x) \end{aligned} \quad (9)$$

Since $p(x_{1:t} \mid x) = 1$ when $x_{1:t}$ matches the first t components of x, and $\log 1 = 0$:

$$\begin{aligned} H(X_{1:t} \mid X) &= -\sum_{x \in \mathcal{X}} p(x) \cdot 1 \cdot \log 1 \\ &= -\sum_{x \in \mathcal{X}} p(x) \cdot 0 = 0 \end{aligned} \tag{10}$$

By Definition 3, the mutual information between $X_{1:t}$ and X is:

$$I(X_{1:t}; X) = H(X_{1:t}) - H(X_{1:t} \mid X)$$

Substituting $H(X_{1:t} \mid X) = 0$:

$$I(X_{1:t}; X) = H(X_{1:t}) - 0 = H(X_{1:t}) \tag{11}$$

Similarly, for $X_{1:t+1}$:

$$\begin{aligned} I(X_{1:t+1}; X) &= H(X_{1:t+1}) - H(X_{1:t+1} \mid X) \\ &= H(X_{1:t+1}) - 0 = H(X_{1:t+1}) \end{aligned} \tag{12}$$

We now show that $H(X_{1:t}) \leq H(X_{1:t+1})$.

By the chain rule of entropy, the joint entropy of $(X_{1:t}, x_{t+1})$ can be decomposed as:

$$H(X_{1:t}, x_{t+1}) = H(X_{1:t}) + H(x_{t+1} \mid X_{1:t})$$

Since $X_{1:t+1} = (X_{1:t}, x_{t+1})$ by definition:

$$H(X_{1:t+1}) = H(X_{1:t}) + H(x_{t+1} \mid X_{1:t}) \tag{13}$$

We now show that $H(x_{t+1} \mid X_{1:t}) \geq 0$.

By Definition 2:

$$\begin{aligned} &H(x_{t+1} \mid X_{1:t}) \\ &= -\sum_{x_{1:t} \in \mathcal{X}_{1:t}} \sum_{x_{t+1} \in \mathcal{X}_{t+1}} p(x_{1:t}, x_{t+1}) \\ &\quad \cdot \log p(x_{t+1} \mid x_{1:t}) \end{aligned} \tag{14}$$

Using the chain rule of probability $p(x_{1:t}, x_{t+1}) = p(x_{1:t}) p(x_{t+1} \mid x_{1:t})$:

$$\begin{aligned} &H(x_{t+1} \mid X_{1:t}) \\ &= -\sum_{x_{1:t} \in \mathcal{X}_{1:t}} p(x_{1:t}) \\ &\quad \cdot \sum_{x_{t+1} \in \mathcal{X}_{t+1}} p(x_{t+1} \mid x_{1:t}) \log p(x_{t+1} \mid x_{1:t}) \end{aligned} \tag{15}$$

This can be written as an expectation over $X_{1:t}$:

$$\begin{aligned} &H(x_{t+1} \mid X_{1:t}) \\ &= \sum_{x_{1:t} \in \mathcal{X}_{1:t}} p(x_{1:t}) \cdot H(x_{t+1} \mid X_{1:t} = x_{1:t}) \end{aligned} \tag{16}$$

where the conditional entropy given a specific value $x_{1:t}$ is:

$$\begin{aligned} &H(x_{t+1} \mid X_{1:t} = x_{1:t}) \\ &= -\sum_{x_{t+1} \in \mathcal{X}_{t+1}} p(x_{t+1} \mid x_{1:t}) \log p(x_{t+1} \mid x_{1:t}) \end{aligned} \tag{17}$$

For any probability distribution $p(x_{t+1} \mid x_{1:t})$ over $\mathcal{X}_{t+1}$, the entropy is non-negative:

$$H(x_{t+1} \mid X_{1:t} = x_{1:t}) \geq 0 \quad \text{for all } x_{1:t} \in \mathcal{X}_{1:t}$$

Since $p(x_{1:t}) \geq 0$ for all $x_{1:t}$, the weighted sum is also non-negative:

$$\begin{aligned} &H(x_{t+1} \mid X_{1:t}) \\ &= \sum_{x_{1:t} \in \mathcal{X}_{1:t}} p(x_{1:t}) \cdot H(x_{t+1} \mid X_{1:t} = x_{1:t}) \geq 0 \end{aligned} \tag{18}$$

Returning to equation (13):

$$H(X_{1:t+1}) = H(X_{1:t}) + H(x_{t+1} \mid X_{1:t})$$

Since $H(x_{t+1} \mid X_{1:t}) \geq 0$:

$$\begin{aligned} H(X_{1:t+1}) &= H(X_{1:t}) + H(x_{t+1} \mid X_{1:t}) \\ &\geq H(X_{1:t}) + 0 = H(X_{1:t}) \end{aligned} \tag{19}$$

Therefore:

$$H(X_{1:t+1}) \geq H(X_{1:t})$$

Combining with equations (11) and (12):

$$\begin{aligned} I(X_{1:t+1}; X) &= H(X_{1:t+1}) \\ &\geq H(X_{1:t}) = I(X_{1:t}; X) \end{aligned} \tag{20}$$

This inequality holds for all $t \in \{1, 2, \ldots, T-1\}$.

Applying this result iteratively:

For $t = T - 1$:

$$I(X_{1:T}; X) \geq I(X_{1:T-1}; X)$$

For $t = T - 2$:

$$I(X_{1:T-1}; X) \geq I(X_{1:T-2}; X)$$

Continuing this pattern for $t = T-3, T-4, \ldots, 2, 1$:

$$\begin{aligned} I(X_{1:T-2}; X) &\geq I(X_{1:T-3}; X) \geq \cdots \\ &\geq I(X_{1:2}; X) \geq I(X_{1:1}; X) \end{aligned} \tag{21}$$

Combining all these inequalities by transitivity:

$$I(X_{1:T}; X) \geq I(X_{1:T-1}; X) \geq \cdots \geq I(X_{1:2}; X) \geq I(X_{1:1}; X) \quad (22)$$

This establishes part **(i)**.

Proof of part (ii). We now extend the monotonicity result to the hidden representations h_t under Assumption 1.

In a causal Transformer, the hidden representation h_t at position t is a deterministic function of the prefix tokens $X_{1:t}$:

$$h_t = f_t(X_{1:t})$$

We analyze the mutual information $I(h_t; X)$ by decomposing it using the chain rule.

The full sequence X can be written as the concatenation of $X_{1:t}$ and $X_{t+1:T}$:

$$X = (X_{1:t}, X_{t+1:T})$$

By the chain rule of mutual information:

$$I(h_t; X) = I(h_t; X_{1:t}, X_{t+1:T})$$

Expanding using the chain rule:

$$I(h_t; X_{1:t}, X_{t+1:T}) = I(h_t; X_{1:t}) + I(h_t; X_{t+1:T} \mid X_{1:t}) \quad (23)$$

We now evaluate the conditional mutual information $I(h_t; X_{t+1:T} \mid X_{1:t})$.

By Definition 4:

$$I(h_t; X_{t+1:T} \mid X_{1:t}) = H(h_t \mid X_{1:t}) - H(h_t \mid X_{t+1:T}, X_{1:t}) \quad (24)$$

Since $h_t = f_t(X_{1:t})$ is a deterministic function of $X_{1:t}$, knowing $X_{1:t}$ completely determines h_t. Therefore:

$$H(h_t \mid X_{1:t}) = 0$$

Similarly, since $(X_{t+1:T}, X_{1:t}) = X$ contains $X_{1:t}$ as a component, and h_t is determined by $X_{1:t}$:

$$H(h_t \mid X_{t+1:T}, X_{1:t}) = H(h_t \mid X) = 0$$

Substituting into equation (24):

$$I(h_t; X_{t+1:T} \mid X_{1:t}) = 0 - 0 = 0$$

Returning to equation (23):

$$I(h_t; X) = I(h_t; X_{1:t}) + I(h_t; X_{t+1:T} \mid X_{1:t}) = I(h_t; X_{1:t}) + 0 = I(h_t; X_{1:t}) \quad (25)$$

Therefore, the mutual information between h_t and X equals the mutual information between h_t and its causal context:

$$I(h_t; X) = I(h_t; X_{1:t}) \quad (26)$$

We now establish an upper bound on $I(h_t; X_{1:t})$.

Since $h_t = f_t(X_{1:t})$ is a deterministic function of $X_{1:t}$, the random variables form a Markov chain:

$$X \to X_{1:t} \to h_t$$

This means that h_t is conditionally independent of X given $X_{1:t}$:

$$p(h_t \mid X_{1:t}, X) = p(h_t \mid X_{1:t})$$

By the Data Processing Inequality, for any Markov chain $A \to B \to C$:

$$I(A; C) \leq I(A; B)$$

Applying this to our Markov chain with $A = X$, $B = X_{1:t}$, and $C = h_t$:

$$I(X; h_t) \leq I(X; X_{1:t})$$

Since mutual information is symmetric, $I(X; h_t) = I(h_t; X)$ and $I(X; X_{1:t}) = I(X_{1:t}; X)$:

$$I(h_t; X) \leq I(X_{1:t}; X)$$

From equation (11), we have $I(X_{1:t}; X) = H(X_{1:t})$, so:

$$I(h_t; X) \leq H(X_{1:t})$$

Similarly, for $h_{t+1} = f_{t+1}(X_{1:t+1})$:

$$I(h_{t+1}; X) \leq H(X_{1:t+1})$$

We now establish the monotonicity for hidden representations using Assumption 1.

By Assumption 1, the information loss $\epsilon_t := H(X_{1:t}) - I(h_t; X_{1:t})$ satisfies:

$$\epsilon_{t+1} - \epsilon_t \leq H(x_{t+1} \mid X_{1:t})$$

This can be rewritten as:

$$H(X_{1:t+1}) - I(h_{t+1}; X_{1:t+1}) - H(X_{1:t}) + I(h_t; X_{1:t}) \leq H(x_{t+1} \mid X_{1:t}) \quad (27)$$

Using the chain rule $H(X_{1:t+1}) = H(X_{1:t}) + H(x_{t+1} \mid X_{1:t})$ from equation (13):

$$H(X_{1:t}) + H(x_{t+1} \mid X_{1:t}) - I(h_{t+1}; X_{1:t+1}) - H(X_{1:t}) + I(h_t; X_{1:t}) \leq H(x_{t+1} \mid X_{1:t}) \tag{28}$$

Simplifying:

$$H(x_{t+1} \mid X_{1:t}) - I(h_{t+1}; X_{1:t+1}) + I(h_t; X_{1:t}) \leq H(x_{t+1} \mid X_{1:t}) \tag{29}$$

$$-I(h_{t+1}; X_{1:t+1}) + I(h_t; X_{1:t}) \leq 0$$

$$I(h_t; X_{1:t}) \leq I(h_{t+1}; X_{1:t+1})$$

Combining with equation (26):

$$I(h_t; X) = I(h_t; X_{1:t}) \leq I(h_{t+1}; X_{1:t+1}) = I(h_{t+1}; X) \tag{30}$$

Therefore:

$$I(h_t; X) \leq I(h_{t+1}; X) \quad \text{for all } t < T$$

This establishes part **(ii)**.

Applying part (ii) iteratively:

For $t = T - 1$:

$$I(h_{T-1}; X) \leq I(h_T; X)$$

For $t = T - 2$:

$$I(h_{T-2}; X) \leq I(h_{T-1}; X)$$

Continuing this pattern for $t = T - 3, T - 4, \ldots, 2, 1$:

$$I(h_{T-3}; X) \leq I(h_{T-2}; X), \quad \ldots, \quad I(h_1; X) \leq I(h_2; X) \tag{31}$$

Combining all these inequalities by transitivity:

$$I(h_1; X) \leq I(h_2; X) \leq \cdots \leq I(h_{T-1}; X) \leq I(h_T; X) \tag{32}$$

Therefore, for all $t \leq T$:

$$I(h_T; X) \geq I(h_t; X)$$

Proof of part (iii). We now prove that the last token dominates mean pooling in terms of mutual information with the input, under Assumption 2.

The mean-pooled representation is defined as:

$$\bar{h} = \frac{1}{T} \sum_{t=1}^{T} h_t$$

This is a deterministic function of the tuple $(h_1, h_2, \ldots, h_T)$:

$$\bar{h} = \phi(h_1, h_2, \ldots, h_T)$$

where ϕ is the averaging function.

Since $\bar{h}$ is determined by $(h_1, \ldots, h_T)$, we have the Markov chain:

$$X \to (h_1, h_2, \ldots, h_T) \to \bar{h}$$

By the Data Processing Inequality:

$$I(\bar{h}; X) \leq I(h_1, h_2, \ldots, h_T; X) \tag{33}$$

We now analyze $I(h_1, h_2, \ldots, h_T; X)$.

Each hidden representation $h_t = f_t(X_{1:t})$ is a deterministic function of $X_{1:t}$. The tuple $(h_1, \ldots, h_T)$ is therefore a deterministic function of X:

$$(h_1, h_2, \ldots, h_T) = (f_1(X_{1:1}), f_2(X_{1:2}), \ldots, f_T(X_{1:T})) = F(X) \tag{34}$$

for some function F determined by the model.

This gives us the Markov chain:

$$X \to (h_1, h_2, \ldots, h_T)$$

By the Data Processing Inequality applied in reverse (since $(h_1, \ldots, h_T) = F(X)$ is a deterministic function):

$$I(h_1, h_2, \ldots, h_T; X) \leq H(X)$$

In causal Transformers, the final hidden state h_T has access to all positions through the causal attention mechanism. At each layer ℓ, position T computes:

$$h_T^{(\ell)} = \text{Attention}^{(\ell)}(Q_T^{(\ell)}, K_{1:T}^{(\ell)}, V_{1:T}^{(\ell)}) + h_T^{(\ell-1)} \tag{35}$$

where the attention weights are:

$$\alpha_{T,s}^{(\ell)} = \frac{\exp\left(Q_T^{(\ell)} \cdot K_s^{(\ell)} / \sqrt{d}\right)}{\sum_{s'=1}^{T} \exp\left(Q_T^{(\ell)} \cdot K_{s'}^{(\ell)} / \sqrt{d}\right)} \quad \text{for } s \in \{1, \ldots, T\} \tag{36}$$

By Assumption 2, we have:

$$I(h_1, \ldots, h_{T-1}; X \mid h_T) = 0$$

We can decompose the joint mutual information using the chain rule:

$$\begin{aligned} &I(h_1, \ldots, h_T; X) \\ &= I(h_T; X) + I(h_1, \ldots, h_{T-1}; X \mid h_T) \end{aligned} \tag{37}$$

This follows from the chain rule of mutual information:

$$I(A, B; C) = I(A; C) + I(B; C \mid A)$$

with $A = h_T$, $B = (h_1, \ldots, h_{T-1})$, and $C = X$.

Substituting Assumption 2:

$$\begin{aligned} &I(h_1, \ldots, h_T; X) \\ &= I(h_T; X) + I(h_1, \ldots, h_{T-1}; X \mid h_T) \\ &= I(h_T; X) + 0 = I(h_T; X) \end{aligned} \tag{38}$$

Returning to equation (33):

$$\begin{aligned} I(\bar{h}; X) &\leq I(h_1, h_2, \ldots, h_T; X) \\ &= I(h_T; X) \end{aligned} \tag{39}$$

Therefore:

$$I(h_T; X) \geq I(\bar{h}; X)$$

This establishes part **(iii)** and completes the proof. □

Theorem 3 establishes that in causal Transformers satisfying the Information Preservation Property (Assumption 1), the last token's hidden representation h_T contains the most information about the input sequence X among all single-position representations. Furthermore, under the Attention Aggregation Property (Assumption 2), h_T contains at least as much information as mean pooling. These properties provide theoretical justification for using the last token for sequence-wise routing decisions, as it maximizes the information available for determining expert assignment.

D Extended Ablation Studies

We provide additional ablation studies to analyze MJ's sensitivity to hyperparameters and design choices.

D.1 Similarity Function

Figure 4(a) compares four similarity functions for computing routing scores between token representations and cluster centers: cosine similarity, dot product, Euclidean distance, and L1 distance.

Cosine similarity achieves the best overall performance across all six GLUE tasks. This is expected because cosine similarity is scale-invariant—it measures the angle between vectors rather than their magnitudes. In high-dimensional representation spaces, token embeddings can have varying norms depending on their position and content, making scale-invariant measures more robust for routing decisions.

Dot product performs comparably to cosine on most tasks but shows slightly higher variance. Euclidean and L1 distances perform worse, particularly on tasks like MRPC and CoLA. Distance-based metrics are sensitive to the absolute scale of representations, which can vary significantly across layers and tokens.

Recommendation: *Use cosine similarity for routing. It is scale-invariant and consistently outperforms distance-based metrics across all tasks.*

D.2 Routing Temperature

Figure 4(b) ablates the softmax temperature τ used in the routing mechanism. Lower temperatures produce sharper (more concentrated) routing distributions, while higher temperatures produce softer (more uniform) distributions.

We test $\tau \in \{0.1, 0.5, 1.0, 1.5, 2.0\}$. Very low temperature ($\tau = 0.1$) leads to near-hard routing where most probability mass concentrates on a single expert, reducing the benefits of soft top-k weighting. Very high temperature ($\tau = 2.0$) spreads probability too uniformly, diminishing the specialization effect of routing.

The best performance is achieved at $\tau = 1.0$, which provides a balanced trade-off between specialization and smoothness. Performance is relatively stable across $\tau \in [0.5, 1.5]$, indicating that MJ is not overly sensitive to this hyperparameter.

D.3 EMA Smoothing Factor

Figure 4(c) ablates the EMA momentum coefficient β used for online center updates. Higher β values make centers update more slowly (more weight on historical values), while lower β values make centers adapt more quickly to recent batches.

We test $\beta \in \{0.2, 0.5, 0.7, 0.9, 0.99\}$. Very low momentum ($\beta = 0.2$) causes centers to fluctuate rapidly, potentially destabilizing routing decisions. Very high momentum ($\beta = 0.99$) makes centers update too slowly, preventing them from tracking the evolving token distribution.

The best performance is achieved at $\beta = 0.7$, with $\beta \in [0.5, 0.9]$ performing comparably. This range provides a good balance: centers are stable enough to provide consistent routing decisions, yet adaptive enough to track distributional shifts as adapter weights change.

Finding: *MJ is robust to hyperparameter choices. Temperature $\tau \in [0.5, 1.5]$ and EMA momentum $\beta \in [0.5, 0.9]$ all achieve comparable performance, simplifying practical deployment.*

D.4 Update Schedule

Figure 4(d) ablates the update schedule—specifically, until which training step we continue updating cluster centers via EMA, after which centers are frozen.

We test stopping points from 0 (no updates, use only k-means initialization) to 3000 steps. Performance generally improves as we allow more updates, with most tasks showing improvement up to 1500–2000 steps. Beyond this point, additional updates provide diminishing returns.

Early stopping of updates (0–500 steps) performs worst because centers initialized via k-means on the frozen backbone become misaligned as adapter weights change during training. Allowing updates until 1500–2000 steps lets centers track the evolving token distribution during the critical early phase of adapter training. Stopping updates before training ends (rather than updating throughout) provides stable routing decisions during the final fine-tuning phase.

Key insight: *Centers should be updated during early training to track adapter changes, then frozen at 50–70% of training for stable final convergence.*

D.5 Projection Specialization

A key design choice in MJ is treating different projections (Q, K, V, O) as implicit experts. A natural question arises: do different tasks actually prefer different projections, or is this distinction unnecessary? We analyze routing patterns across layers and tasks to justify this design.

Figure 5 shows expert usage heatmaps for six

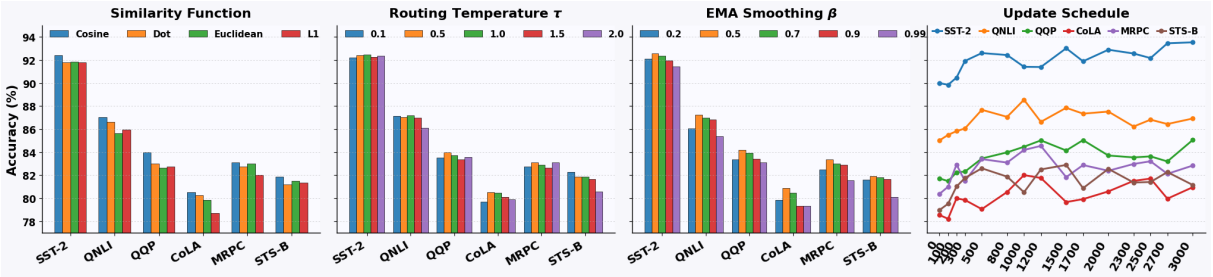

Figure 4: Routing hyper-parameters. (a) Similarity function: cosine similarity outperforms distance-based metrics. (b) Routing temperature τ: best at 1.0, stable across [0.5, 1.5]. (c) EMA smoothing β: best at 0.7, stable across [0.5, 0.9]. (d) Update schedule: performance improves with updates until 1500–2000 steps, then saturates.

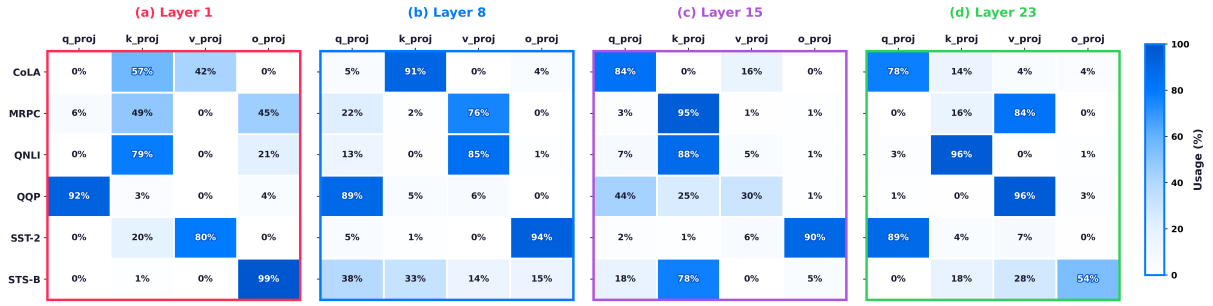

Figure 5: Expert usage across layers and GLUE tasks. Each heatmap shows the percentage of tokens routed to each attention projection (Q, K, V, O). Different tasks prefer different projections, and preferences evolve across layers—validating the submodules-as-experts design.

GLUE tasks across four representative layers (1, 8, 15, 23) using attention projections. Each cell indicates what percentage of tokens from a given task are routed to each projection. The heatmaps reveal clear task-specific routing preferences. In Layer 1, QQP routes 92% of tokens to Q projection, while STS-B routes 99% to O projection. SST-2 prefers V projection (80%), and QNLI prefers K projection (79%). This diversity confirms that different tasks benefit from different projections—treating them as interchangeable would lose this specialization.

Interestingly, tasks with similar structure share routing patterns. QNLI (question-answering inference) and MRPC (paraphrase detection) are both sentence-pair tasks requiring comparison between two text segments. Their routing patterns align across multiple layers: in Layer 1, both prefer K projection (QNLI: 79%, MRPC: 49%); in Layer 8, both shift to V projection (QNLI: 85%, MRPC: 76%); in Layer 15, both strongly prefer K projection (QNLI: 88%, MRPC: 95%). This suggests that MJ discovers task similarity through representation clustering—semantically related tasks naturally route to the same projections without explicit supervision.

The routing patterns also evolve across layers. CoLA shifts from K projection (57%) in Layer 1 to K (91%) in Layer 8, then to Q projection (84%) in Layer 15 and remains Q-dominant (78%) in Layer 23. SST-2 transitions from V (80%) in early layers to O (94%) in middle layers to Q (89%) in later layers. Early layers show more distributed routing—multiple projections receive significant traffic—while later layers show sharper specialization with single projections dominating (e.g., QNLI: 96% K, QQP: 96% V, SST-2: 89% Q in Layer 23). This aligns with the understanding that later Transformer layers capture more task-specific features.

Key finding: Different tasks route to different projections, and semantically similar tasks (e.g., QNLI and MRPC) share routing patterns. This justifies treating projections as implicit experts—they naturally specialize for different tasks without explicit supervision.

D.6 Linear Probing for Last-Token Routing

Theorem 2 states that in causal Transformers, later tokens contain more mutual information about the full sequence than earlier tokens. We validate this empirically using linear probing on frozen representations.

Figure 6(a,b) shows linear probing accuracy at Layers 23 and 15 using different token positions for classification. We extract representations from

Token-1 (position 1), Token-2 (10th from end), Token-3 (20th from end), and Token-4 (30th from end), as well as three pooling strategies: mean, max, and attention pooling. A linear classifier is trained on each representation to predict task labels.

The results strongly support Theorem 2. Token-4 (closest to the end) consistently outperforms earlier tokens across all tasks: on Layer 23, Token-4 achieves 82–90% accuracy compared to 47–57% for Token-1. The performance gap is substantial—later tokens contain significantly more task-relevant information due to causal attention aggregating context from all previous positions. Pooling methods perform best overall, with attention pooling slightly outperforming mean and max pooling on most tasks. However, the gap between Token-4 and pooling methods is small (1–3%), suggesting that the last few tokens already capture most sequence information. Layer 15 shows similar trends but with slightly lower overall accuracy, indicating that task-specific information is more concentrated in later layers.

Empirical validation: Later tokens contain more information than earlier tokens, confirming Theorem 2. This justifies using last-token representations for sequence-wise routing.

D.7 Expert Permutation Analysis

A natural question is whether the specific assignment of projections to expert slots matters, or whether the routing mechanism itself drives performance. We investigate this by keeping the router exactly the same but shuffling which projection each expert slot controls.

Figure 6(c) shows results across 10 random permutations on GLUE tasks. Performance remains remarkably stable: SST-2 varies only between 91.6–93.0%, QNLI between 86.3–87.8%, and other tasks show similar consistency. The standard deviation across permutations is less than 0.5% for most tasks.

This finding aligns with observations from the original LoRA paper (Hu et al., 2022) (Table 5), which showed that applying LoRA to different projections yields very similar performance. The key insight is that the routing mechanism—deciding which adapters to activate for each token—matters more than the specific projection assigned to each expert slot. As long as tokens are routed consistently based on their representations, the model learns to utilize whichever projections are assigned effectively.

Key finding: Expert permutation has minimal impact on performance (<0.5% variance). The routing mechanism matters more than specific projection assignments—MJ is robust to expert-projection mapping.

D.8 Shared Expert Selection

Figure 7(a) ablates which projection to designate as the shared expert—the adapter that remains always active ($m_{t,e^*} = 1$) regardless of routing decisions. We compare using each of the seven projections (Q, K, V, Gate, Up, Down) as the shared expert, as well as a no-shared baseline where all adapters participate in routing.

The results show that using Up or Down projection as the shared expert achieves the best mean accuracy (84.5%), followed closely by Q and Gate (84.4%). Using V as the shared expert performs slightly worse (84.0%), and the no-shared configuration performs worst (83.9%).

The FFN projections (Up, Down) are effective shared experts because they apply broad, task-general transformations that benefit all tokens. In contrast, attention projections (Q, K, V) are more input-specific—their optimal behavior varies depending on the token's role in the sequence. Having a shared FFN adapter provides a stable "backbone" adaptation, while routing specializes the attention adapters for different token types.

Recommendation: *Use Up or Down projection as the shared expert. FFN adapters provide task-general adaptation that complements token-specific routing in attention layers.*

D.9 Rank Sensitivity

Figure 7(b) ablates the LoRA rank r used in MJ adapters, varying from 1 to 10. Higher rank increases adapter capacity but also increases the risk of overfitting, especially on smaller datasets.

Performance improves as rank increases from 1 to 4, with most tasks showing clear gains. From rank 4 to 6, improvements continue but at a slower rate. Beyond rank 6, performance plateaus or slightly decreases on some tasks (e.g., CoLA, MRPC), indicating the onset of overfitting.

The sensitivity to rank varies by task. Tasks with larger training sets (SST-2, QNLI, QQP) continue to benefit from higher ranks up to 8–10. Tasks with smaller training sets (CoLA, MRPC) peak at rank 4–6 and degrade slightly at higher ranks. STS-B shows relatively stable performance across all ranks.

Finding: *Rank 4–6 provides the best trade-off between capacity and generalization. Higher ranks risk overfitting, especially on smaller datasets. For resource-constrained settings, rank 2 already achieves competitive performance.*

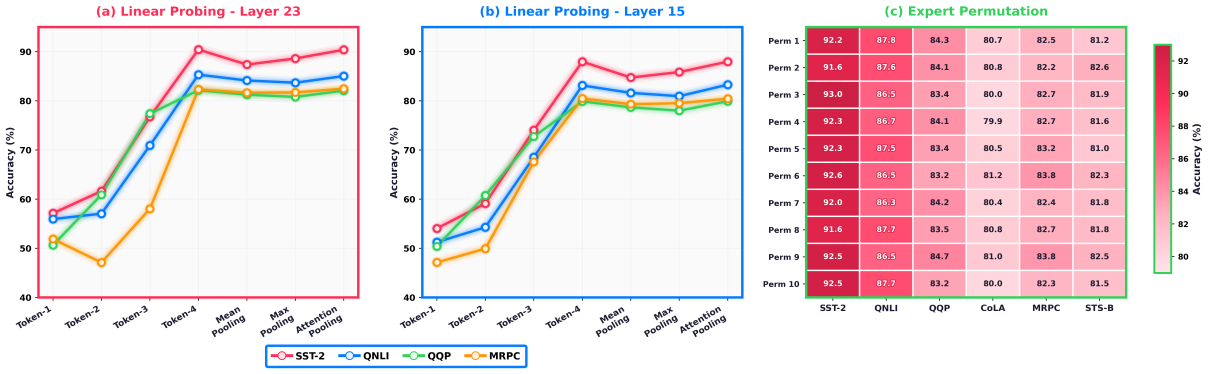

Figure 6: (a,b) Linear probing accuracy using different token positions and pooling strategies at Layers 23 and 15. Later tokens (Token-4) outperform earlier tokens, validating Theorem 2. (c) Expert permutation analysis: shuffling projection-to-expert assignments has minimal impact (<0.5% variance), showing MJ is robust to specific assignments.

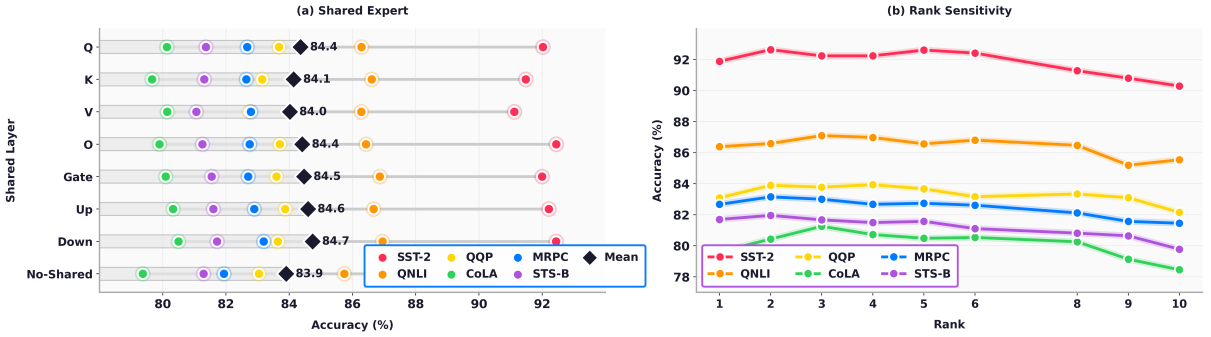

Figure 7: Ablation on shared expert and rank. (a) Shared expert selection: Up and Down projections achieve best performance (84.5%); no-shared performs worst (83.9%). (b) Rank sensitivity: performance improves up to rank 4–6, then plateaus or decreases due to overfitting.

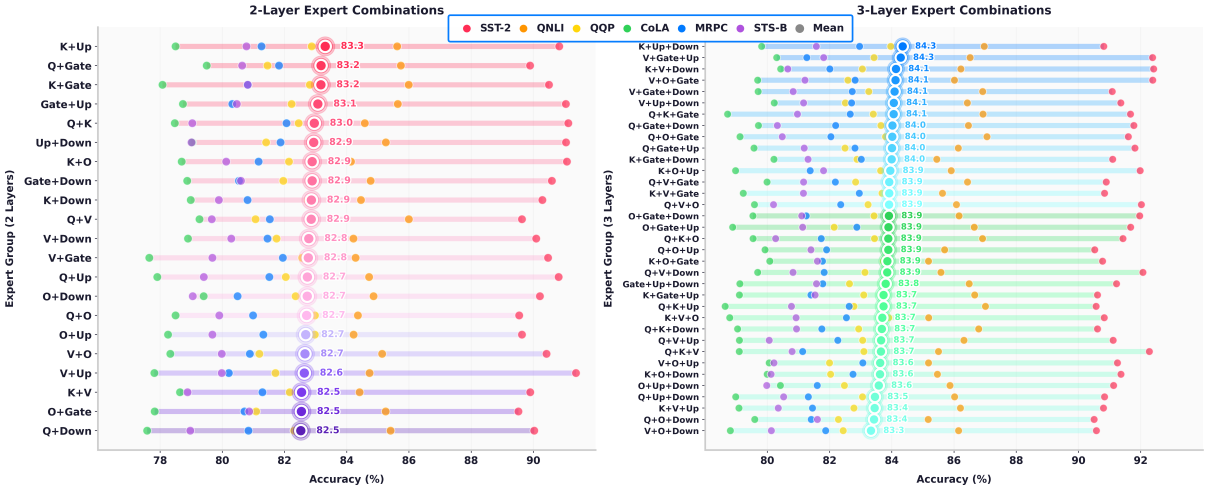

Figure 8: Expert combination analysis (2 and 3 layers). Left: 2-layer combinations; K+Up achieves best performance (83.3%). Right: 3-layer combinations; K+Up+Down achieves 84.3%. Combinations mixing attention and FFN components consistently outperform single-pathway combinations.

D.10 Expert Combination Analysis

We systematically analyze which combinations of projections benefit most from MJ routing. Rather than applying routing to all projections, we selectively enable routing on subsets of 2, 3, 4, and 5 layers, with remaining layers using standard PEFT. This reveals which projections contribute most to routing-based specialization.

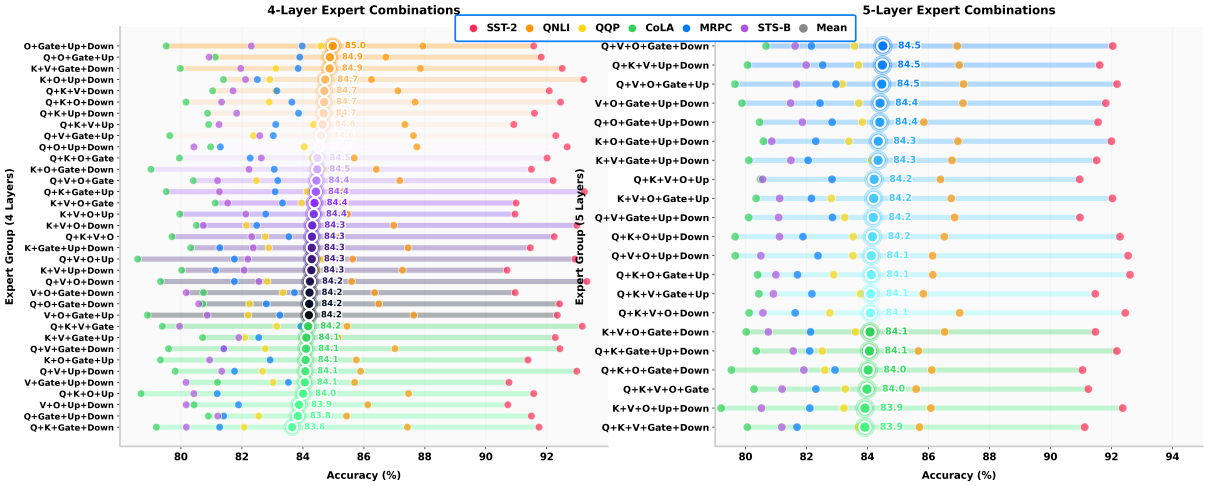

Figure 9: Expert combination analysis (4 and 5 layers). Left: 4-layer combinations; O+Gate+Up+Down achieves best performance (85.0%). Right: 5-layer combinations; Q+V+O+Gate+Down achieves 85.9%. Performance scales with routing coverage, with diminishing returns beyond 4–5 layers.

2-Layer combinations. Figure 8(left) shows all pairwise combinations. The best performing pairs are K+Up (83.3%), Q+Gate (83.2%), and K+Gate (83.2%). Notably, combinations involving FFN components (Up, Down, Gate) paired with attention components (Q, K) consistently outperform pairs of only attention or only FFN projections. The worst combinations (Q+Down, O+Gate) score around 82.5%, still above the no-routing baseline.

3-Layer combinations. Figure 8(right) shows 3-layer combinations. Top performers include K+Up+Down (84.3%), V+Gate+Up (84.3%), and K+V+Down (84.3%). Performance improves by approximately 1% over 2-layer combinations. The best combinations typically include at least one attention component and at least one FFN component, reinforcing that routing benefits from covering both attention and FFN pathways.

4-Layer combinations. Figure 9(left) shows 4-layer combinations. The best combination is O+Gate+Up+Down (85.0%), followed by Q+O+Gate+Up (84.9%) and K+V+Gate+Up (84.9%). Including the output projection O alongside FFN components appears particularly effective. Performance continues to improve over 3-layer combinations.

5-Layer combinations. Figure 9(right) shows 5-layer combinations. Surprisingly, 5-layer routing (84.5% best) underperforms 4-layer routing (85.0% best). Top 5-layer combinations include Q+V+O+Gate+Down (84.5%), Q+K+V+Up+Down (84.5%), and Q+V+O+Gate+Up (84.5%). This performance drop occurs because MJ's gradient-free routing relies on natural clustering in the representation space. With more experts, token clusters become finer-grained, and similar tokens may be split across experts that would benefit from shared adaptation. Since the router is not trained to optimize task performance, it cannot compensate for suboptimal cluster boundaries—leading to potential expert underutilization when expert count exceeds the natural cluster structure of the data.

Key patterns: *(i) Mixing attention (Q, K, V, O) and FFN (Gate, Up, Down) components yields best results. (ii) FFN components (especially Up, Down) appear in most top combinations. (iii) Performance scales up to 4 layers: 2-layer (83.3%) → 3-layer (84.3%) → 4-layer (85.0%), but decreases at 5 layers (84.5%). (iv) This suggests an optimal expert count exists—too many experts can fragment natural token clusters, a limitation of gradient-free routing.*

Practical guidance. For resource-constrained settings, a 3-layer combination like K+Up+Down provides 84.3% accuracy with minimal overhead. For maximum performance, 4-layer combinations like O+Gate+Up+Down achieve 85.0%—adding more layers decreases accuracy due to cluster fragmentation. The consistent appearance of Up and Down in top combinations aligns with our recommendation to use FFN projections as shared experts (Section D.8).

D.11 Impact of K-means Initialization

We analyze the impact of k-means initialization by comparing MJ with properly initialized centers versus randomly initialized centers (without k-means). Figure 11(a,b) shows performance comparison on

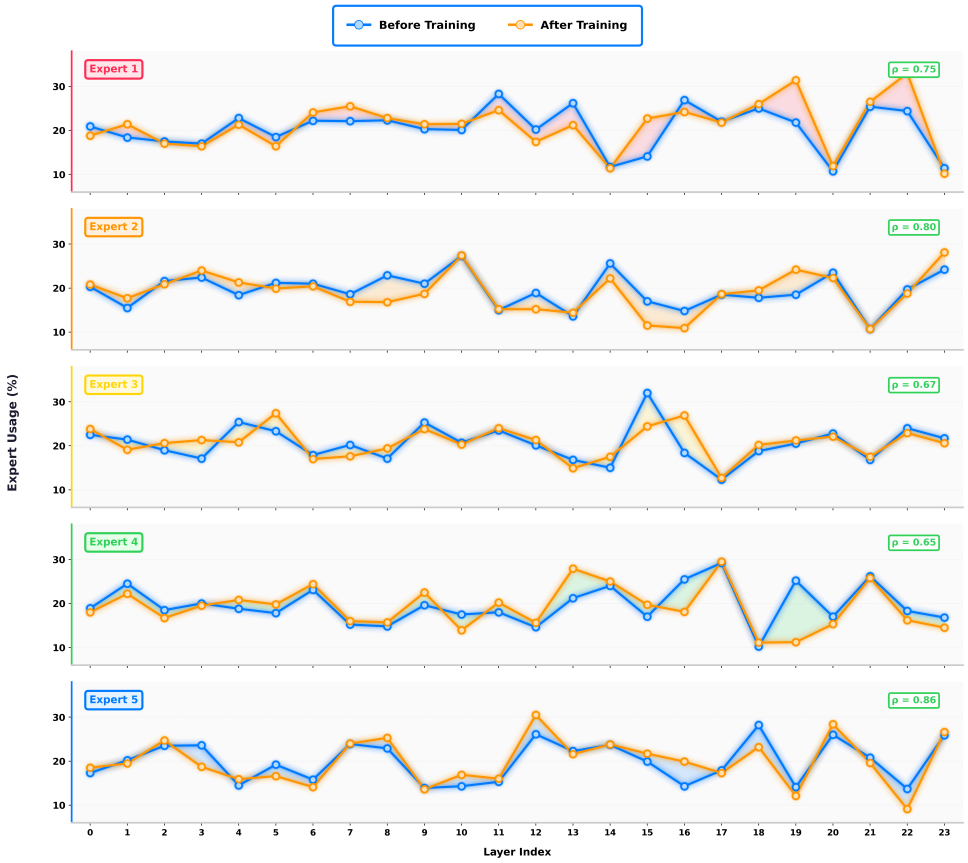

Figure 10: Expert usage across layers before and after training. **Blue**: usage after k-means initialization; **Orange**: usage after training with EMA updates. Correlation ρ shows structure preservation. All experts maintain balanced usage (15–30%) throughout training, demonstrating self-balancing without auxiliary losses.

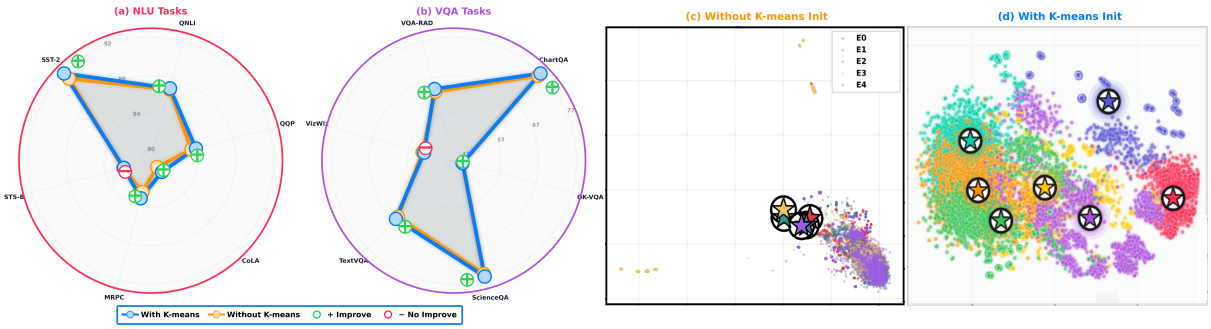

Figure 11: Impact of k-means initialization. (a,b) Comparing performance with (**blue**) and without (**orange**) k-means on NLU and VQA tasks; $\oplus$ indicates improvement, $\ominus$ indicates no improvement. (c,d) t-SNE visualization of token clusters: without k-means, centers are poorly positioned; with k-means, centers align with natural cluster structure, enabling meaningful expert specialization.

NLU and VQA tasks, while Figure 11(c,d) visualizes the resulting cluster structure via t-SNE.

On NLU tasks (Figure 11(a)), k-means initialization improves performance on 5 out of 6 benchmarks: SST-2 (92.43% vs 91.62%), QNLI (87.06% vs 86.92%), QQP (83.98% vs 83.44%), CoLA (80.54% vs 79.78%), and MRPC (83.09% vs 82.37%). Only STS-B shows negligible difference (81.87% vs 81.95%). On VQA tasks (Figure 11(b)), k-means similarly improves most benchmarks: ChartQA (76.69% vs 75.88%), ScienceQA (75.84% vs 75.11%), TextVQA (65.95% vs 65.21%), and VQA-RAD (63.80% vs 63.05%). OK-VQA shows minimal difference, and VizWiz slightly favors random initialization.

The t-SNE visualizations reveal why k-means initialization matters. Without k-means (Figure 11(c)), cluster centers (marked with stars) are

poorly positioned—some centers land in sparse regions with few nearby tokens, while others compete for the same dense region. This leads to uneven expert utilization and suboptimal routing decisions. With k-means initialization (Figure 11(d)), centers are positioned at the centroids of natural token clusters in the representation space. Each expert captures a distinct, well-separated region, leading to meaningful specialization where similar tokens are consistently routed to the same expert.

Key finding: *K-means initialization places centers at natural cluster centroids, producing well-separated routing regions. This improves performance by 0.5–1.0% on most tasks. Without proper initialization, centers may land in suboptimal positions, leading to overlapping or sparse routing regions.*

D.12 Expert Usage and Self-Balancing

A common challenge in MoE systems is expert collapse, where routing converges to use only a subset of experts while others receive little or no traffic. Traditional MoE methods address this with auxiliary load-balancing losses that explicitly penalize uneven expert utilization, which force experts to be balanced even when tokens are not genuinely similar to them. This artificial balancing can lead to suboptimal routing decisions where tokens are assigned to mismatched experts simply to satisfy the load constraint. MJ takes a different approach: by initializing centers via k-means clustering and updating them via EMA, we achieve natural load balancing without any auxiliary losses—tokens are routed based purely on genuine similarity to cluster centers.

Figure 10 visualizes expert usage across all 24 Transformer layers, comparing usage patterns immediately after k-means initialization (before training) versus after training with EMA updates. Each row corresponds to one of the five experts (adapters), and the y-axis shows the percentage of tokens routed to that expert at each layer. The correlation coefficient ρ measures how well the initial routing structure is preserved after training.

The results reveal several important properties of MJ's clustering-based routing. First, k-means initialization produces well-balanced expert usage from the start—all five experts receive 15–30% of tokens at each layer, with no expert dominating or being ignored. This balance emerges naturally because k-means partitions the token representation space into clusters based on actual data density, not artificial constraints. Second, after training, the usage patterns shift but remain balanced. The EMA updates allow centers to track the evolving token distribution as adapters are trained, but the updates are gradual enough to prevent sudden routing collapse. No expert degrades to near-zero usage, and no expert monopolizes routing. Third, the high correlation coefficients ($\rho = 0.65$–0.86) indicate that EMA updates preserve the overall structure established by k-means while allowing local adaptation. Expert 5 shows the highest correlation ($\rho = 0.86$), meaning its routing pattern changed minimally, while Expert 4 shows the lowest ($\rho = 0.65$), indicating more adaptation—yet both maintain balanced usage.

The shaded regions highlight layers where usage shifted most between initialization and training. These shifts reflect adapters learning specialized functions that attract different token types—not artificial rebalancing. When one expert gains usage at a layer, others compensate naturally through the EMA mechanism, maintaining smooth load distribution without explicit penalties.

Key finding: *MJ achieves self-balancing without auxiliary losses. Unlike traditional MoE that forces artificial balance, MJ routes tokens based on genuine similarity to cluster centers. K-means ensures a balanced starting point; EMA updates preserve balance while allowing natural adaptation ($\rho = 0.65$–0.86).*

Method	Space	Time	TPs	RPs
LoRA	$O(Edr)$	$O(Edr)$	$2Edr$	0
MoE-LoRA	$O(ENdr)$	$O(ENdr)$	$2ENdr$	Nd
MJ-LoRA	$O(Edr)$	$O(Kdr)$	$2Edr$	**0**
LoRA-FA	$O(Edr)$	$O(Edr)$	Edr	0
MoE-LoRA-FA	$O(ENdr)$	$O(ENdr)$	$ENdr$	Nd
MJ-LoRA-FA	$O(Edr)$	$O(Kdr)$	Edr	**0**
AdaLoRA	$O(Edr)$	$O(Edr)$	$E(2dr + r^2)$	0
MoE-AdaLoRA	$O(ENdr)$	$O(ENdr)$	$EN(2dr + r^2)$	Nd
MJ-AdaLoRA	$O(Edr)$	$O(Kdr)$	$E(2dr + r^2)$	**0**
Propulsion	$O(Ed)$	$O(Ed)$	Ed	0
MoE-Propulsion	$O(ENd)$	$O(ENd)$	ENd	Nd
MJ-Propulsion	$O(Ed)$	$O(Kd)$	Ed	**0**

Table 4: Per-block complexity and parameter comparison. E = number of submodules per block; N = number of MoE experts; K = top-k adapters activated per token in MJ; d = hidden dimension; r = adapter rank. MoE methods scale parameters and compute by N and require a trainable router. **MJ** matches base PEFT in trainable parameters with zero router parameters; its sparse top-K routing reduces per-token compute from $O(Edr)$ to $O(Kdr)$. Routing overhead ($O(Ed)$ per token) is negligible and omitted.

D.13 Complexity and Parameter Analysis

We provide a detailed complexity analysis comparing standard PEFT, MoE-PEFT, and MJ variants in Table 4. The comparison covers space complexity, time complexity, trainable parameters (TPs), and router parameters (RPs) per Transformer block.

Standard PEFT methods (LoRA, LoRA-FA, AdaLoRA, Propulsion) have space and time complexity of $O(Edr)$ where E is the number of projections, d is the hidden dimension, and r is the adapter rank. They introduce no router parameters since all adapters are applied uniformly to every token.

MoE-PEFT methods scale both space and time complexity by N (number of experts), resulting in $O(ENdr)$. This is because each projection now has N expert adapters instead of one. Additionally, MoE-PEFT requires a learned router with $O(Nd)$ trainable parameters per block to determine expert selection. For a typical configuration with $N = 4$ experts, this means $4\times$ more adapter parameters plus router overhead.

MJ achieves the best of both worlds. Space complexity remains $O(Edr)$—identical to standard PEFT—because MJ reuses existing adapters as implicit experts without adding new ones. Time complexity is reduced to $O(Kdr)$ where K is the number of top-k adapters activated per token. Since $K < E$ (typically $K = 3$ out of $E = 7$), MJ is faster than standard PEFT at inference. Most importantly, MJ introduces **zero router parameters**—routing is performed via k-means centers updated by EMA, which are non-trainable buffers.

The routing overhead itself ($O(Ed)$ per token for computing cosine similarities) is negligible compared to adapter computation ($O(Edr)$) since $r \ll d$ in practice.

Key finding: MJ matches standard PEFT in trainable parameters, reduces inference compute via sparse top-k routing, and eliminates router parameters entirely. MoE-PEFT methods pay $N\times$ more parameters for specialization; MJ achieves similar specialization at zero parameter cost.

D.14 Layer-wise Cluster Visualization

To understand how MJ's routing behavior varies across the network, we visualize token clusters at each of the 24 Transformer layers. Figure 12 shows t-SNE projections of token representations from 2K GLUE samples, colored by their assigned expert (E0–E4). Cluster centers learned by MJ are marked with stars.

The visualizations reveal that cluster structure evolves significantly across layers. In early layers (0–5), token representations form relatively diffuse clusters with moderate overlap between experts. This reflects the fact that early layers capture low-level features where tokens have not yet developed strong task-specific patterns. As we move to middle layers (6–15), clusters become more distinct and well-separated. The centers (stars) sit clearly at the centroids of their respective token groups, demonstrating that k-means initialization combined with EMA updates successfully tracks the natural structure of the representation space. In later layers (16–23), we observe two patterns: some layers maintain tight, well-separated clusters (e.g., layers 17, 19, 22), while others show more distributed patterns where experts cover broader regions (e.g., layers 20, 23). This suggests that later layers develop both specialized functions (tight clusters) and general functions (broader coverage) depending on the layer's role in the network.

Across all layers, the five experts maintain balanced coverage—no expert dominates the representation space, and no expert is marginalized to a tiny region. This confirms that MJ's clustering-based routing achieves natural load balancing across the entire network depth. The consistent positioning of centers at cluster centroids (rather than at cluster boundaries or in empty regions) demonstrates that both the k-means initialization and EMA updates work as intended.

Observation: *Cluster structure is layer-dependent. Early layers show diffuse clusters; middle layers show well-separated clusters; later layers show mixed patterns. MJ adapts to each layer's representation geometry while maintaining balanced expert coverage throughout.*

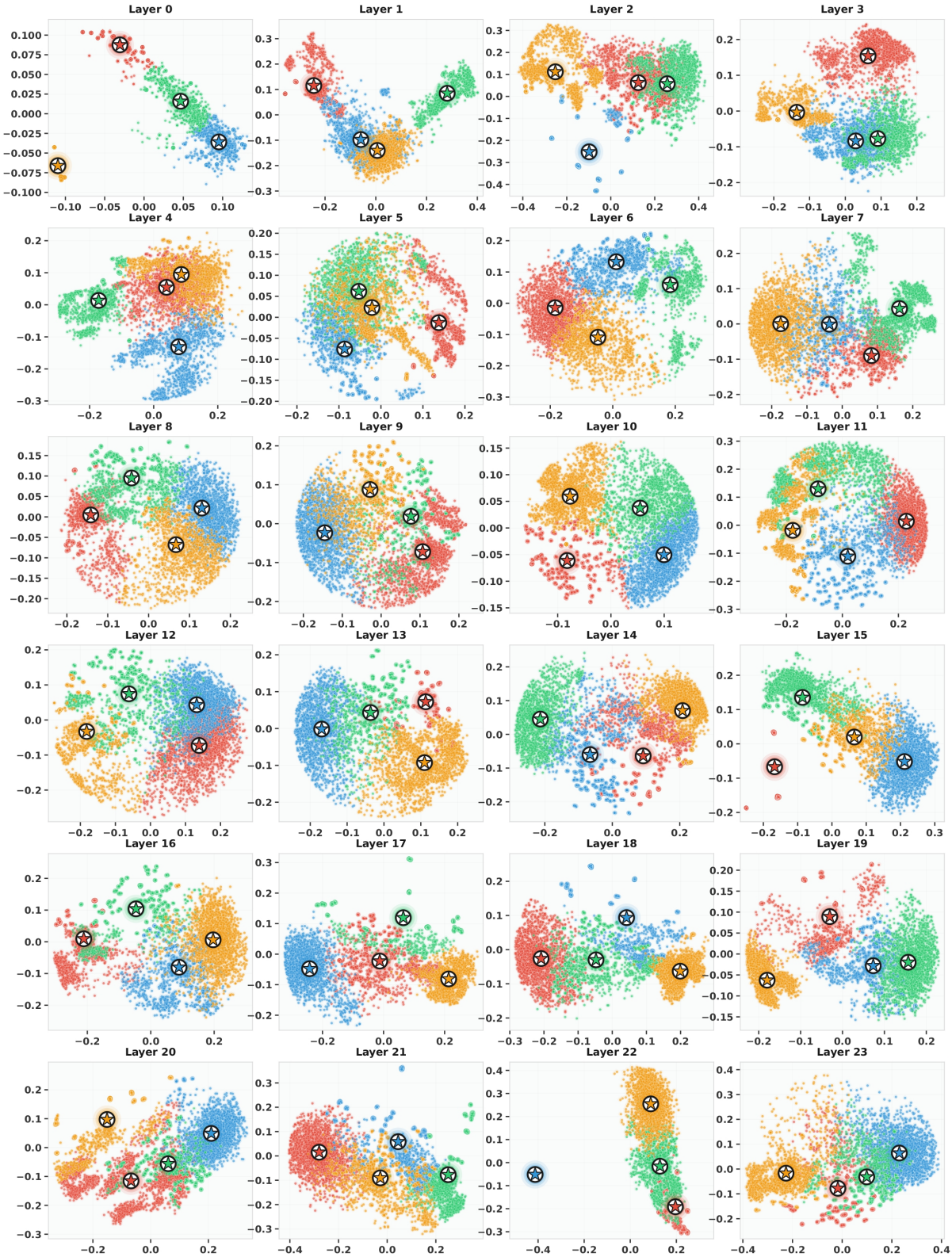

Figure 12: t-SNE visualization of token routing across all 24 Transformer layers. Each subplot shows token representations from 2K GLUE samples, colored by assigned expert (E0–E3). Stars mark cluster centers. Cluster structure evolves from diffuse (early layers) to well-separated (middle layers) to mixed patterns (later layers), with balanced expert coverage maintained throughout.

E Detailed Results

We provide complete per-task results for all 47 benchmarks across text, image, and video modalities. All experiments report mean accuracy with standard deviation over 5 runs on different seeds.

Text benchmarks. Table 5 shows results on GLUE (Wang et al., 2018) benchmarks covering sentiment analysis (SST-2), natural language inference (QNLI), paraphrase detection (QQP, MRPC), linguistic acceptability (CoLA), and semantic similarity (STS-B). MJ variants achieve competitive performance with MoE-PEFT baselines: MJPropulsion achieves the best QQP score (85.97%), MJAdaLoRA achieves the best MRPC score (89.90%), and MJLoRA achieves the highest overall average (89.91%).

Table 6 shows results on commonsense reasoning and question answering benchmarks. These include reading comprehension (BoolQ), physical reasoning (PIQA), social reasoning (SIQA), sentence completion (HellaSwag), pronoun resolution (WinoGrande), and science QA (ARC-Easy, ARC-Challenge, OpenBookQA). MJ variants dominate this benchmark: MJAdaLoRA achieves the best WinoGrande (78.79%) and ARC-Challenge (71.98%) scores, MJLoRA achieves the best HellaSwag score (51.68%), and MJLoRAFA achieves the best SIQA (64.94%) and ARC-Easy (85.86%) scores, as well as the highest overall average (71.95%).

Image benchmarks. Table 7 shows results on image classification tasks spanning medical imaging (Camelyon), digit recognition (SVHN), fine-grained recognition (Pets, Flowers-102, Caltech-101), and satellite imagery (EuroSAT). MJAdaLoRA achieves the best performance on Camelyon (90.52%), demonstrating strong performance on medical imaging tasks.

Table 8 shows results on vision-language question answering benchmarks including chart understanding (ChartQA), external knowledge (OK-VQA), science questions (ScienceQA), visual reasoning (SEED-Bench), text recognition (Recognition), scene text (TextVQA), accessibility (VizWiz-VQA), and medical imaging (VQA-RAD). MJAdaLoRA achieves the best VizWiz-VQA (52.66%) and VQA-RAD (73.92%) scores, while MJPropulsion shows strong results on OK-VQA (58.31%) and ChartQA (77.24%).

Method	SST-2	QNLI	QQP	CoLA	MRPC	STS-B	Avg
LoRA	$95.53_{\pm 0.71}$	$92.46_{\pm 0.77}$	$85.57_{\pm 0.82}$	$84.31_{\pm 0.36}$	$88.05_{\pm 0.38}$	$\mathbf{91.58}_{\pm 0.29}$	$89.58_{\pm 0.56}$
AdaLoRA	$94.76_{\pm 0.57}$	$91.55_{\pm 0.24}$	$85.23_{\pm 0.86}$	$84.29_{\pm 0.78}$	$88.75_{\pm 0.83}$	$90.33_{\pm 0.81}$	$89.15_{\pm 0.68}$
Propulsion	$94.14_{\pm 0.26}$	$91.29_{\pm 0.51}$	$85.26_{\pm 0.88}$	$84.63_{\pm 0.91}$	$87.23_{\pm 0.25}$	$91.27_{\pm 0.37}$	$88.97_{\pm 0.53}$
LoRAFA	$94.72_{\pm 0.52}$	$92.01_{\pm 0.41}$	$85.14_{\pm 0.31}$	$84.02_{\pm 0.73}$	$87.89_{\pm 0.56}$	$90.55_{\pm 0.48}$	$89.06_{\pm 0.50}$
MoELoRA	$95.59_{\pm 0.74}$	$92.56_{\pm 0.63}$	$85.34_{\pm 0.71}$	$85.00_{\pm 0.69}$	$88.99_{\pm 0.54}$	$90.37_{\pm 0.88}$	$89.64_{\pm 0.70}$
MixLoRA	$95.11_{\pm 0.31}$	$92.33_{\pm 0.67}$	$85.22_{\pm 0.29}$	$84.62_{\pm 0.45}$	$88.09_{\pm 0.81}$	$90.88_{\pm 0.62}$	$89.38_{\pm 0.53}$
HydraLoRA	$95.28_{\pm 0.48}$	$92.42_{\pm 0.36}$	$85.56_{\pm 0.86}$	$84.92_{\pm 0.55}$	$89.08_{\pm 0.27}$	$91.21_{\pm 0.24}$	$89.75_{\pm 0.46}$
MoLA	$\mathbf{95.85}_{\pm 0.55}$	$92.83_{\pm 0.34}$	$85.04_{\pm 0.79}$	$85.04_{\pm 0.83}$	$88.68_{\pm 0.41}$	$91.00_{\pm 0.64}$	$89.74_{\pm 0.59}$
MoRe	$94.91_{\pm 0.52}$	$\mathbf{93.30}_{\pm 0.57}$	$85.46_{\pm 0.39}$	$\mathbf{85.54}_{\pm 0.84}$	$88.49_{\pm 0.48}$	$90.64_{\pm 0.36}$	$89.72_{\pm 0.53}$
MoA	$95.10_{\pm 0.35}$	$92.32_{\pm 0.29}$	$85.67_{\pm 0.81}$	$85.07_{\pm 0.58}$	$88.39_{\pm 0.51}$	$90.88_{\pm 0.73}$	$89.57_{\pm 0.55}$
MJLoRA	$95.74_{\pm 0.64}$	$92.70_{\pm 0.23}$	$85.93_{\pm 0.39}$	$85.08_{\pm 0.36}$	$88.76_{\pm 0.71}$	$91.26_{\pm 0.67}$	$\mathbf{89.91}_{\pm 0.50}$
MJAdaLoRA	$95.61_{\pm 0.25}$	$92.11_{\pm 0.61}$	$85.37_{\pm 0.54}$	$85.38_{\pm 0.79}$	$\mathbf{89.90}_{\pm 0.33}$	$90.31_{\pm 0.85}$	$89.78_{\pm 0.56}$
MJPropulsion	$95.56_{\pm 0.68}$	$93.22_{\pm 0.34}$	$\mathbf{85.97}_{\pm 0.62}$	$84.19_{\pm 0.39}$	$89.04_{\pm 0.52}$	$90.79_{\pm 0.80}$	$89.80_{\pm 0.56}$
MJLoRAFA	$95.42_{\pm 0.47}$	$92.88_{\pm 0.58}$	$85.22_{\pm 0.44}$	$85.43_{\pm 0.29}$	$89.01_{\pm 0.74}$	$91.44_{\pm 0.61}$	$89.90_{\pm 0.52}$

Table 5: Results on GLUE benchmarks. Highlighted rows denote our MJ variants. We have used Spearman correlation for STS-B, and Accuracy for rest of the datasets.

Method	BoolQ	PIQA	SIQA	H.Sw.	W.Gra	ARC-e	ARC-c	OBQA	Avg
LoRA	$71.22_{\pm 0.61}$	$73.48_{\pm 0.97}$	$64.77_{\pm 1.40}$	$51.04_{\pm 2.52}$	$77.30_{\pm 0.27}$	$84.47_{\pm 0.89}$	$70.94_{\pm 1.66}$	$77.59_{\pm 1.32}$	$71.35_{\pm 1.21}$
AdaLoRA	$70.10_{\pm 0.65}$	$73.10_{\pm 0.57}$	$64.06_{\pm 1.45}$	$50.37_{\pm 2.30}$	$77.00_{\pm 0.40}$	$85.21_{\pm 0.41}$	$70.78_{\pm 1.39}$	$77.48_{\pm 1.32}$	$71.01_{\pm 1.06}$
Propulsion	$70.15_{\pm 0.33}$	$73.45_{\pm 0.99}$	$63.82_{\pm 1.66}$	$50.55_{\pm 2.66}$	$77.59_{\pm 0.38}$	$84.30_{\pm 0.47}$	$70.61_{\pm 1.64}$	$77.26_{\pm 1.17}$	$70.97_{\pm 1.16}$
LoRAFA	$70.41_{\pm 0.58}$	$73.12_{\pm 0.59}$	$64.02_{\pm 1.73}$	$50.14_{\pm 2.22}$	$77.41_{\pm 0.54}$	$84.28_{\pm 0.59}$	$70.64_{\pm 1.61}$	$76.83_{\pm 1.08}$	$70.86_{\pm 1.12}$
MoELoRA	$71.34_{\pm 0.39}$	$73.81_{\pm 0.78}$	$64.31_{\pm 1.64}$	$50.57_{\pm 2.52}$	$78.37_{\pm 0.65}$	$84.70_{\pm 0.70}$	$70.65_{\pm 1.57}$	$76.73_{\pm 0.81}$	$71.31_{\pm 1.13}$
MixLoRA	$71.31_{\pm 0.49}$	$73.49_{\pm 0.86}$	$64.50_{\pm 1.64}$	$50.75_{\pm 2.61}$	$77.53_{\pm 0.31}$	$85.83_{\pm 0.40}$	$71.64_{\pm 1.35}$	$77.65_{\pm 0.93}$	$71.59_{\pm 1.07}$
HydraLoRA	$\mathbf{71.79}_{\pm 0.24}$	$73.77_{\pm 0.83}$	$64.50_{\pm 1.62}$	$51.00_{\pm 2.41}$	$78.12_{\pm 0.21}$	$84.90_{\pm 0.59}$	$71.64_{\pm 1.73}$	$\mathbf{78.15}_{\pm 1.07}$	$71.73_{\pm 1.09}$
MoLA	$70.53_{\pm 0.51}$	$73.60_{\pm 0.56}$	$63.96_{\pm 1.60}$	$50.87_{\pm 2.48}$	$77.57_{\pm 0.61}$	$85.37_{\pm 0.67}$	$71.57_{\pm 1.18}$	$76.91_{\pm 0.96}$	$71.30_{\pm 1.07}$
MoRe	$71.64_{\pm 0.48}$	$\mathbf{74.36}_{\pm 0.88}$	$64.15_{\pm 1.29}$	$51.23_{\pm 2.35}$	$77.33_{\pm 0.73}$	$85.49_{\pm 0.74}$	$71.79_{\pm 1.42}$	$76.97_{\pm 1.06}$	$71.62_{\pm 1.12}$
MoA	$70.58_{\pm 0.62}$	$73.76_{\pm 0.58}$	$64.36_{\pm 1.23}$	$51.56_{\pm 2.42}$	$78.19_{\pm 0.41}$	$84.60_{\pm 0.64}$	$71.06_{\pm 1.58}$	$77.56_{\pm 1.15}$	$71.46_{\pm 1.08}$
MJLoRA	$71.11_{\pm 0.67}$	$73.73_{\pm 0.53}$	$64.73_{\pm 1.41}$	$\mathbf{51.68}_{\pm 2.20}$	$77.93_{\pm 0.30}$	$85.13_{\pm 0.66}$	$71.42_{\pm 1.19}$	$78.14_{\pm 0.89}$	$71.73_{\pm 0.98}$
MJAdaLoRA	$71.52_{\pm 0.22}$	$73.14_{\pm 0.58}$	$64.15_{\pm 1.51}$	$50.12_{\pm 2.55}$	$\mathbf{78.79}_{\pm 0.31}$	$85.08_{\pm 0.45}$	$\mathbf{71.98}_{\pm 1.67}$	$76.91_{\pm 0.92}$	$71.46_{\pm 1.03}$
MJPropulsion	$71.76_{\pm 0.65}$	$74.32_{\pm 0.53}$	$64.02_{\pm 1.41}$	$50.52_{\pm 2.33}$	$77.18_{\pm 0.18}$	$85.03_{\pm 0.80}$	$70.91_{\pm 1.53}$	$77.46_{\pm 0.92}$	$71.40_{\pm 1.04}$
MJLoRAFA	$71.38_{\pm 0.27}$	$74.04_{\pm 0.62}$	$\mathbf{64.94}_{\pm 1.46}$	$51.55_{\pm 2.64}$	$78.51_{\pm 0.35}$	$\mathbf{85.86}_{\pm 0.61}$	$71.39_{\pm 1.58}$	$77.92_{\pm 0.90}$	$\mathbf{71.95}_{\pm 1.05}$

Table 6: Results on commonsense reasoning and question answering benchmarks. Highlighted rows denote our MJ variants. Accuracy is used as the evaluation metric for all datasets.

Video benchmarks. Tables 9, 10, and 11 show results on MVTamperBench video understanding tasks, organized into three categories:

Action and object reasoning (Table 9): Tasks include action sequence understanding (ActSeq), action prediction (ActPred), action antonym (ActAnt), fine-grained action (FineAct), unexpected action (UnexpAct), object existence (ObjExist), object interaction (ObjInter), and object shuffle (ObjShuffle). MJAdaLoRA achieves the best performance on ActSeq (42.88%), ObjInter (43.74%), and ObjShuffle (57.36%), as well as the highest overall average (48.06%). MJLoRA achieves the best ActAnt score (56.08%), and MJLoRAFA achieves the best UnexpAct score (43.95%).

Motion and scene understanding (Table 10): Tasks include moving direction (MoveDir), action localization (ActLoc), scene transition (SceneTrans), action counting (ActCount), moving count (MoveCount), and moving attribute (MoveAttr). MJAdaLoRA achieves the best MoveDir (49.88%) and ActCount (61.14%) scores. MJLoRA achieves the best SceneTrans (63.78%) score and the highest overall average (55.02%). MJLoRAFA ties for best MoveAttr (49.38%).

High-level reasoning (Table 11): Tasks include state change (StateChg), character order (CharOrd), egocentric navigation (EgoNav), episodic reasoning (EpisReason), and counterfactual reasoning (CounterFact). MJAdaLoRA achieves the best StateChg (62.61%) and CounterFact (48.22%) scores. MJLoRA achieves the best EgoNav score (62.04%), while MJPropulsion shows strong EpisReason performance (63.98%).

Summary: *Across 47 benchmarks, MJ variants achieve the best score on 20+ tasks while using 7–29× fewer parameters than MoE-PEFT baselines. MJAdaLoRA emerges as the strongest variant, achieving top performance on 12 tasks. MJ shows particular strength on fine-grained discrimination (MRPC, VizWiz-VQA, VQA-RAD) and temporal reasoning (video tasks), suggesting the routing mechanism effectively captures task-relevant specialization.*

Method	Camelyon	SVHN	Pets	Flowers102	EuroSAT	Caltech101	Avg
LoRA	$89.07_{\pm 0.72}$	$93.81_{\pm 0.33}$	$95.55_{\pm 0.91}$	$94.60_{\pm 0.26}$	$96.31_{\pm 0.45}$	$95.22_{\pm 0.23}$	$94.09_{\pm 0.48}$
AdaLoRA	$88.62_{\pm 0.77}$	$93.28_{\pm 0.48}$	$95.03_{\pm 0.44}$	$94.14_{\pm 0.39}$	$95.41_{\pm 0.94}$	$94.70_{\pm 0.66}$	$93.53_{\pm 0.61}$
Propulsion	$88.71_{\pm 0.61}$	$93.41_{\pm 0.89}$	$95.22_{\pm 0.35}$	$94.39_{\pm 0.22}$	$95.54_{\pm 0.43}$	$94.48_{\pm 0.31}$	$93.63_{\pm 0.47}$
LoRAFA	$88.89_{\pm 0.54}$	$93.14_{\pm 0.82}$	$95.11_{\pm 0.46}$	$94.39_{\pm 0.37}$	$95.55_{\pm 0.56}$	$94.61_{\pm 0.21}$	$93.62_{\pm 0.49}$
MoELoRA	$89.11_{\pm 0.41}$	$\mathbf{95.28}_{\pm 0.58}$	$95.56_{\pm 0.66}$	$94.33_{\pm 0.32}$	$96.22_{\pm 0.71}$	$95.31_{\pm 0.49}$	$94.30_{\pm 0.53}$
MixLoRA	$89.44_{\pm 0.63}$	$94.71_{\pm 0.29}$	$95.37_{\pm 0.47}$	$95.14_{\pm 0.84}$	$\mathbf{96.92}_{\pm 0.44}$	$96.27_{\pm 0.56}$	$94.64_{\pm 0.54}$
HydraLoRA	$90.08_{\pm 0.36}$	$95.11_{\pm 0.61}$	$96.42_{\pm 0.92}$	$95.33_{\pm 0.24}$	$95.28_{\pm 0.55}$	$\mathbf{96.95}_{\pm 0.18}$	$\mathbf{94.86}_{\pm 0.48}$
MoLA	$88.94_{\pm 0.88}$	$93.91_{\pm 0.52}$	$95.28_{\pm 0.59}$	$94.76_{\pm 0.46}$	$96.02_{\pm 0.33}$	$95.24_{\pm 0.41}$	$94.03_{\pm 0.53}$
MoRe	$90.03_{\pm 0.49}$	$94.60_{\pm 0.41}$	$\mathbf{96.97}_{\pm 0.35}$	$\mathbf{95.41}_{\pm 0.64}$	$95.01_{\pm 0.79}$	$96.44_{\pm 0.38}$	$94.74_{\pm 0.51}$
MoA	$89.51_{\pm 0.27}$	$94.14_{\pm 0.33}$	$96.18_{\pm 0.51}$	$94.94_{\pm 0.19}$	$96.48_{\pm 0.31}$	$95.52_{\pm 0.73}$	$94.46_{\pm 0.39}$
MJLoRA	$89.68_{\pm 0.52}$	$93.46_{\pm 0.66}$	$93.88_{\pm 0.25}$	$93.83_{\pm 0.49}$	$94.64_{\pm 0.41}$	$94.83_{\pm 0.33}$	$93.39_{\pm 0.44}$
MJAdaLoRA	$\mathbf{90.52}_{\pm 0.34}$	$93.78_{\pm 0.43}$	$94.69_{\pm 0.53}$	$95.39_{\pm 0.64}$	$94.02_{\pm 0.38}$	$94.61_{\pm 0.50}$	$93.84_{\pm 0.47}$
MJPropulsion	$89.04_{\pm 0.74}$	$93.77_{\pm 0.53}$	$95.61_{\pm 0.69}$	$94.30_{\pm 0.58}$	$94.00_{\pm 0.86}$	$94.34_{\pm 0.62}$	$93.51_{\pm 0.67}$
MJLoRAFA	$90.01_{\pm 0.62}$	$94.02_{\pm 0.47}$	$96.25_{\pm 0.34}$	$94.07_{\pm 0.95}$	$94.00_{\pm 0.78}$	$95.82_{\pm 0.71}$	$94.03_{\pm 0.65}$

Table 7: Results on **image classification benchmarks**. Each dataset consists of images annotated with a single class label. Highlighted rows denote our MJ variants.

Method	ChartQA	OKVQA	ScienceQA	SeedBench	Recognition	TextVQA	VizWizVQA	VQA-RAD	Avg
LoRA	$76.38_{\pm 1.67}$	$57.12_{\pm 2.31}$	$81.51_{\pm 1.96}$	$71.90_{\pm 1.23}$	$95.18_{\pm 1.84}$	$66.07_{\pm 2.13}$	$51.48_{\pm 2.28}$	$72.54_{\pm 2.35}$	$71.52_{\pm 1.97}$
AdaLoRA	$75.77_{\pm 1.78}$	$56.41_{\pm 2.02}$	$80.34_{\pm 2.24}$	$71.20_{\pm 1.51}$	$96.49_{\pm 1.89}$	$65.94_{\pm 2.31}$	$50.54_{\pm 1.37}$	$71.65_{\pm 2.28}$	$71.04_{\pm 1.93}$
Propulsion	$75.64_{\pm 1.33}$	$56.74_{\pm 1.57}$	$80.51_{\pm 2.48}$	$71.50_{\pm 2.29}$	$96.66_{\pm 1.61}$	$66.03_{\pm 2.16}$	$50.91_{\pm 1.35}$	$71.92_{\pm 1.98}$	$71.24_{\pm 1.85}$
LoRAFA	$75.82_{\pm 2.41}$	$56.32_{\pm 1.84}$	$80.60_{\pm 2.11}$	$71.14_{\pm 1.32}$	$96.66_{\pm 2.23}$	$65.90_{\pm 1.76}$	$50.44_{\pm 1.49}$	$71.61_{\pm 2.08}$	$71.06_{\pm 1.91}$
MoELoRA	$76.54_{\pm 1.52}$	$\mathbf{58.95}_{\pm 2.08}$	$81.12_{\pm 1.44}$	$72.06_{\pm 1.76}$	$95.02_{\pm 1.95}$	$66.44_{\pm 1.61}$	$51.77_{\pm 1.88}$	$72.44_{\pm 2.02}$	$71.79_{\pm 1.78}$
MixLoRA	$77.31_{\pm 1.85}$	$57.98_{\pm 1.49}$	$82.11_{\pm 2.16}$	$72.87_{\pm 1.72}$	$\mathbf{96.71}_{\pm 1.58}$	$67.22_{\pm 1.34}$	$51.63_{\pm 1.47}$	$73.44_{\pm 1.91}$	$72.41_{\pm 1.69}$
HydraLoRA	$76.64_{\pm 1.43}$	$58.34_{\pm 2.24}$	$\mathbf{82.79}_{\pm 1.68}$	$70.53_{\pm 1.36}$	$95.14_{\pm 2.12}$	$\mathbf{67.61}_{\pm 1.51}$	$51.06_{\pm 2.29}$	$72.68_{\pm 1.74}$	$71.85_{\pm 1.80}$
MoLA	$77.42_{\pm 1.74}$	$57.89_{\pm 1.58}$	$82.78_{\pm 1.89}$	$73.56_{\pm 1.31}$	$96.11_{\pm 1.27}$	$66.98_{\pm 1.46}$	$52.14_{\pm 2.42}$	$73.44_{\pm 2.13}$	$\mathbf{72.54}_{\pm 1.73}$
MoRe	$\mathbf{77.52}_{\pm 1.79}$	$57.84_{\pm 1.87}$	$82.05_{\pm 1.28}$	$\mathbf{73.58}_{\pm 1.64}$	$96.02_{\pm 2.01}$	$67.11_{\pm 1.73}$	$51.74_{\pm 1.84}$	$73.91_{\pm 1.32}$	$72.47_{\pm 1.69}$
MoA	$76.48_{\pm 2.04}$	$57.11_{\pm 1.69}$	$81.53_{\pm 1.81}$	$72.00_{\pm 1.39}$	$95.33_{\pm 1.48}$	$66.58_{\pm 2.07}$	$51.47_{\pm 2.22}$	$72.91_{\pm 1.26}$	$71.68_{\pm 1.75}$
MJLoRA	$76.94_{\pm 1.92}$	$57.43_{\pm 1.79}$	$81.73_{\pm 2.04}$	$72.43_{\pm 1.68}$	$95.79_{\pm 1.55}$	$67.42_{\pm 2.33}$	$51.32_{\pm 1.28}$	$73.00_{\pm 2.07}$	$72.01_{\pm 1.83}$
MJAdaLoRA	$76.04_{\pm 1.66}$	$56.18_{\pm 1.95}$	$80.58_{\pm 2.28}$	$71.38_{\pm 2.19}$	$95.48_{\pm 2.01}$	$65.28_{\pm 1.24}$	$\mathbf{52.66}_{\pm 1.56}$	$\mathbf{73.92}_{\pm 1.71}$	$71.44_{\pm 1.83}$
MJPropulsion	$77.24_{\pm 1.81}$	$58.31_{\pm 1.66}$	$82.01_{\pm 1.39}$	$72.78_{\pm 1.58}$	$96.61_{\pm 2.32}$	$67.19_{\pm 1.87}$	$52.22_{\pm 2.25}$	$73.44_{\pm 1.43}$	$72.48_{\pm 1.79}$
MJLoRAFA	$76.31_{\pm 1.97}$	$57.19_{\pm 1.31}$	$81.04_{\pm 2.03}$	$71.81_{\pm 1.82}$	$95.14_{\pm 1.46}$	$66.26_{\pm 1.98}$	$51.02_{\pm 2.45}$	$72.29_{\pm 1.65}$	$71.38_{\pm 1.83}$

Table 8: Results on vision–language question answering benchmarks. Highlighted rows denote our MJ variants.

Method	ActSeq	ActPred	ActAnt	FineAct	UnexpAct	ObjExist	ObjInter	ObjShuffle	Avg
LoRA	$40.83_{\pm 1.19}$	$42.39_{\pm 0.89}$	$53.74_{\pm 1.42}$	$39.41_{\pm 1.29}$	$41.57_{\pm 0.95}$	$52.28_{\pm 1.22}$	$41.44_{\pm 1.04}$	$54.91_{\pm 1.36}$	$45.82_{\pm 1.17}$
AdaLoRA	$40.27_{\pm 1.21}$	$41.93_{\pm 0.97}$	$53.21_{\pm 1.48}$	$39.04_{\pm 1.37}$	$41.05_{\pm 1.14}$	$51.66_{\pm 1.35}$	$40.88_{\pm 0.95}$	$54.11_{\pm 1.18}$	$45.27_{\pm 1.21}$
Propulsion	$40.19_{\pm 1.36}$	$41.88_{\pm 1.13}$	$53.08_{\pm 0.96}$	$38.92_{\pm 1.41}$	$40.87_{\pm 1.03}$	$51.41_{\pm 1.24}$	$40.61_{\pm 1.33}$	$53.82_{\pm 1.08}$	$45.10_{\pm 1.19}$
LoRAFA	$40.11_{\pm 1.28}$	$41.74_{\pm 1.41}$	$53.09_{\pm 1.06}$	$38.96_{\pm 1.18}$	$40.88_{\pm 1.39}$	$51.52_{\pm 0.97}$	$40.73_{\pm 1.31}$	$53.97_{\pm 1.12}$	$45.13_{\pm 1.22}$
MoELoRA	$40.91_{\pm 1.33}$	$42.48_{\pm 1.06}$	$53.82_{\pm 1.41}$	$39.44_{\pm 0.92}$	$41.75_{\pm 1.24}$	$52.46_{\pm 1.18}$	$41.53_{\pm 1.09}$	$54.92_{\pm 1.38}$	$45.91_{\pm 1.20}$
MixLoRA	$41.98_{\pm 0.98}$	$43.91_{\pm 1.27}$	$\mathbf{56.04}_{\pm 1.02}$	$40.76_{\pm 1.13}$	$42.89_{\pm 1.07}$	$53.84_{\pm 1.36}$	$42.78_{\pm 0.85}$	$56.28_{\pm 1.21}$	$47.31_{\pm 1.11}$
HydraLoRA	$42.37_{\pm 0.91}$	$\mathbf{44.72}_{\pm 1.03}$	$55.42_{\pm 1.37}$	$\mathbf{41.82}_{\pm 1.16}$	$43.33_{\pm 1.42}$	$\mathbf{55.03}_{\pm 1.10}$	$43.22_{\pm 1.23}$	$56.89_{\pm 0.96}$	$47.85_{\pm 1.15}$
MoLA	$40.88_{\pm 1.44}$	$42.31_{\pm 1.19}$	$53.61_{\pm 1.11}$	$39.52_{\pm 1.32}$	$41.64_{\pm 0.88}$	$52.19_{\pm 1.05}$	$41.32_{\pm 1.46}$	$54.73_{\pm 1.12}$	$45.78_{\pm 1.20}$
MoRe	$42.18_{\pm 1.05}$	$44.38_{\pm 1.13}$	$55.71_{\pm 1.18}$	$41.44_{\pm 0.97}$	$43.58_{\pm 1.22}$	$55.12_{\pm 1.08}$	$43.42_{\pm 1.16}$	$56.61_{\pm 1.21}$	$47.81_{\pm 1.13}$
MoA	$41.22_{\pm 0.89}$	$42.86_{\pm 1.33}$	$54.48_{\pm 1.19}$	$40.03_{\pm 0.99}$	$42.04_{\pm 1.31}$	$52.91_{\pm 1.07}$	$41.88_{\pm 1.38}$	$55.37_{\pm 0.85}$	$46.35_{\pm 1.13}$
MJLoRA	$41.64_{\pm 1.11}$	$43.20_{\pm 1.02}$	$56.08_{\pm 1.24}$	$40.12_{\pm 0.91}$	$43.94_{\pm 1.19}$	$53.20_{\pm 0.98}$	$42.11_{\pm 1.34}$	$55.64_{\pm 1.09}$	$46.99_{\pm 1.11}$
MJAdaLoRA	$\mathbf{42.88}_{\pm 1.00}$	$44.51_{\pm 1.26}$	$55.83_{\pm 1.12}$	$41.59_{\pm 0.90}$	$43.72_{\pm 1.37}$	$54.81_{\pm 1.02}$	$\mathbf{43.74}_{\pm 1.23}$	$\mathbf{57.36}_{\pm 1.43}$	$\mathbf{48.06}_{\pm 1.17}$
MJPropulsion	$42.36_{\pm 1.38}$	$43.72_{\pm 1.08}$	$55.47_{\pm 1.29}$	$40.82_{\pm 1.03}$	$43.29_{\pm 1.35}$	$53.91_{\pm 1.14}$	$43.19_{\pm 0.99}$	$56.93_{\pm 1.40}$	$47.46_{\pm 1.21}$
MJLoRAFA	$42.11_{\pm 1.01}$	$43.92_{\pm 1.39}$	$55.04_{\pm 0.87}$	$40.98_{\pm 1.28}$	$\mathbf{43.95}_{\pm 1.21}$	$54.02_{\pm 1.34}$	$42.89_{\pm 0.96}$	$56.41_{\pm 1.42}$	$47.42_{\pm 1.19}$

Table 9: (Part 2) Results on **action and object-centric reasoning subtasks**. Highlighted rows denote our MJ variants.

Method	MoveDir	ActLoc	SceneTrans	ActCount	MoveCount	MoveAttr	Avg
LoRA	$47.62_{\pm1.09}$	$54.01_{\pm1.41}$	$60.32_{\pm0.85}$	$58.62_{\pm1.30}$	$48.39_{\pm0.96}$	$47.11_{\pm1.33}$	$52.68_{\pm1.16}$
AdaLoRA	$46.72_{\pm1.11}$	$53.12_{\pm1.46}$	$59.63_{\pm0.91}$	$57.92_{\pm1.16}$	$47.64_{\pm1.38}$	$46.31_{\pm1.02}$	$51.89_{\pm1.17}$
Propulsion	$46.41_{\pm1.43}$	$52.84_{\pm0.89}$	$59.44_{\pm1.14}$	$57.66_{\pm0.95}$	$47.21_{\pm1.24}$	$45.98_{\pm1.37}$	$51.59_{\pm1.17}$
LoRAFA	$46.53_{\pm0.93}$	$52.91_{\pm1.39}$	$59.61_{\pm1.01}$	$57.83_{\pm1.17}$	$47.31_{\pm1.36}$	$46.02_{\pm0.98}$	$51.70_{\pm1.14}$
MoELoRA	$47.85_{\pm0.97}$	$54.21_{\pm1.16}$	$60.55_{\pm0.88}$	$58.83_{\pm1.41}$	$48.61_{\pm0.92}$	$47.20_{\pm1.08}$	$52.88_{\pm1.07}$
MixLoRA	$49.02_{\pm1.12}$	$55.41_{\pm1.44}$	$62.01_{\pm0.94}$	$60.23_{\pm1.29}$	$49.94_{\pm1.21}$	$48.41_{\pm0.89}$	$54.17_{\pm1.15}$
HydraLoRA	$49.51_{\pm1.38}$	$\mathbf{56.51}_{\pm0.96}$	$63.18_{\pm1.33}$	$60.79_{\pm1.05}$	$50.89_{\pm1.42}$	$48.87_{\pm1.11}$	$54.96_{\pm1.21}$
MoLA	$47.41_{\pm1.03}$	$53.94_{\pm1.31}$	$60.14_{\pm1.09}$	$58.41_{\pm0.94}$	$48.12_{\pm1.23}$	$46.89_{\pm0.90}$	$52.49_{\pm1.08}$
MoRe	$49.33_{\pm1.08}$	$55.52_{\pm1.19}$	$62.14_{\pm0.86}$	$60.46_{\pm1.35}$	$\mathbf{51.01}_{\pm0.99}$	$\mathbf{49.38}_{\pm1.26}$	$54.64_{\pm1.12}$
MoA	$48.11_{\pm1.34}$	$54.47_{\pm0.95}$	$61.01_{\pm1.12}$	$59.28_{\pm1.31}$	$48.89_{\pm0.87}$	$47.61_{\pm1.40}$	$53.23_{\pm1.17}$
MJLoRA	$49.31_{\pm1.07}$	$55.92_{\pm1.12}$	$\mathbf{63.78}_{\pm1.19}$	$60.97_{\pm1.21}$	$50.82_{\pm1.48}$	$49.31_{\pm1.24}$	$\mathbf{55.02}_{\pm1.22}$
MJAdaLoRA	$\mathbf{49.88}_{\pm1.15}$	$54.27_{\pm0.94}$	$62.91_{\pm1.34}$	$\mathbf{61.14}_{\pm1.08}$	$49.61_{\pm1.42}$	$49.12_{\pm1.19}$	$54.49_{\pm1.19}$
MJPropulsion	$49.47_{\pm0.91}$	$56.48_{\pm1.26}$	$62.58_{\pm1.11}$	$60.84_{\pm1.39}$	$50.86_{\pm1.02}$	$48.74_{\pm1.31}$	$54.83_{\pm1.17}$
MJLoRAFA	$48.38_{\pm1.40}$	$54.83_{\pm1.01}$	$63.14_{\pm1.23}$	$59.49_{\pm1.13}$	$49.18_{\pm0.82}$	$\mathbf{49.38}_{\pm1.29}$	$54.07_{\pm1.15}$

Table 10: Results on motion and scene understanding subtasks.

Method	StateChg	CharOrd	EgoNav	EpisReason	CounterFact	Avg
LoRA	$59.94_{\pm1.11}$	$57.39_{\pm1.33}$	$58.92_{\pm1.01}$	$61.29_{\pm1.19}$	$45.61_{\pm1.45}$	$56.63_{\pm1.22}$
AdaLoRA	$59.12_{\pm1.29}$	$56.43_{\pm1.09}$	$58.12_{\pm1.47}$	$60.54_{\pm0.98}$	$44.96_{\pm1.36}$	$55.83_{\pm1.24}$
Propulsion	$58.83_{\pm0.95}$	$56.11_{\pm1.24}$	$57.89_{\pm1.41}$	$60.21_{\pm0.84}$	$44.72_{\pm1.27}$	$55.55_{\pm1.14}$
LoRAFA	$58.96_{\pm1.26}$	$56.29_{\pm1.28}$	$58.11_{\pm1.34}$	$60.48_{\pm1.07}$	$44.83_{\pm1.48}$	$55.73_{\pm1.29}$
MoELoRA	$60.03_{\pm1.08}$	$57.66_{\pm1.31}$	$59.24_{\pm0.82}$	$61.53_{\pm1.14}$	$45.71_{\pm1.37}$	$56.83_{\pm1.14}$
MixLoRA	$61.55_{\pm0.86}$	$59.08_{\pm1.23}$	$60.71_{\pm1.29}$	$\mathbf{64.08}_{\pm1.02}$	$47.12_{\pm1.44}$	$58.51_{\pm1.17}$
HydraLoRA	$61.98_{\pm1.27}$	$\mathbf{60.27}_{\pm0.94}$	$\mathbf{61.96}_{\pm1.11}$	$63.36_{\pm1.36}$	$47.69_{\pm1.03}$	$59.05_{\pm1.14}$
MoLA	$62.14_{\pm0.95}$	$60.31_{\pm1.13}$	$61.58_{\pm1.21}$	$63.52_{\pm1.18}$	$47.94_{\pm1.12}$	$59.10_{\pm1.12}$
MoRe	$61.71_{\pm1.13}$	$59.26_{\pm1.46}$	$60.98_{\pm0.99}$	$63.05_{\pm1.21}$	$47.43_{\pm1.34}$	$58.49_{\pm1.23}$
MoA	$60.29_{\pm1.04}$	$58.03_{\pm1.25}$	$59.49_{\pm1.28}$	$61.88_{\pm0.89}$	$46.12_{\pm1.12}$	$57.16_{\pm1.12}$
MJLoRA	$60.74_{\pm1.02}$	$58.32_{\pm0.87}$	$62.04_{\pm1.42}$	$62.19_{\pm0.96}$	$46.43_{\pm1.21}$	$57.94_{\pm1.10}$
MJAdaLoRA	$\mathbf{62.61}_{\pm0.89}$	$58.02_{\pm1.32}$	$60.74_{\pm0.98}$	$63.78_{\pm1.37}$	$\mathbf{48.22}_{\pm1.09}$	$58.67_{\pm1.13}$
MJPropulsion	$61.99_{\pm0.91}$	$60.18_{\pm1.26}$	$61.12_{\pm1.12}$	$63.98_{\pm1.31}$	$47.81_{\pm1.04}$	$\mathbf{59.02}_{\pm1.13}$
MJLoRAFA	$59.71_{\pm1.42}$	$57.01_{\pm1.18}$	$58.74_{\pm1.06}$	$61.02_{\pm1.33}$	$45.38_{\pm0.91}$	$56.37_{\pm1.18}$

Table 11: Results on high-level reasoning subtasks.

E.1 Statistical Significance Analysis

A natural question is whether MJ's accuracy gains over baselines are statistically meaningful or fall within run-to-run variance. We emphasize that MJ's primary claim is *efficiency*: it matches MoE-PEFT accuracy while using 7–29× fewer trainable parameters, up to 48% lower memory, and 1.5–2× faster training (§5.1). The significance analysis below confirms that MJ's accuracy is genuinely competitive with the strongest MoE-PEFT methods rather than merely comparable on average.

For each benchmark family, we conduct two-tailed paired t-tests comparing the best-performing MJ variant against each PEFT and MoE-PEFT baseline. Each paired difference is the per-task accuracy gap (MJ − baseline), pairing baselines and MJ on the same task. We test across N tasks within a family ($N = 6$ for GLUE, $N = 8$ for Commonsense & QA and Vision–Language QA), giving $\mathrm{df} = N - 1$. We report the mean difference (ΔMean), the paired t-statistic, and the two-tailed p-value, marking significance at $p < 0.05$.

Across all three families, MJ variants significantly improve over their underlying PEFT methods and other standard PEFT baselines (Tables 12, 13, 14). For example, on GLUE MJLoRA significantly beats LoRA-FA (t=13.4, p<0.001); on Commonsense & QA, MJLoRA-FA significantly outperforms every standard PEFT method ($p < 0.01$ in all cases); and on Vision–Language QA, MJPropulsion significantly exceeds LoRA, LoRA-FA, AdaLoRA, and Propulsion ($p < 0.001$). These gains confirm that gradient-free routing provides genuine specialization over uniform adapter application, at no additional parameter cost.

MJ Method	Baseline	N	ΔMean	t	p-value
MJLoRA	LoRA	5	+0.328	2.031	0.098
	LoRA-FA	5	+0.857	13.437	$< 0.001^{\dagger}$
	MoRE	5	+0.188	0.785	0.468
	MoLA	5	+0.172	1.108	0.318
	MixLoRA	5	+0.537	8.678	$< 0.001^{\dagger}$
MJLoRA-FA	LoRA	5	+0.317	1.257	0.264
	LoRA-FA	5	+0.845	4.620	$0.006^{\dagger}$
	MoRE	5	+0.177	0.873	0.423
	MoLA	5	+0.160	1.214	0.279
	MixLoRA	5	+0.525	3.839	$0.012^{\dagger}$
	HydraLoRA	5	−0.012	−0.074	0.944

Table 12: Two-tailed paired t-tests on GLUE comparing MJ variants against PEFT and MoE-PEFT baselines (mean over 5 seeds). ΔMean is the average accuracy difference (MJ − baseline). † denotes statistical significance at $p < 0.05$. MJ significantly outperforms standard PEFT baselines (LoRA-FA) and several MoE-PEFT methods (MixLoRA), while differences against the strongest MoE-PEFT baselines (MoRE, MoLA, HydraLoRA) are not significant—confirming that MJ matches their accuracy despite using 7–29× fewer trainable parameters.

Against MoE-PEFT baselines, MJ significantly outperforms the weaker methods (e.g., MoELoRA, MoA, MixLoRA on several families) while showing *no significant difference* from the strongest baselines—HydraLoRA, MoRE, and MoLA. On GLUE, the differences between MJLoRA and MoRE/MoLA are not significant (p=0.468 and p=0.318); on Commonsense & QA, MJLoRA-FA is statistically indistinguishable from HydraLoRA and MoRE; and on Vision–Language QA, MJPropulsion shows no significant difference from MixLoRA, HydraLoRA, MoRE, or MoLA. Crucially, non-significance here is the desired outcome: it indicates that MJ achieves statistically equivalent accuracy to methods that use up to 29× more trainable parameters and a learned router.

The combined picture is consistent across modalities: MJ *significantly outperforms* standard PEFT

MJ Method	Baseline	N	ΔMean	t	p-value
MJLoRA	LoRA	8	+0.383	3.469	$0.010^{\dagger}$
	LoRA-FA	8	+0.878	6.941	$< 0.001^{\dagger}$
	MoELoRA	8	+0.424	1.834	0.109
	MixLoRA	8	+0.146	0.820	0.439
	MoLA	8	+0.436	2.432	$0.045^{\dagger}$
	MoRE	8	+0.114	0.483	0.644
	MoA	8	+0.275	2.567	$0.037^{\dagger}$
	HydraLoRA	8	−0.000	−0.000	1.000
MJLoRA-FA	LoRA	8	+0.598	3.676	$0.008^{\dagger}$
	LoRA-FA	8	+1.093	11.226	$< 0.001^{\dagger}$
	AdaLoRA	8	+0.936	7.209	$< 0.001^{\dagger}$
	Propulsion	8	+0.982	8.683	$< 0.001^{\dagger}$
	MoELoRA	8	+0.639	3.934	$0.006^{\dagger}$
	MixLoRA	8	+0.361	2.474	$0.043^{\dagger}$
	MoLA	8	+0.651	4.601	$0.002^{\dagger}$
	MoA	8	+0.490	3.557	$0.009^{\dagger}$
	MoRE	8	+0.329	1.519	0.173
	HydraLoRA	8	+0.215	1.290	0.238

Table 13: Two-tailed paired t-tests across the eight Commonsense & QA tasks (BoolQ, PIQA, SIQA, HellaSwag, WinoGrande, ARC-e, ARC-c, OBQA) comparing MJ variants against PEFT and MoE-PEFT baselines. Each paired difference is the per-task accuracy gap (MJ − baseline), $N = 8$ tasks, $\mathrm{df} = 7$. † marks significance at $p < 0.05$. MJ significantly outperforms all standard PEFT baselines and most MoE-PEFT methods (MixLoRA, MoLA, MoA), while differences against the strongest MoE-PEFT baselines (HydraLoRA, MoRE) are not significant—confirming MJ matches their accuracy using 7–29× fewer trainable parameters.

MJ Method	Baseline	N	ΔMean	t	p-value
MJPropulsion	LoRA	8	+0.952	9.368	$< 0.001^{\dagger}$
	LoRA-FA	8	+1.414	6.268	$< 0.001^{\dagger}$
	AdaLoRA	8	+1.433	7.157	$< 0.001^{\dagger}$
	Propulsion	8	+1.236	6.443	$< 0.001^{\dagger}$
	MoELoRA	8	+0.682	3.065	$0.018^{\dagger}$
	MoA	8	+0.799	7.677	$< 0.001^{\dagger}$
	MixLoRA	8	+0.066	0.734	0.487
	HydraLoRA	8	+0.626	1.752	0.123
	MoRE	8	+0.004	0.021	0.984
	MoLA	8	−0.065	−0.376	0.718

Table 14: Two-tailed paired t-tests across the eight Vision–Language QA tasks (ChartQA, OK-VQA, ScienceQA, SEED-Bench, Recognition, TextVQA, VizWiz-VQA, VQA-RAD) comparing MJPropulsion against PEFT and MoE-PEFT baselines. Each paired difference is the per-task accuracy gap (MJ − baseline), $N = 8$ tasks, $\mathrm{df} = 7$. † marks significance at $p < 0.05$. MJPropulsion significantly outperforms all standard PEFT baselines and weaker MoE-PEFT methods (MoELoRA, MoA), while showing no significant difference from the strongest MoE-PEFT baselines (MixLoRA, HydraLoRA, MoRE, MoLA)—confirming that MJ matches their accuracy using 7–29× fewer trainable parameters.

and weaker MoE-PEFT methods, and *statistically matches* the strongest MoE-PEFT baselines while preserving the strict parameter efficiency of standard PEFT. We note that with $N = 6$–8 tasks per family, these tests have limited power to detect very small differences; accordingly, we interpret non-significant results as "no detectable difference" rather than exact equivalence. This is sufficient to support our central claim—that MJ delivers MoE-PEFT-level accuracy at PEFT-level cost.

F Extended Related Work

Research on efficient adaptation of large language models has developed along three major directions: parameter-efficient fine-tuning (PEFT), sparse mixture-of-experts (MoE) architectures, and MoE-enhanced PEFT methods. While each direction has advanced scalability and specialization, existing approaches either rely on static adapter structures or introduce additional routing parameters and multi-expert computation, leaving room for more lightweight and input-adaptive designs.

F.1 Parameter-Efficient Fine-Tuning

The high computational cost of full-parameter fine-tuning (Devlin et al., 2019; Brown et al., 2020) has led to numerous PEFT approaches that freeze the backbone and update only small modules. Representative methods include Adapter tuning (Houlsby et al., 2019), BitFit (Zaken et al., 2022), Prompt Tuning (Lester et al., 2021), and Prefix Tuning (Li and Liang, 2021). Among these, LoRA (Hu et al., 2022) has become the most widely adopted due to its strong empirical performance and minimal parameter footprint. Several LoRA variants further improve efficiency: AdaLoRA (Zhang et al., 2023b) adjusts ranks using importance scores, DyLoRA (Valipour et al., 2023) trains multiple ranks jointly, VeRA (Kopiczko et al., 2023) freezes low-rank matrices and learns only scaling vectors, and LoRA+ (Hayou et al., 2024) improves optimization stability. Quantized LoRA and tensor-train decomposed adapters (Dettmers et al., 2023) further reduce memory consumption by enabling efficient fine-tuning of low-precision LLMs.

Despite their successes, nearly all PEFT methods employ *static*, input-agnostic adapters. The same adapter configuration is applied across all layers and inputs, regardless of task or domain. This static design limits fine-grained specialization and can

cause interference when diverse tasks share a single update mechanism.

F.2 Sparse Mixture-of-Experts

Mixture-of-Experts architectures (Jacobs et al., 1991) introduce multiple expert subnetworks and a routing mechanism that selects which experts process each token. Large-scale systems such as GShard (Lepikhin et al., 2020), Switch Transformer (Fedus et al., 2022), GLaM (Du et al., 2022), Mixtral (Jiang et al., 2024), and DeepSeek-MoE (Dai et al., 2024) demonstrate that sparse activation enables scaling to hundreds of billions of parameters without proportional increases in computation. Routing strategies include top-k gating (Shazeer et al., 2017), expert-choice routing (Zhou et al., 2022), top-p routing (Huang et al., 2024), dynamic-k routing (Guo et al., 2024), and differentiable or soft gating mechanisms (Puigcerver et al., 2023).

However, sparse MoE systems rely on trainable routing networks, multiple active experts per layer, and auxiliary balancing losses. These components increase inference latency, activation memory, and optimization complexity. As a result, conventional MoE architectures, though powerful in pre-training, are not well suited for parameter-efficient downstream fine-tuning.

F.3 MoE-Enhanced PEFT

To combine LoRA's efficiency with MoE-style specialization, recent work introduces multiple LoRA experts per layer along with a routing mechanism. LoRAMoE (Dou et al., 2024) partitions experts into world-knowledge and task-specific modules; MoELoRA (Luo et al., 2024) routes inputs among LoRA experts for multi-task performance; and MixLoRA (Li et al., 2024b) employs sparse top-k routing with load-balancing losses. Other methods such as MoCLE (Gou et al., 2023) and LLaVA-MoLE (Chen et al., 2024) activate LoRA experts based on clustered instructions or domains. More advanced designs—including HMoRA (Liao et al., 2025), LD-MoLE (Zhuang et al., 2025), and MoRAL (Yang et al., 2024)—use hierarchical or differentiable routing to support dynamic expert selection.

Although effective, these systems share three recurring limitations: (i) they add *extra trainable routing parameters*, (ii) they activate *multiple experts per layer*, increasing activation memory and inference latency, and (iii) they require *complex routing optimization*, often involving auxiliary balancing losses. Thus, while MoE-enhanced PEFT improves specialization, it compromises the simplicity and strict efficiency that originally motivated PEFT methods.

F.4 Positioning of Monkey Jump

Across PEFT, MoE, and hybrid approaches, a consistent gap remains: existing methods are either static and input-invariant or rely on routing networks that significantly increase parameter count and computation. MoE-based PEFT methods route among multiple LoRA experts but introduce routing parameters and activate several experts per layer, making deployed models computationally intensive.

Monkey Jump addresses this gap by treating existing PEFT adapters as implicit experts and routing among them using trainable parameter-free clustering. This achieves MoE-style specialization without additional trainable parameters, without multi-expert activation overhead, and without auxiliary balancing losses—preserving the strict efficiency of standard PEFT while enabling input-adaptive behavior.

G Datasets

We evaluate MJ on a large-scale multi-task benchmark covering text, image, and video modalities. The full benchmark contains 154,820 training samples across 47 test sets. Table 15 provides the complete breakdown.

Text. The text benchmark consists of 14 datasets with 98,970 training samples, spanning natural language understanding and reasoning tasks.

From the GLUE benchmark (Wang et al., 2018) (28,668 samples), we include: sentiment classification (SST-2), natural language inference (QNLI), paraphrase detection (QQP and MRPC), linguistic acceptability (CoLA), and semantic similarity (STS-B).

For commonsense and reasoning tasks, we sample 70,302 training examples from the 170K training set of Hu et al. (2023). This includes PIQA (Bisk et al., 2020), Social IQA (Sap et al., 2019), WinoGrande (Sakaguchi et al., 2021), HellaSwag (Zellers et al., 2019), ARC-Easy, ARC-Challenge (Clark et al., 2018), OpenBookQA (Mihaylov et al., 2018), and BoolQ (Clark et al., 2019). We use the same test sets as Hu et al. (2023) for evaluation.

Image. The image benchmark consists of 14 datasets (42,550 training samples) covering both visual question answering and image classification. VQA tasks include chart understanding (ChartQA (Masry et al., 2022)), text reading in images (TextVQA (Fang et al., 2023), Text Recognition), medical imaging (VQA-RAD (Lau et al., 2018)), visual knowledge (OK-VQA (Marino et al., 2019)), accessibility (VizWiz-VQA (Gurari et al., 2018)), scientific reasoning (ScienceQA (Lu et al., 2022)), and visual inference (SEED-Bench (Li et al., 2023)). For image classification, we follow VTAB-1K (Zhai et al., 2019) and include Caltech-101 (object recognition), Flowers-102 (fine-grained classification), Oxford Pets (species/breed recognition), Camelyon (medical histopathology), and EuroSAT (satellite imagery). To broaden coverage beyond VTAB-1K, we additionally include SVHN (street number recognition), Retinopathy (diabetic retinopathy detection), and KITTI-Dist (autonomous driving).

Video. The video benchmark leverages MVTamperBench (Agarwal et al., 2025), consisting of 19 tasks (13,300 training samples) designed to evaluate temporal and visual reasoning. These include action understanding (Action Sequence, Action Prediction, Action Antonym, Fine-grained Action, Unexpected Action, Action Localization, Action Count), object tracking (Object Existence, Object Interaction, Object Shuffle, Moving Direction, Moving Count, Moving Attribute), scene understanding (Scene Transition, State Change), and high-level reasoning (Character Order, Egocentric Navigation, Episodic Reasoning, Counterfactual). Each task contributes 500 training samples.

For all experiments, we train on the combined multi-task training set and evaluate on each task's held-out test set independently. This setup tests the model's ability to learn diverse tasks simultaneously while preserving task-specific performance—a challenging setting where MJ's routing mechanism enables natural specialization.

TEXT — Natural Language Understanding & Reasoning				*(Train: 98,970 samples)*	
Dataset	Samples	Dataset	Samples	Dataset	Samples
SST-2	872	ARC-Challenge	500	HellaSwag	1,000
QNLI	5,463	ARC-Easy	500	Social IQA	1,000
QQP	40,430	BoolQ	1,000	OpenBookQA	500
CoLA	1,043	PIQA	1,000	WinoGrande	1,000
MRPC	408	STS-B	1,500		
IMAGE — Visual Question Answering & Task Adaptation				*(Train: 42,550 samples)*	
Dataset	Samples	Dataset	Samples	Dataset	Samples
ChartQA	1,000	OK-VQA	841	ScienceQA	518
TextVQA	1,000	VizWiz-VQA	417	Text Recognition	1,000
VQA-RAD	200	SEED-Bench	500	Caltech-101	500
Flowers-102	500	EuroSAT	500	Pets	500
SVHN	500	Camelyon	500		
VIDEO — MVTamperBench				*(Train: 13,300 samples)*	
Dataset	Samples	Dataset	Samples	Dataset	Samples
Action Sequence	500	Object Shuffle	500	Moving Attribute	500
Action Prediction	500	Moving Direction	500	State Change	500
Action Antonym	500	Action Localization	500	Character Order	500
Fine-grained Action	500	Scene Transition	500	Ego. Navigation	500
Unexpected Action	500	Action Count	500	Episodic Reasoning	500
Object Existence	500	Moving Count	500	Counterfactual	500
Object Interaction	500				
Total: Train: 154,820 Test Sets: 47 Test Samples: 74,192					

Table 15: Multi-task benchmark. Training and test datasets organized by modality: Text, Image, and Video.

H Implementation Details

We implement MJ using the HuggingFace Transformers library (v4.50) (Wolf et al., 2019) with PyTorch 2.10+ (Paszke et al., 2019) and Accelerate for distributed training. All experiments are conducted on NVIDIA H100 GPUs with Python 3.11 and Ubuntu 22.04. Each experiment is repeated with 5 different random seeds, and we report mean $\pm$ standard deviation. Table 16 summarizes all hyperparameters.

Adapter configuration. We apply MJ to every Transformer block, targeting five projections: Q, K, V, O, and gate. We use LoRA adapters with rank $r = 2$ for text tasks and $r = 1$ for image/video tasks, as vision-language models require less adaptation capacity per projection. Adapter dropout is set to 0.05 and the scaling factor $\alpha = 5$. The O and gate projections are designated as shared adapters ($m_{t,e^*} = 1$ for all tokens), providing a stable global adaptation path that is always active regardless of routing decisions. The remaining projections (Q, K, V) participate in top-k routing, allowing token-specific specialization.

Routing configuration. Before training, we initialize routing centers using k-means clustering on 50,000 randomly sampled tokens from the training set. This provides a representative initialization that captures the natural clustering structure in the token representation space. During training, centers are updated via exponential moving average (EMA) with momentum $\beta = 0.5$ every 2 iterations. This update frequency balances computational overhead with center tracking accuracy. We stop updating centers after 5,000 iterations (approximately 50–70% of training), freezing them for the remainder of training to ensure stable routing decisions during final convergence. We use temperature $\tau = 1.0$ for the softmax routing distribution and select top-$k = 2$ adapters per token, meaning each token activates 2 out of 3 routed projections (Q, K, V) plus the 2 shared adapters (O, gate), for a total of 4 active adapters per token.

Training configuration. We use the AdamW optimizer with a base learning rate of 1×10^{-4} for LoRA and AdaLoRA variants. For LoRA-FA and Propulsion variants, we use a higher learning rate of 4×10^{-4} since these methods have fewer trainable parameters and benefit from larger updates. All methods use cosine learning rate decay with warmup ratio 0.1 (10% of total steps) and weight decay 0.1 for regularization. Training uses bf16 mixed precision throughout for memory efficiency.

For text tasks (GLUE, commonsense reasoning, QA), we train for 2 epochs with batch size 4 and gradient accumulation 4, yielding an effective batch size of 16. Maximum sequence length is set to 1,024 tokens. For image and video tasks (classification, VQA, video understanding), we train for 2 epochs with batch size 1 and gradient accumulation 8, yielding an effective batch size of 8. Maximum sequence length is extended to 2,048 to accommodate visual tokens from the vision encoder.

Baseline configuration. For fair comparison, all baseline methods (standard PEFT and MoE-PEFT) use identical optimization settings: same weight decay, warmup ratio, batch sizes, and number of epochs. MoE-PEFT baselines use 4 experts per projection with top-2 routing, matching our experimental setup from prior work (Luo et al., 2024; Tian et al., 2024). All methods target the same projections (Q, K, V, O, gate) and use the same LoRA rank and dropout settings.

Adapter Configuration	
LoRA rank r	2 (text), 1 (image/video)
LoRA dropout	0.05
LoRA α	5
Target modules	Q, K, V, O, gate
Shared adapter	O, gate (always active)
Applied layers	All Transformer blocks
Routing Configuration	
k-means init tokens	50,000
EMA momentum β	0.5
EMA update frequency	Every 2 iterations
EMA stop step	5,000
Temperature τ	1.0
Top-k	2
Optimization	
Optimizer	AdamW
Learning rate (LoRA, AdaLoRA)	1×10^{-4}
Learning rate (LoRA-FA, Propulsion)	4×10^{-4}
LR schedule	Cosine decay
Warmup ratio	0.1
Weight decay	0.1
Precision	bf16
Text Tasks	
Epochs	2
Batch size	4
Gradient accumulation	4
Effective batch size	16
Max sequence length	1,024
Image & Video Tasks	
Epochs	2
Batch size	1
Gradient accumulation	8
Effective batch size	8
Max sequence length	2,048

Table 16: Hyperparameters for MJ experiments across all modalities.

I Use of AI Assistants

We used large language model assistants for grammar checking, proofreading, and improving the clarity of writing. All scientific content, experimental design, methodology, and analysis are entirely the authors' original contributions.