Routing by Analogy: kNN-Augmented Expert Assignment for Mixture-of-Experts

Anonymous ACL submission

Abstract

Mixture-of-Experts (MoE) architectures scale large language models efficiently by employing a parametric "router" to dispatch tokens to a sparse subset of experts. Typically, this router is trained once and then frozen, rendering routing decisions brittle under distribution shifts. We address this limitation by introducing **kNN-MoE**, a retrieval-augmented routing framework that reuses locally optimal expert assignments from a memory of similar past cases. This memory is constructed offline by directly optimizing token-wise routing logits to maximize the likelihood on a reference set. Crucially, we use the average similarity of retrieved neighbors as a confidence-driven mixing coefficient, thus allowing the method to fall back to the frozen router when no relevant cases are found. Experiments show that kNN-MoE outperforms the zero-shot baseline and is competitive with computationally intensive supervised fine-tuning.

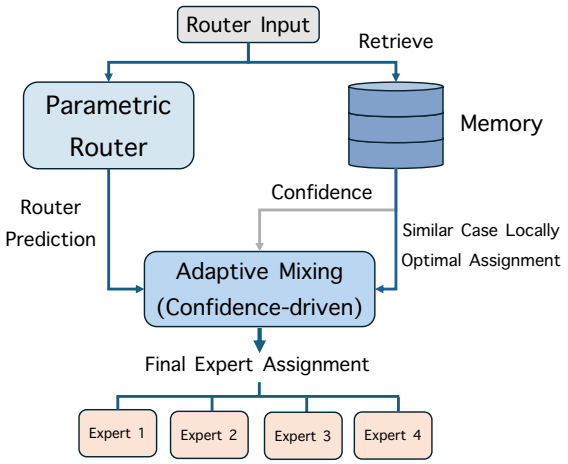

Figure 1: Schematic illustration of the **kNN-MoE** inference mechanism. Given a router input, the model simultaneously obtains a prediction from the frozen "Parametric Router" and retrieves similar past cases from the "Memory". The "Adaptive Mixing" module fuses these two signals based on a confidence score (derived from the average similarity of retrieved neighbors) to produce the final expert assignment.

1 Introduction

The scaling of Large Language Models (LLMs) has been significantly advanced by combining Transformers (Vaswani et al., 2017) with Mixture-of-Experts (MoE) architectures (Jacobs et al., 1991; Jordan and Jacobs, 1994; Shazeer et al., 2017; Fedus et al., 2022), which increase model capacity via sparse activation without a proportional increase in computational cost (Fedus et al., 2022). A routing network (or "router") dynamically activates a small subset of experts for each input token (Fedus et al., 2022; Jiang et al., 2024). However, the efficacy of an MoE model hinges critically on the quality of its routing decisions.

Standard routers are lightweight, parametric classifiers trained to predict expert assignments based on hidden states. Once trained, the router implements a fixed policy. While effective on in-distribution data, it lacks the flexibility to adjust routing decisions during inference. When facing inputs that deviate from the training distribution—often manifested as high perplexity—the router must extrapolate from fixed priors, potentially selecting suboptimal experts (Su et al., 2025; Li et al., 2025b,a). This lack of flexibility restricts the potential of MoE models, particularly on specialized or out-of-distribution tasks.

In this paper, we propose **kNN-MoE**, which refines routing decisions by retrieving the nearest neighbors (Dasarathy, 1991) of past router inputs and reusing their offline-computed locally optimal expert assignments (Figure 1). Concretely, we build a memory that stores router inputs (keys) paired with locally optimal expert assignments (values) obtained by maximizing the likelihood on a reference set. At test time, kNN-MoE retrieves similar keys from the memory whose expert assign-

ments led to higher past case likelihoods and then interpolates the router prediction with the retrieved expert assignment, using similarity as a confidence score. Intuitively, when retrieved neighbors are highly similar, the method trusts the retrieved routing; otherwise, it falls back to the original parametric router, avoiding noise from irrelevant retrievals.

We evaluated kNN-MoE on three MoE models—OLMoE (Muennighoff et al., 2024), GPT-OSS (OpenAI, 2025), and Qwen3 (Team, 2025)—across benchmarks including GPQA (Rein et al., 2024), SuperGPQA (Team et al., 2025), MMLU (Hendrycks et al., 2021), medical tasks (USMLE (Jin et al., 2020) and MedMCQA (Pal et al., 2022)) and a code task (MBPP (Austin et al., 2021)). Empirical results show that kNN-MoE outperforms the zero-shot baseline and achieves performance competitive with computationally intensive supervised fine-tuning, all without modifying the model parameters. Ultimately, kNN-MoE bridges the gap between static parametric routing and dynamic inference needs, offering a scalable pathway to enhance MoE adaptability without the computational burden of fine-tuning.

2 Related Work

2.1 Expert Assignment Optimization for MoE

Prior work has demonstrated that optimizing expert assignments in MoE can enhance model performance. Li et al. (2025a) proposed C3PO, a test-time adaptation method that fine-tunes routing logits using examples correctly answered by the model. However, this approach incurs a computational overhead approximately five times greater than that of the baseline (Li et al., 2025a). Li et al. (2025b) proposed fine-tuning the router via an auxiliary loss; unlike our non-parametric approach, which maintains a frozen model, their method requires permanently updating the router parameters. Su et al. (2025) introduced a test-time adaptation framework that dynamically optimizes expert assignments by minimizing loss on the input context. In contrast, kNN-MoE eliminates this online optimization overhead by directly retrieving cached locally optimal assignments from a memory store.

Notably, ERC loss (Lv et al., 2026), has incorporated nearest-neighbor concepts. Yet, this mechanism fundamentally differs from our approach: ERC loss computes distances between static router weights to bound injected noise during resource-intensive pre-training. Conversely, kNN-MoE operates as a compute-efficient, inference-time adaptation. By retrieving the nearest neighbors of continuous input representations from an external memory, it directly reuses locally optimal expert assignments without the need for parameter updates.

2.2 Non-Parametric and Retrieval-Augmented LMs

Our work is closely related to non-parametric methods that augment neural networks with external data stores. A prominent line of research involves kNN-LMs (Khandelwal et al., 2020; Xu et al., 2023), which retrieve nearest-neighbor tokens from a data store of cached hidden states to interpolate the final output distribution. Similarly, Retrieval-Augmented Generation models (Lewis et al., 2020) retrieve textual chunks to condition the generation process. Furthermore, Deguchi and Nagata (2025) explored retrieving past high-utility decisions in reranking to guide output selection.

3 Mixture-of-Experts

Consider an MoE Transformer (Fedus et al., 2022) model with L layers, where a subset of indices $\mathcal{L}_{\text{MoE}} \subset \{1, \ldots, L\}$ corresponds to MoE layers. For any specific layer $\ell \in \mathcal{L}_{\text{MoE}}$, the module consists of N expert networks $\{E_i^{(\ell)}\}_{i=1}^{N}$ and a router $R^{(\ell)}$.

Let $x^{(\ell)} \in \mathbb{R}^d$ denote the input hidden state of the MoE module at layer ℓ, which is typically the output of the preceding attention block or normalization module. The core mechanism of the MoE module employs a router to assign the input to the most relevant experts. The router is typically parameterized by a learnable weight matrix $W_r^{(\ell)} \in \mathbb{R}^{d \times N}$, and it generates the expert assignment (a sparse gating distribution) $a^{(\ell)}(x^{(\ell)}) \in \mathbb{R}^N$ via a TopK softmax function:

$$a^{(\ell)}(x^{(\ell)}) = \text{TopK}\left(\text{Softmax}(x^{(\ell)} W_r^{(\ell)})\right). \quad (1)$$

The final output of the MoE module, denoted as $h^{(\ell)}$, is computed as the linear combination of the selected experts' outputs using the predicted expert assignment:

$$h^{(\ell)} = \sum_{i=1}^{N} a^{(\ell)}(x^{(\ell)})_i \cdot E_i^{(\ell)}(x^{(\ell)}).$$

During standard inference, the router weights $W_r^{(\ell)}$ remain fixed. This static routing strategy may limit adaptability when the test distribution diverges from the training distribution.

4 Proposed Method: kNN-MoE

We propose **kNN-MoE** (Figure 1), a retrieval-based framework that enhances routing decisions by leveraging a memory of historical router inputs and their locally optimal expert assignments. We equip *each* MoE module $\ell \in \mathcal{L}_{\text{MoE}}$ with an independent memory store $\mathcal{M}^{(\ell)}$. We omit the layer superscript (ℓ) as the method operates identically across layers. kNN-MoE consists of two phases: (1) **Memory Construction** (§4.1), which builds the expert assignment memory offline; and (2) **Confidence-Aware Adaptive Mixing** (§4.2), which retrieves and utilizes stored expert assignments during inference.

4.1 Memory Construction

The goal of memory construction is to create a key-value store $\mathcal{M} = \{(k_t, v_t)\}$ for the layer, where the key k_t is a router input and the value v_t is the locally optimal expert assignment that outperforms the original router. This process involves two steps: data collection and deriving locally optimal expert assignments.

Data Collection We use a reference dataset $\mathcal{D}_{\text{ref}}$ consisting of sequences of ground-truth tokens. Let $\mathbf{y} = (y_1, \ldots, y_T)$ denote a sequence in $\mathcal{D}_{\text{ref}}$. We run the frozen MoE model on these sequences in teacher-forcing mode. For every time step $t \in [1, T]$, we collect the router input x_t at the current layer and the corresponding target token y_t from the sequence. The router input serves as the retrieval key, i.e., $k_t = x_t$.

Deriving Locally Optimal Expert Assignments Instead of storing the model's original routing decision defined in Eq. 1, we seek the locally optimal expert assignment that maximizes the prediction probability of the ground-truth token y_t. We formulate this as an optimization problem for each token.

Let $\pi(r) = \text{TopK}(\text{Softmax}(r))$ denote the mapping from logits $r \in \mathbb{R}^N$ to sparse expert weights. For a specific token t, we introduce a learnable logit vector r_t to replace the parametric output $x_t W_r$. We optimize r_t to minimize the negative log-likelihood of the target token y_t:

$$r_t^* = \arg\min_{r \in \mathbb{R}^N} \mathcal{L}_t(r), \tag{2}$$
$$\mathcal{L}_t(r) = -\log p_\theta\big(y_t \mid x_t, \text{routing}{=}\pi(r)\big),$$

where θ represents the frozen parameters of the rest of the network.

We solve Eq. 2 via gradient descent. We initialize r_t with the original parametric logits, i.e., $r_t^{(0)} = x_t W_r$. We then perform S steps of updates:

$$r_t^{(s+1)} = r_t^{(s)} - \eta \nabla_r \mathcal{L}_t(r_t^{(s)}), \text{ for } s = 0, \ldots, S-1,$$

where η is the learning rate. Note that while we describe r_t for a generic layer, this optimization is performed jointly across the entire model; specifically, the routing logits of all layers $\{r_t^{(\ell)}\}_{\ell \in \mathcal{L}_{\text{MoE}}}$ are updated simultaneously to minimize the loss. Additionally, the optimization and data collection processes are conducted independently across different sequences in $\mathcal{D}_{\text{ref}}$.

The final optimized routing logits are $r_t^{(S)}$. These logits serve as the target value in our memory, i.e., $v_t = r_t^{(S)}$. They represent the "(locally) optimal" continuous routing signals that best predict the next token.

Finally, the memory for the current layer is constructed as:

$$\mathcal{M} = \{(x_t, r_t^{(S)}) \mid t \in \mathcal{D}_{\text{ref}}\}.$$

This process is repeated for every layer in $\mathcal{L}_{\text{MoE}}$.

4.2 Confidence-Aware Adaptive Mixing

During inference, for each MoE layer and each router input x, we retrieve the set of K nearest neighbors $\mathcal{N}(x) \subset \mathcal{M}$ from the corresponding memory constructed in §4.1.

We formulate a non-parametric routing logit proposal, denoted as $r_{\text{mem}}(x)$, by aggregating the retrieved optimal logits according to their similarity to the input. Specifically, let $\{(k_j, v_j)\}_{j=1}^K$ be the retrieved key-value pairs in $\mathcal{N}(x)$. The memory-based logit proposal is computed as:

$$r_{\text{mem}}(x) = \sum_{j=1}^{K} \frac{s(x, k_j)}{\sum_{m=1}^{K} s(x, k_m)} v_j,$$

where $s(\cdot, \cdot)$ is a similarity function.

To fuse the parametric router's original logits, $r(x) = x W_r^{(\ell)}$, with the non-parametric proposal $r_{\text{mem}}(x)$, we introduce an adaptive mixing coefficient $\lambda(x)$ that reflects retrieval confidence. We quantify this confidence using the average similarity of the neighbors:

$$\lambda(x) = \frac{1}{K} \sum_{j=1}^{K} s(x, k_j).$$

The final routing logits are obtained by linearly interpolating between the parametric and memory-based logits. We then apply the TopK-Softmax function $\pi(\cdot)$ to generate the final sparse expert assignment:

$$r_{\text{final}}(x) = (1 - \lambda(x))r(x) + \lambda(x)r_{\text{mem}}(x), \quad (3)$$
$$a_{\text{final}}(x) = \pi(r_{\text{final}}(x)).$$

Finally, we use $a_{\text{final}}(x)$ to compute the MoE layer output h:

$$h = \sum_{i=1}^{N} a_{\text{final}}(x)_i \cdot E_i(x).$$

Performing the interpolation in the continuous logit space ensures that the final expert assignment strictly maintains the model's original Top-K sparsity constraint. Furthermore, this mixing ensures that when $\lambda(x) \approx 1$, the model trusts the retrieved routing signals to correct the router; conversely, when $\lambda(x) \approx 0$, it falls back to the parametric router to avoid retrieval noise.

5 Experiments

5.1 Experimental Settings

Datasets We evaluated kNN-MoE on general reasoning and domain-specific benchmarks: GPQA, SuperGPQA, MMLU, USMLE, MedMCQA, and MBPP. For each, we defined a disjoint test set $\mathcal{D}_{\text{test}}$ and a reference set $\mathcal{D}_{\text{ref}}$. The same $\mathcal{D}_{\text{ref}}$ was reused across all methods: kNN-MoE used it to construct the expert assignment memory, 5-shot (Brown et al., 2020) used it as the retrieval pool for in-context examples, and supervised fine-tuning (SFT) used it as the training dataset. Table 16 detailed the splits.

Models We employed three representative MoE models: OLMoE (`allenai/OLMoE-1B-7B-0125-Instruct`) (Muennighoff et al., 2024), GPT-OSS (`openai/gpt-oss-20b`) (OpenAI, 2025), and Qwen3 (`Qwen/Qwen3-30B-A3B-Instruct-2507`) (Team, 2025). These models were selected because they incorporated modern MoE architectural designs and were computationally feasible to run within our research budget.

Baselines We compared kNN-MoE against two categories of baselines. First, we included **Zero-shot** and **5-shot** as standard, cost-effective inference baselines. Second, and importantly, we compared against **SFT**. Since kNN-MoE leveraged the reference set $\mathcal{D}_{\text{ref}}$ to improve performance, SFT served as the primary methodological baseline: similar to kNN-MoE, it utilized the reference set $\mathcal{D}_{\text{ref}}$ to optimize performance in an *offline* stage (training), ensuring that the inference process remained efficient. This made SFT the most direct comparison for evaluating the efficacy of our offline memory construction.

- **Zero-shot:** Zero-shot used the original MoE model with its parametric routers $R^{(\ell)}$ and a zero-shot QA prompt.
- **5-shot:** For each test instance in $\mathcal{D}_{\text{test}}$, we retrieved five examples from the reference set $\mathcal{D}_{\text{ref}}$ whose input questions were most similar to the test question. Concretely, we encoded all questions using the `Qwen3-Embedding-0.6B` (Zhang et al., 2025) sentence embedding model and selected the five reference questions with the highest cosine similarity to the test question. These examples, together with their correct answers, were concatenated before the test question to form the in-context prompt.
- **SFT:** For SFT, we trained on $\mathcal{D}_{\text{ref}}$ using standard cross-entropy loss over the correct answers. We applied LoRA adapters (Hu et al., 2022) to all linear layers. We set the maximum number of training epochs to 3, split $\mathcal{D}_{\text{ref}}$ into an 85% training split and a 15% validation split, and selected the checkpoint with the lowest validation loss. During training and inference, SFT used a zero-shot QA prompt.
- **SFT (Router Only):** We fine-tuned only the router parameters $\{W_r^{(\ell)}\}_{\ell \in \mathcal{L}_{\text{MoE}}}$ on $\mathcal{D}_{\text{ref}}$, while freezing all expert and non-router parameters. The training objective was the same supervised cross-entropy loss as in SFT. We used the same training protocol (including training epochs, data splitting, and checkpoint selection methods) and prompt as SFT.

See Appendix A and B for details on prompt templates and SFT training, respectively.

Note on Other Methods We excluded token-level retrieval methods like kNN-LM (Khandelwal et al., 2020) from the main comparison due to the lack of efficient implementations compatible with modern LLM architectures. Furthermore, we deferred the comparison with C3PO (Li et al.,

Model	Method	GPQA	MMLU	SuperGPQA	USMLE	MedMCQA	MBPP	Avg.
OLMoE	Zero-shot	27.27	46.77	13.02	32.81	35.57	30.00	26.82
	5-shot	21.72	33.06	11.09	31.31	25.36	28.50	20.30
	SFT	21.72	46.89	**13.67**†	**37.84**†	34.78	31.60	27.26†
	SFT (Router Only)	24.24	45.27	11.66	31.22	34.23	29.60	25.41
	kNN-MoE	**29.80**	**47.81**†	13.27†	35.04†	**37.01**†	**32.20**	**27.53**†
	kNN-MoE (IVFPQ)	**29.80**	47.79†	13.14†	35.04†	36.98†	31.83	27.44†
GPT-OSS	Zero-shot	43.94	70.20	23.52	67.29	56.20	55.00	43.27
	5-shot	36.87	63.72	19.95	60.02	52.76	46.83	38.61
	SFT	41.92	70.28	**24.83**†	65.61	56.06	48.83	43.85†
	SFT (Router Only)	41.41	69.18	18.83	65.05	54.89	45.00	40.09
	kNN-MoE	**45.45**	70.28	24.35†	**68.31**†	57.06†	**56.50**	**43.88**†
	kNN-MoE (IVFPQ)	44.95	**70.30**	24.14†	68.13†	**57.14**†	56.17	43.77†
Qwen3	Zero-shot	41.41	78.59	34.85	75.30	62.51	59.80	52.93
	5-shot	44.44	**80.48**†	35.56	77.17†	66.58†	**62.33**	54.41†
	SFT	39.90	79.08	**40.20**†	**80.80**†	66.60†	55.33	**56.44**†
	SFT (Router Only)	43.94	78.36	33.15	76.05	66.03†	56.83	52.27
	kNN-MoE	**44.95**	78.86†	35.15	76.70†	66.65†	61.83	53.66†
	kNN-MoE (IVFPQ)	43.43	78.79	35.13	76.14	**66.70**†	61.83	53.61†

Table 1: Performance comparison across three MoE models. † denotes a statistically significant improvement ($p < 0.05$) over the Zero-shot baseline based on paired bootstrap resampling.

2025a) to Appendix C.1. The rationale was that C3PO operated as a *test-time adaptation* method, performing computationally intensive optimization *during* inference for each input. In contrast, kNN-MoE aligned with the SFT paradigm by shifting the computational burden to an *offline* memory construction, allowing for fast inference. Thus, we treated C3PO as a distinct category of method with different latency-accuracy trade-offs.

Implementation Details of kNN-MoE We used FAISS (Johnson et al., 2019) for retrieval and built one index per MoE layer, using the router inputs $\{x_t\}$ stored in that layer's memory $\mathcal{M}$ as keys. The naive implementation of kNN-MoE relies on FP16 index storage and exact distance search. To balance search accuracy and efficiency, we consider an optimized version based on Product Quantization (PQ) (Jégou et al., 2011), Inverted File (IVF) clustering, and dimension reduction. Specifically, we quantize the FP16 index to 8-bit, search across only 32 centroids during retrieval, and compress the hidden dimension to one-eighth. We refer to this variant as **kNN-MoE (IVFPQ)**. We used a single hyperparameter K (neighbors per router) and shared it across layers; by default, we set $K = 1$ based on validation (§6.3). We set the learning rate to $\eta = 2 \times 10^{-2}$ and used a single gradient descent step ($S = 1$) for each token. We found that using more steps (e.g., $S = 3$ or $S = 10$) yielded very similar performance while incurring a substantially higher construction cost; we provided an ablation over S in §6.4. The similarity function was an RBF kernel $s(x, k) = \exp(-\gamma\|x - k\|^2)$, with γ set heuristically based on the average nearest neighbor distance in the memory.

Evaluation We reported **accuracy** (%) on $\mathcal{D}_{\text{test}}$ for all datasets, using the official benchmark answers. To ensure a fair comparison, we kept the answer parsing identical across all methods. Additionally, we reported the overall accuracy (**Avg.**), which is calculated over a concatenated dataset comprising all individual test benchmarks. To assess statistical significance, we performed paired bootstrap resampling (Koehn, 2004) comparing all methods against the zero-shot baseline.

5.2 Results

Comparison with Zero-shot and SFT (Router Only) Across all models and datasets, kNN-MoE consistently improves over both Zero-shot and SFT (Router Only). These trends indicate that refining expert assignment via retrieved cases is more effective than solely relying on the fixed parametric router. It is worth noting that the accuracy of the IVFPQ variant closely matches that of the naive implementation, experiencing only a slight degradation. Importantly, it still consistently outperforms both Zero-shot and SFT (Router Only).

Comparison with 5-shot Compared to 5-shot, kNN-MoE exhibits more stable behavior across models. On Qwen3 30B, both 5-shot and kNN-

MoE can improve over Zero-shot; but for GPT-OSS and OLMoE, 5-shot often degrades performance relative to Zero-shot. A likely reason is that concatenating the five nearest reference examples substantially lengthens the input sequence, pushing these models into length regimes where their performance is less robust and amplifying the effect of retrieval noise.

Comparison with SFT While kNN-MoE outperforms SFT in certain scenarios, its relative performance varies. In settings with limited reference sets (GPQA, USMLE and MBPP), SFT occasionally underperforms compared to Zero-shot. Our analysis in Appendix C.2 indicates that this degradation cannot be attributed to a collapse in routing load balancing; therefore, we attribute it to potential overfitting induced by the small dataset size. In contrast, our kNN-MoE consistently outperforms Zero-shot in these data-scarce settings.

Statistical Significance Based on our paired bootstrap resampling tests, although improvements on certain individual benchmarks are not always statistically significant, kNN-MoE demonstrates a statistically significant improvement on the overall concatenated dataset (**Avg.**). This overall significance across diverse tasks solidifies our main conclusion. Furthermore, on benchmarks where kNN-MoE's gains are not statistically significant (e.g., GPQA), competitive methods such as SFT similarly fail to achieve significant improvements. This suggests that the lack of significance on these specific tasks is likely not a limitation of our routing refinement framework, but rather indicates that existing models struggle to extract further performance gains relying solely on the information provided in the given reference sets.

6 Discussion

6.1 Why Case-Based MoE Works

Our main hypothesis is that retrieval is most beneficial when the parametric router is uncertain, which is often correlated with increased perplexity. To test this, we bucketed test instances into three equal-sized groups based on the perplexity (PPL) of the original model (with zero-shot prompt) (High / Mid / Low PPL), and reported the accuracy gain of kNN-MoE over Zero-shot within each bucket. Table 2 reports results on OLMoE.

kNN-MoE usually yields the largest gains on High-PPL instances across all benchmarks (e.g., +6.15 on GPQA and +6.32 on USMLE), while the improvements on Mid-PPL instances are smaller but still consistently positive. In contrast, the gains on Low-PPL instances are close to zero and can become slightly negative on some datasets. The slight degradation on a subset of Low-PPL instances suggests that kNN-MoE can occasionally introduce noise when the router prediction is strong.

Benchmark	High PPL	Mid PPL	Low PPL
GPQA	+6.15	+2.99	-1.52
SuperGPQA	+0.17	+0.65	-0.06
MMLU	+2.07	+1.15	-0.13
USMLE	+6.32	+1.41	-1.13
MedMCQA	+6.04	+1.13	+0.56

Table 2: Accuracy gains (kNN-MoE minus Zero-shot) bucketed by perplexity (PPL) of the original model (Zero-shot), using OLMoE.

Overall, we hypothesize that the strong correlation between perplexity and kNN-MoE gains stems from the fact that higher perplexity indicates a greater deviation of the input from the router's data distribution. Consequently, the expert assignment predicted by the router becomes unreliable, whereas kNN-MoE improves the assignment.

Method	GPQA	MMLU	USMLE
kNN-MoE	29.80	47.81	35.04
kNN-MoE (Selective)	30.30	47.85	35.41

Table 3: Performance comparison on OLMoE between standard kNN-MoE and a **Selective** variant. The Selective variant falls back to the zero-shot baseline for inputs in the lowest perplexity tercile.

Inspired by this observation, we investigated a variant termed **kNN-MoE (Selective)**. This variant dynamically deactivates retrieval for inputs where Zero-shot is confident (specifically, the bottom 33% of test instances by perplexity) and falls back to the parametric router. As shown in Table 3, this strategy yields marginal improvements. However, it introduces significant computational overhead, as it requires a preliminary forward pass to estimate baseline perplexity. Given the negligible performance gain relative to the added cost, we retained the standard kNN-MoE as our primary approach.

6.2 Impact of Reference Set Size $|\mathcal{D}_{\text{ref}}|$

We next varied the size of the reference set used to build the per-layer memory and evaluate kNN-MoE on MedMCQA. Table 4 shows that larger memories generally improve performance for all three models.

$\lvert\mathcal{D}_{\text{ref}}\rvert$	OLMoE	GPT-OSS	Qwen3
0	35.57	56.20	62.51
0.25k	35.99	56.41	66.01
0.50k	36.87	56.92	66.68
1.00k	37.01	57.06	66.65

Table 4: Accuracy (%) of kNN-MoE on MedMCQA as a function of the reference set size $|\mathcal{D}_{\text{ref}}|$ used for memory construction. $|\mathcal{D}_{\text{ref}}| = 0$ corresponds to Zero-shot (no memory).

When $|\mathcal{D}_{\text{ref}}| = 0$, kNN-MoE degenerates to the zero-shot baseline. As $|\mathcal{D}_{\text{ref}}|$ increases, accuracy improves monotonically, with diminishing returns beyond roughly 0.5k–1k examples.

6.3 Impact of the Number of Neighbors (K)

K	OLMoE	GPT-OSS	Qwen3
1	37.01	57.06	66.65
3	36.89	56.55	65.76
5	36.02	56.74	65.98
10	36.34	56.11	65.19

Table 5: Accuracy (%) of kNN-MoE on MedMCQA using different numbers of neighbors K.

We investigated the sensitivity of kNN-MoE to the value of K. Table 5 presents the accuracy on MedMCQA for varying $K \in \{1, 3, 5, 10\}$. Contrary to typical non-parametric approaches (Khandelwal et al., 2020; Deguchi and Nagata, 2025) where aggregating multiple neighbors (e.g., $K \geq 5$) helps reduce variance and smooth out noise, we observe that performance consistently peaks at $K = 1$ across all three models. As K increases, accuracy generally degrades.

We interpret this essentially as a negative result regarding the benefit of neighbor aggregation in the context of MoE routing. This phenomenon suggests that the locally optimal expert assignments responsible for positive outcomes are rare within the memory $\mathcal{M}$. In other words, for a given query, there is often only a single (or very few) historical case that provides a truly beneficial routing signal. Consequently, forcing the retrieval of additional neighbors ($K > 1$) likely incorporates cases with lower relevance or conflicting expert assignments. Instead of reinforcing the correct decision, these additional neighbors dilute the strong signal provided by the nearest neighbor, thereby introducing noise that harms the final model performance. Based on this finding, we adopt $K = 1$ as the default setting for all other experiments.

6.4 Impact of Gradient Descent Steps (S)

S	OLMoE	GPT-OSS	Qwen3
1	37.01	57.06	66.65
3	37.05	57.12	66.58
10	36.98	57.01	66.71

Table 6: Accuracy (%) of kNN-MoE on MedMCQA with varying gradient descent steps S used for memory construction.

We studied the impact of the number of gradient descent steps S used during the *Memory Construction* phase (§4.1) to derive the locally optimal expert assignment. Intuitively, more optimization steps should lead to a lower negative log-likelihood on the reference tokens, potentially yielding higher-quality targets for the memory. We compared the performance of kNN-MoE constructed with $S \in \{1, 3, 10\}$ steps.

Table 6 summarizes the results on MedMCQA. We observe that increasing S beyond a single step yields negligible improvements in downstream accuracy. We hypothesize that the initial gradient direction $\nabla_r \mathcal{L}_t(r_t^{(0)})$ already captures the most critical information regarding which experts are underutilized or should be promoted. Since the memory stores discrete locally optimal assignment values v_t, slight refinements to the logits r_t in subsequent steps ($S > 1$) may not significantly alter the top-k ranking or the resulting routing distribution used as targets. Crucially, the computational cost of memory construction scales linearly with S. Setting $S = 1$ allows for rapid memory construction (as shown in §6.5) without compromising model performance. Therefore, we adopted $S = 1$ as the default setting for efficiency.

6.5 Memory Construction and Inference Costs

We investigated the memory construction and inference costs of kNN-MoE. Table 7 compares the preparation time on OLMoE with $|\mathcal{D}_{\text{ref}}| = 1.00$k, all conducted on a single NVIDIA A100 80GB. Constructing the kNN-MoE memory took 0.46 hours, which was substantially faster than both SFT and SFT (Router Only). This speedup is mainly driven by optimizing token-specific routing logits (rather than updating model parameters) and using only one gradient descent step per token. Additionally, the IVFPQ variant introduces an extra 0.21h for index quantization and compression. However,

this minor overhead does not alter our overall conclusion regarding the significant preparation time advantage over SFT.

$\lvert\mathcal{D}_{\text{ref}}\rvert$	SFT	SFT (Router Only)	kNN-MoE
1.00k	1.34h	1.33h	0.46h

Table 7: Memory construction time on OLMoE with $\lvert\mathcal{D}_{\text{ref}}\rvert = 1\text{k}$. Other reference sizes scale approximately linearly with $\lvert\mathcal{D}_{\text{ref}}\rvert$.

At inference time, kNN-MoE incurs additional latency due to per-token nearest neighbor retrieval. Table 8 reports the average latency per example as a function of $\lvert\mathcal{D}_{\text{ref}}\rvert$. As expected, latency increases with $\lvert\mathcal{D}_{\text{ref}}\rvert$ because each router query searches a larger index. With $\lvert\mathcal{D}_{\text{ref}}\rvert = 1.00\text{k}$ and $K = 1$, the overhead of naive kNN-MoE remains moderate. Fortunately, our IVFPQ variant successfully mitigates this inference time overhead. Even with $\lvert\mathcal{D}_{\text{ref}}\rvert = 1.00\text{k}$, it only increases the inference latency by approximately 3.67% relative to the zero-shot baseline.

$\lvert\mathcal{D}_{\text{ref}}\rvert$	kNN-MoE	kNN-MoE (IVFPQ)
0	5.99s	5.99s
0.25k	6.19s	6.14s
0.50k	6.41s	6.16s
1.00k	6.83s	6.21s

Table 8: Inference speed (seconds per example) using different reference set sizes $\lvert\mathcal{D}_{\text{ref}}\rvert$.

Table 9 illustrates the VRAM consumption during inference. While the VRAM usage of the naive kNN-MoE implementation increases significantly as $\lvert\mathcal{D}_{\text{ref}}\rvert$ grows, the IVFPQ variant exhibits a substantially slower growth rate. This efficiency is achieved through vector quantization, dimension reduction, and clustering. Even at $\lvert\mathcal{D}_{\text{ref}}\rvert = 1.00\text{k}$, the IVFPQ variant only increases VRAM usage by approximately 9.43% compared to the baseline.

$\lvert\mathcal{D}_{\text{ref}}\rvert$	kNN-MoE	kNN-MoE (IVFPQ)
0	16.02GB	16.02GB
0.25k	19.31GB	16.21GB
0.50k	22.47GB	16.94GB
1.00k	28.81GB	17.53GB

Table 9: VRAM consumption (GB) during inference across different reference set sizes $\lvert\mathcal{D}_{\text{ref}}\rvert$.

7 Conclusions and Future Work

We introduced kNN-MoE, a framework that augments MoE parametric routers with a non-parametric memory of locally optimal expert assignments derived from a labeled reference set. By interpolating router predictions with retrieved nearest-neighbor assignments using a similarity-based coefficient, kNN-MoE improves routing without parameter updates. Across three models and six benchmarks, it outperforms Zero-shot and router-only SFT, achieving SFT-competitive results with substantially less preparation time.

Future work will focus on two directions. First, we aim to accelerate retrieval via memory pruning to reduce computational overhead while preserving accuracy. Second, we plan to extend kNN-MoE to unlabeled reference sets by utilizing *LLM-as-a-judge* (Zheng et al., 2023) to generate pseudo-labels for constructing locally optimal expert assignments.

Limitations

The primary limitation of our current framework lies in its dependence on a labeled reference set ($\mathcal{D}_{\text{ref}}$) that shares a similar distribution with the target test data. Our experimental success is predicated on the assumption that such accessible, high-quality data is available. However, in practical, open-ended deployment scenarios, acquiring domain-specific labeled data can be challenging or expensive. Consequently, this dependency may restrict the applicability of kNN-MoE in strictly zero-resource settings or when facing severe distribution shifts where no relevant historical cases exist. As mentioned in Section 7, leveraging *LLM-as-a-judge* to generate pseudo-labels offers a potential pathway to mitigate this constraint by utilizing unlabeled data; however, we leave the validation of this semi-supervised approach to future work.

Additionally, while kNN-MoE avoids the high cost of parameter updates, the non-parametric retrieval mechanism introduces necessary overheads. Specifically, the inference latency increases due to neighbor search, and the memory storage requirements scale with the size of the reference set. While currently manageable for the scales investigated in this paper, these factors may necessitate further optimization, such as index pruning, for deployment in strictly resource-constrained environments.

Finally, our experiments involved only a limited number of datasets and models, which means our

conclusions are currently restricted to the relevant tasks and MoE models with up to 128 experts.

References

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. Program synthesis with large language models. *CoRR*, abs/2108.07732.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA. Curran Associates Inc.

B.V. Dasarathy. 1991. *Nearest Neighbor (NN) Norms: Nn Pattern Classification Techniques*. IEEE Computer Society Press tutorial. IEEE Computer Society Press.

Hiroyuki Deguchi and Masaaki Nagata. 2025. Case-based decision-theoretic decoding with quality memories. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 33679–33694, Suzhou, China. Association for Computational Linguistics.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res.*, 23(1).

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*.

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

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, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, and 7 others. 2024. Mixtral of experts. *Preprint*, arXiv:2401.04088.

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

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547.

Michael I. Jordan and Robert A. Jacobs. 1994. Hierarchical mixtures of experts and the em algorithm. *Neural Computation*, 6(2):181–214.

Herve Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2020. Generalization through memorization: Nearest neighbor language models. In *International Conference on Learning Representations*.

Philipp Koehn. 2004. Statistical significance tests for machine translation evaluation. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 388–395, Barcelona, Spain. Association for Computational Linguistics.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA. Curran Associates Inc.

Zhongyang Li, Ziyue Li, and Tianyi Zhou. 2025a. C3PO: Critical-layer, core-expert, collaborative pathway optimization for test-time expert re-mixing. In *Second Conference on Language Modeling*.

Zhongyang Li, Ziyue Li, and Tianyi Zhou. 2025b. Routing manifold alignment improves generalization of mixture-of-experts llms. *Preprint*, arXiv:2511.07419.

Ang Lv, Jin Ma, Yiyuan Ma, and Siyuan Qiao. 2026. Coupling experts and routers in mixture-of-experts via an auxiliary loss. In *The Fourteenth International Conference on Learning Representations*.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, Yuling Gu, Shane Arora, Akshita Bhagia, Dustin Schwenk, David Wadden, Alexander Wettig, Binyuan Hui, Tim Dettmers, Douwe Kiela, and 5 others. 2024. Olmoe: Open mixture-of-experts language models. *Preprint*, arXiv:2409.02060.

OpenAI. 2025. gpt-oss-120b & gpt-oss-20b model card. *Preprint*, arXiv:2508.10925.

Ankit Pal, Logesh Kumar Umapathi, and Malaikannan Sankarasubbu. 2022. Medmcqa: A large-scale multi-subject multi-choice dataset for medical domain question answering. In *Proceedings of the Conference on Health, Inference, and Learning*, volume 174 of *Proceedings of Machine Learning Research*, pages 248–260. PMLR.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2024. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*.

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*.

Guinan Su, Yanwu Yang, Li Shen, Lu Yin, Shiwei Liu, and Jonas Geiping. 2025. Rewiring experts on the fly:continuous rerouting for better online adaptation in mixture-of-expert models. *Preprint*, arXiv:2510.14853.

M-A-P Team, Xinrun Du, Yifan Yao, Kaijing Ma, Bingli Wang, Tianyu Zheng, Kang Zhu, Minghao Liu, Yiming Liang, Xiaolong Jin, Zhenlin Wei, Chujie Zheng, Kaixin Deng, Shian Jia, Sichao Jiang, Yiyan Liao, Rui Li, Qinrui Li, Sirun Li, and 77 others. 2025. Supergpqa: Scaling llm evaluation across 285 graduate disciplines. *Preprint*, arXiv:2502.14739.

Qwen Team. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Frank F. Xu, Uri Alon, and Graham Neubig. 2023. Why do nearest neighbor language models work? In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. 2025. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

A Prompt Templates

To ensure reproducibility, we provide the exact prompt templates used for our Zero-shot and 5-shot inference baselines. Please note that the templates below illustrate the case for multiple-choice questions with four options (A–D). For questions with a different number of options, the list of choices is adjusted accordingly while maintaining the same overall format.

A.1 Zero-shot Prompt

For all models and test sets in the Zero-shot setting, we formatted the input question and options using the template below. The placeholders `{question}` and `{choices}` are substituted with the specific instance data.

```
What is the correct answer to this
    question: {question}

Choices:
(A) {choices[0]}
(B) {choices[1]}
(C) {choices[2]}
(D) {choices[3]}

Answer with the format: The correct
    answer is (X).
```

Figure 2: Prompt template for the zero-shot setting.

A.2 5-shot Prompt

For the 5-shot setting, we constructed the prompt by concatenating five retrieved expert examples followed by the target question. Similar to the zero-shot case, the example below assumes four choices per question.

```
Here are some example expert
    multiple-choice questions. After
    the examples, answer the final
    question.

Question: {s1.question}
Choices:
(A) {s1.choices[0]}
(B) {s1.choices[1]}
(C) {s1.choices[2]}
(D) {s1.choices[3]}
The correct answer is ({s1.answer})

... [3 more examples] ...

Question: {s5.question}
Choices:
(A) {s5.choices[0]}
(B) {s5.choices[1]}
(C) {s5.choices[2]}
(D) {s5.choices[3]}
The correct answer is ({s5.answer})

Question: {target.question}
Choices:
(A) {target.choices[0]}
(B) {target.choices[1]}
(C) {target.choices[2]}
(D) {target.choices[3]}
```

Figure 3: Prompt template for the 5-shot setting.

B Training Implementation Details

Our supervised fine-tuning baselines—SFT and SFT (Router Only)—were implemented based on the TRL library[1]. In the SFT (Router Only) setting, we constrained the optimizer to update only the router parameters.

Table 15 summarizes the hyperparameters used for training.

C Additional Discussion

C.1 Comparison with C3PO

We compared kNN-MoE with C3PO (Li et al., 2025a), a state-of-the-art test-time routing adaptation method. Table 10 presents the accuracy on OLMoE across three benchmarks with varying reference set sizes. We observe distinct behaviors depending on the availability of reference data. On benchmarks with limited reference sets, (GPQA and USMLE), C3PO yields marginal gains or even performance degradation compared to the zero-shot baseline. In contrast, kNN-MoE achieves consistent improvements in these data-scarce regimes. However, on MMLU, which provides a larger reference set ($|\mathcal{D}_{\text{ref}}| \approx 1.81$k), C3PO outperforms kNN-MoE.

Method	GPQA	MMLU	USMLE
Zero-shot	27.27	46.77	32.81
C3PO	24.24	**48.04**	32.93
kNN-MoE	**29.80**	47.81	**35.04**

Table 10: Performance comparison with the state-of-the-art method C3PO using the OLMoE model. kNN-MoE demonstrates superior sample efficiency on smaller datasets (GPQA, USMLE).

We attribute these findings to the difference in sample efficiency. C3PO relies on a filtering mechanism that only utilizes reference examples correctly predicted by the baseline model to guide the router update. This significantly reduces the effective training signal when the reference set is small or the task is difficult. Conversely, kNN-MoE

[1] https://github.com/huggingface/trl

exploits *all* retrieved neighbors by computing the locally optimal expert assignment that maximizes the likelihood of the ground-truth token, regardless of the original model's correctness. This capability allows kNN-MoE to extract richer supervision signals from small datasets, making it more robust for specialized domains with limited data.

C.2 Load Balancing Analysis

We analyze the changes in the average load balancing loss across the entire test set for different methods to investigate the reasons behind the performance degradation of SFT. We report the results evaluated on GPQA using the OLMoE model.

As shown in Table 11, the load balancing losses for Zero-shot, SFT, SFT (Router Only), and kNN-MoE are highly similar. This indicates that the routing load balancing in SFT and its Router-Only variant did not collapse. This is an expected outcome, as the load balancing loss was incorporated as a constraint into the overall training objective. Therefore, we conjecture that the performance degradation observed in SFT and SFT (Router Only) is primarily due to overfitting caused by the limited size of the training dataset.

Zero-shot	SFT	SFT (Router Only)	kNN-MoE
8.27	8.29	8.28	8.29

Table 11: Average load balancing loss on the GPQA test set using the OLMoE model across different methods.

C.3 Impact of Choice of $s(\cdot, \cdot)$

$s(\cdot, \cdot)$	OLMoE	GPT-OSS	Qwen3
RBF	**37.01**	**57.06**	66.65
RBF + L2 Norm	36.52	56.19	**66.68**
Cosine	36.69	56.01	66.65

Table 12: Accuracy (%) of kNN-MoE on MedMCQA using different similarity functions $s(\cdot, \cdot)$. "RBF + L2 Norm" applies the RBF kernel to L2-normalized router inputs, isolating the contribution of vector magnitude.

We analyzed the sensitivity of kNN-MoE to the choice of the similarity function $s(x, x')$. Specifically, we compared our default RBF kernel, which relies on Euclidean distance, against cosine similarity, which measures the cosine of the angle between vectors. Table 12 presents the results on MedMCQA. We observe that the RBF kernel outperforms or matches cosine similarity.

We conjecture that this disparity stems from the way each metric handles vector magnitude. Cosine similarity normalizes the input vectors, effectively projecting them onto a unit hypersphere and discarding the information encoded in their norms: $s_{\text{cosine}}(x, k) = \frac{x \cdot k}{\|x\|\|k\|}$. In contrast, the RBF kernel depends on the distance $\|x - k\|^2$, which inherently preserves information about the scale (magnitude) of the router inputs. The superior performance of RBF on OLMoE and GPT-OSS suggests that for these architectures, the magnitude of the router input vector x likely exhibits significant variance and carries discriminative signals for expert assignment—information that is discarded when using cosine similarity. Conversely, the insensitivity of Qwen3 to the results with different metrics imply that its router inputs may be more uniform in magnitude or that its routing policy is primarily direction-dependent.

To directly test this hypothesis, we additionally evaluated an **RBF + L2 Norm** variant, in which the RBF kernel is applied to L2-normalized router inputs. This variant preserves the kernel form of the original RBF but removes magnitude information, isolating it as the only difference from standard RBF. As shown in Table 12, once magnitude is removed, the performance of RBF on OLMoE and GPT-OSS drops to essentially the same level as cosine similarity, while Qwen3 remains nearly unchanged. This result directly confirms that the gap between RBF and cosine on OLMoE and GPT-OSS is attributable to the magnitude of the router inputs, and that for Qwen3 the router inputs are effectively magnitude-uniform—so removing magnitude has no effect. We therefore retain the standard RBF kernel as the default in kNN-MoE, since it captures magnitude information that is useful on at least a subset of models while not hurting magnitude-uniform models such as Qwen3.

C.4 Further Analysis on the Number of Neighbors (K)

As shown in §6.3, kNN-MoE empirically prefers $K = 1$ across all three models, which appears to conflict with the intuition behind kNN aggregation. To better understand this phenomenon, we conducted an additional analysis quantifying how much of the aggregated similarity mass is concentrated on the single nearest neighbor under a $K = 5$ retrieval setting. Specifically, for each retrieved set $\{(k_j, v_j)\}_{j=1}^{5}$, we computed the share of total sim-

ilarity captured by the top-1 neighbor:

$$\text{Top-1 Share} = \frac{s(x, k_1)}{\sum_{j=1}^{5} s(x, k_j)},$$

where k_1 denotes the most similar neighbor. Table 13 reports the average Top-1 Share over the corresponding test set.

Dataset	OLMoE	GPT-OSS	Qwen3
GPQA	0.91	0.82	0.94
MMLU	0.78	0.76	0.83
MedMCQA	0.88	0.85	0.93

Table 13: Average share of total similarity mass held by the top-1 neighbor under $K = 5$ retrieval. Across all settings, the nearest neighbor already accounts for the overwhelming majority of the aggregated similarity.

Across all model–benchmark pairs, the top-1 neighbor accounts for 76%–94% of the total similarity mass. This indicates that even when $K > 1$, the aggregated memory-based proposal $r_{\text{mem}}(x)$ is already dominated by the nearest neighbor, while the marginal contribution of farther neighbors is small in magnitude. However, as Table 5 shows, increasing K consistently degrades accuracy. This suggests that the additional neighbors are small in magnitude but harmful in direction: they inject conflicting routing signals rather than reinforce the correct one, which is consistent with our earlier interpretation that beneficial expert assignments are sparse in the memory.

We further emphasize that the empirical preference for $K = 1$ is a property of the current models and reference sets rather than a structural property of the framework. We deliberately retain the general K-neighbor formulation in §4.2 so that the method remains applicable to future settings (e.g., larger or more diverse memories, alternative aggregation schemes, or domains in which beneficial routing signals are less sparse), even though the empirical sweet spot in our current setup happens to be $K = 1$.

D Impact of Random Seeds

We analyze the impact of random seeds on the performance of different methods. We restrict our analysis to SFT and its variants, as these are the only methods involving training where random LoRA parameter initialization can introduce performance variance. Other methods (e.g., Zero-shot and kNN-MoE) are deterministic in this regard. Due to computational constraints, we report the mean and standard deviation of the accuracy on a subset of benchmarks using the OLMoE model.

As shown in Table 14, SFT and its variants exhibit a certain degree of accuracy fluctuation across different runs. Nevertheless, this variance is relatively small and does not alter our primary conclusions.

Benchmark	SFT	SFT (Router Only)
GPQA	22.56±1.89	23.39±1.89
MMLU	46.21±1.02	45.67±0.95
USMLE	38.29±0.51	31.59±0.32

Table 14: Accuracy (mean ± standard deviation, %) across different random seeds for SFT and SFT (Router Only) using the OLMoE model.

Configuration	Value
Optimization	
Optimizer	AdamW
Batch Size	16
Weight Decay	0.01
Learning Rate Schedule	Linear Decay
LoRA Parameters	
Rank (r)	32
Alpha (α)	32
Dropout	0.0
Bias	None
Hardware	
GPU	$1 \times$ NVIDIA A100 (80GB)

Table 15: Hyperparameters and configuration for SFT and SFT (Router Only) experiments.

Benchmark	Test Source	Ref. Source	$\lvert\mathcal{D}_{\text{test}}\rvert$	$\lvert\mathcal{D}_{\text{ref}}\rvert$
GPQA	GPQA Diamond	GPQA Main (filtered)	0.20k	0.20k
MMLU	Test subset	Train subset	14.00k	1.81k
SuperGPQA	Held-out split	Random subset	23.50k	3.00k
USMLE	Test subset	Train subset	1.27k	0.20k
MedMCQA	Test subset	Train subset	6.15k	1.00k
MBPP	Held-out split	Random subset	0.60k	0.37k

Table 16: Statistics of evaluation benchmarks. $\mathcal{D}_{\text{test}}$ was used for reporting accuracy; $\mathcal{D}_{\text{ref}}$ served as the shared reference split across methods (for memory construction in kNN-MoE, retrieval in 5-shot, and SFT).