RAVE and QeRAVE: Router-Aware Virtual Experts for Efficient Mixture-of-Experts LLM Compression

Przemek Chojecki
ulam.ai

6th June 2026

Abstract

Sparsely activated Mixture-of-Experts (MoE) large language models reduce active computation but retain a large memory footprint because every expert must be stored and routed at inference time. Recent one-shot compression methods expose a tension between expert pruning and expert merging. Router-weighted Expert Activation Pruning (REAP) preserves the router's independent control over surviving experts but deletes low-saliency experts entirely. Router-weighted Expert Activation Merging (REAM) preserves more information by grouping and merging experts, but tied merged experts can reduce functional degrees of freedom and may not reduce active expert computation. This paper proposes RAVE: Router-Aware Virtual Experts, a method positioned between pruning and merging. RAVE classifies experts into four states: full keep, hard prune, centroid absorb, and *virtual residual expert*. A virtual expert shares a retained centroid expert but carries a small low-rank residual, preserving router-addressable specialization at a fraction of the full expert cost. We then define QeRAVE as the quantization-enhanced version of RAVE: quantization-induced perturbations are used as redundancy probes, so experts whose saliency and reconstruction behavior remain cold and stable under low-bit noise are safer to prune or absorb, while brittle experts are retained or converted into virtual residual experts. Beyond the method, we report an internal proof-judge pilot on seven SU-01-family variants, including the original `Simplified-Reasoning/SU-01`, over 40 held-out synthetic IMO-style problems under direct 8192-token decoding. `rave_64` is the aggregate winner with 154/280 points, an average score of 3.85/7, and 9/40 passes; `reap_64` ties the pass count with 146/280 points, while the original direct run scores 143/280 with 5/40 passes. We provide the method, optimization objectives, implementation sketch, latency and memory cost models, compute budgets, and a clear boundary between supported pilot results and hypotheses that still require public benchmark and serving validation.

1 Introduction

Sparse Mixture-of-Experts (MoE) layers are a central mechanism for increasing parameter count without increasing per-token computation in the same proportion. A router maps each token to a small subset of expert feed-forward networks, and the layer output is the gate-weighted sum of those selected experts [1–4]. This design makes it possible to build models with high total capacity but moderate active computation. However, deployment remains constrained by memory footprint, expert sharding, prefetching, and all-to-all communication: even when only a few experts are active for each token, all experts must be stored and made available.

Post-training MoE compression therefore matters. One direction is expert pruning. REAP proposes a router-weighted expert-activation criterion and argues that pruning can outperform merging on generative tasks because it preserves independent, input-dependent router control over the experts that remain [5]. Another direction is expert merging. REAM groups experts and merges their weights using router-aware similarity, pseudo-pruning, activation/weight alignment, and sequential statistic recomputation; it reports strong results on Qwen3-family and GLM-family MoE models and emphasizes calibration-mixture trade-offs between multiple-choice and generative benchmarks [6].

The key gap is that "delete" and "merge" are not the only possible actions. Some experts are truly redundant and can be removed. Some are well represented by a neighboring centroid expert and can be absorbed. Others are rarely used but encode tail behavior that is too valuable to delete and too specialized to average into a centroid. RAVE targets this middle regime by preserving such experts as *virtual residual experts*: they keep router identities but share most computation and storage with retained centroid experts.

The core method does not require quantization. This is important. The most defensible contribution is structural: retain a small number of full experts, attach low-rank residuals for ambiguous experts, and execute selected virtual experts by grouping them by shared centroid. Quantization is useful but optional. We therefore distinguish between two methods:

- **RAVE:** the base method. It uses clean calibration statistics, router-aware saliency, centroid assignment, and low-rank residual fitting. It is the simpler and lower-risk method to implement first.
- **QeRAVE:** the quantization-enhanced variant. It adds perturbation ensembles, low-bit expert passes, router-logit noise, and optional low-bit deployment. It is more ambitious and can be evaluated as an extension or ablation rather than as a prerequisite.

This distinction addresses a likely reviewer concern: if quantization and virtual experts are introduced at the same time, it is hard to know which component matters. A clean paper should first show that RAVE works without quantization, and then test whether QeRAVE improves calibration robustness, search cost, or deployment efficiency.

This paper makes five contributions.

1. It proposes RAVE, a quad-state expert-compression framework that interpolates between REAP-style pruning and REAM-style merging using low-rank virtual residual experts.
2. It defines QeRAVE as an optional quantization-enhanced version of RAVE, using low-bit perturbations as a redundancy and router-brittleness probe rather than only as a deployment format.
3. It derives storage, active-compute, and latency models that clarify when virtual experts can be faster than the original model: parameter reduction alone is insufficient; one must reduce the number of unique full centroids executed per token or lower memory bandwidth.
4. It reports a strict internal olympiad-style proof-judge pilot comparing `rave_64`, `rave_77`, `reap_64`, `reap_77`, `ream_64`, `ream_77`, and the original SU-01 direct run.
5. It gives a compute plan and fair-comparison protocol for public validation, including clean RAVE, post-hoc quantized RAVE, quantized baselines, QeRAVE, and latency-matched serving measurements.

All empirical statements in this manuscript are restricted to the internal pilot in Section 7. The broader method, latency, and deployment claims are hypotheses or planning estimates until validated on public benchmarks and real serving kernels.

2 Related Work

Sparse MoE language models. Adaptive mixtures of local experts were introduced decades ago [1]. Sparsely gated MoE layers later made conditional computation practical at scale [2], and Switch/GShard-style systems helped establish MoE as a route to very high parameter counts with sparse per-token activation [3, 4]. Decoder-only MoE LLMs such as Mixtral and Qwen3-family models demonstrate that the design is practically useful for open-weight language models [7, 8]. The deployment challenge is that sparse activation saves compute but not necessarily memory or serving complexity.

Expert pruning. Expert pruning removes experts or substructures judged unimportant. Prior methods use routing frequency, saliency, differentiable masks, evolutionary search, and router/activation-aware criteria. Weight-level dense-model pruning methods such as SparseGPT and Wanda motivate activation-aware post-training compression, but MoE expert removal adds the additional problem of preserving router-controlled specialization [9, 10]. EEP explicitly considers inference-cost reductions and can remove active experts in addition to total experts [11]. DiEP formulates non-uniform layer-wise pruning as a differentiable search problem [12]. MoE-I2 combines inter-expert pruning with intra-expert low-rank decomposition, while MoE-Pruner uses router-weighted activation hints for weight-level pruning and reports that expert-wise distillation can recover much of the lost quality after pruning [13, 14]. REAP focuses on one-shot expert-level pruning with a router-weighted expert-activation score, and argues that merging can introduce functional subspace collapse because several independently gated functions become tied [5]. RAVE reuses the insight that router-weighted contribution is a strong saliency proxy, but it adds a middle action between keeping and deleting.

Expert merging. Expert merging reduces stored experts by grouping similar experts and replacing each group with a merged expert. MC-SMoE and related methods use routing statistics, expert similarity, and

neuron alignment [15]. HC-SMoE applies output-based hierarchical clustering without retraining [16]. Sub-MoE uses shared-subspace merging to reduce conflicts among experts [17]. REAM improves merging with gate-logit/output similarity, pseudo-pruning, activation/weight alignment, and sequential recomputation, and evaluates both multiple-choice and generative benchmarks across calibration mixtures [6]. RAVE treats absorption into a centroid as one option but avoids forcing every compressed expert into a fully merged weight average.

Low-rank adaptation, quantization, and distillation. LoRA shows that low-rank adapters can represent useful parameter updates with small storage overhead [18]. QLoRA combines quantized base models and low-rank updates for efficient adaptation [19]. Post-training quantization methods such as GPTQ and SmoothQuant reduce memory bandwidth and can be complementary to expert compression [20, 21]. QeRL combines low-bit quantization with LoRA-style updates for reinforcement learning and reports that quantization noise can increase policy entropy and aid exploration [22]. QeRAVE imports the idea of useful noise into compression search. Optional recovery training follows the general distillation framework [23] and MoE-specific recovery strategies such as MoE-Pruner's expert-wise distillation [14].

3 Background and Problem Setup

Consider a Transformer with L MoE layers [24]. At layer ℓ, the router produces logits

$$r_\ell(x) = W_\ell^{(r)} h_\ell(x), \tag{1}$$

where $h_\ell(x) \in \mathbb{R}^d$ is the token representation. The selected experts are

$$\mathcal{T}_\ell(x) = \mathrm{TopK}(r_\ell(x), k), \tag{2}$$

and the normalized gate for expert e is $g_{\ell,e}(x)$. The MoE output is

$$y_\ell(x) = \sum_{e \in \mathcal{T}_\ell(x)} g_{\ell,e}(x) E_{\ell,e}(h_\ell(x)), \tag{3}$$

where $E_{\ell,e}$ is an expert MLP.

The compression problem is to replace the original model M with $\tilde{M}$ while controlling quality loss, memory, and latency:

$$\begin{aligned} \min_{\tilde{M}} \quad & \mathcal{L}_{\mathrm{task}}(\tilde{M}; \mathcal{D}_{\mathrm{val}}) - \mathcal{L}_{\mathrm{task}}(M; \mathcal{D}_{\mathrm{val}}) \\ \text{s.t.} \quad & \mathrm{Mem}(\tilde{M}) \le B_{\mathrm{mem}}, \qquad \mathrm{Lat}(\tilde{M}) \le B_{\mathrm{lat}}. \end{aligned} \tag{4}$$

Directly optimizing Equation 4 is expensive, so one-shot methods use proxy scores.

A REAP-style expert saliency can be written as

$$s_{\ell,e}^{\mathrm{act}} = \mathbb{E}_{x \sim \mathcal{D}_{\mathrm{cal}}} \left[\mathbf{1}\{e \in \mathcal{T}_\ell(x)\}\, g_{\ell,e}(x)\, \|E_{\ell,e}(h_\ell(x))\|_2 \right]. \tag{5}$$

Frequency only counts selection. Equation 5 additionally measures how much the expert contributes to the gated layer output.

Expert merging also needs similarity. A generic gated-output similarity is

$$\mathrm{sim}_\ell(e, e') = \mathbb{E}_{x \sim \mathcal{D}_{\mathrm{cal}}} \left[\cos_\epsilon \left(g_{\ell,e}(x) E_{\ell,e}(h_\ell(x)), g_{\ell,e'}(x) E_{\ell,e'}(h_\ell(x)) \right) \right], \tag{6}$$

where $\cos_\epsilon$ denotes cosine similarity with a small denominator floor. Practical variants combine gate-logit similarity, output similarity, activation alignment, and weight-alignment costs [6]. In large MoEs, Equation 6 should be implemented with care: only selected expert outputs are cheap to cache, so the expectation should either be conditional on a union or co-activation set, or computed from compact projections after a candidate-neighbor filtering stage. A literal all-pairs, all-token expert-output cache is not intended for 512-expert models.

4 RAVE: Router-Aware Virtual Experts

4.1 The quad-state expert action space

RAVE assigns each expert to exactly one of four states:

1. **Full keep.** The expert remains a full expert and retains its router identity.
2. **Hard prune.** The expert is removed and its router row is deleted or masked.
3. **Centroid absorb.** The expert's router identity may be removed or remapped to a retained centroid, with no residual parameters.
4. **Virtual residual expert.** The expert keeps a virtual router identity but shares a retained centroid expert and adds a small residual.

This action space is the core difference from both REAP and REAM. REAP primarily decides whether to keep or delete an expert. REAM primarily decides how to group and merge experts. RAVE asks whether an expert is redundant, centroid-equivalent, or worth preserving as a cheap residual around a centroid. In particular, rank-zero RAVE absorption is not the same operation as REAM merging: the absorbed expert is either removed from routing or remapped to an existing centroid, while the centroid weights themselves are not averaged with the absorbed expert unless an optional centroid-correction adapter is trained later.

4.2 Virtual expert parameterization

Let $c_\ell(e)$ denote the centroid assigned to expert e in layer ℓ. RAVE approximates a virtual expert as

$$\tilde{E}_{\ell,e}(h) = E_{\ell,c_\ell(e)}(h) + \Delta_{\ell,e}(h), \tag{7}$$

where $E_{\ell,c_\ell(e)}$ is a retained full expert and $\Delta_{\ell,e}$ is a low-rank residual. If $\Delta_{\ell,e} = 0$, this is pure absorption. If the residual rank is high enough, the virtual expert approaches a separate full expert. Useful compression is expected in the low-rank middle.

For a SwiGLU-style expert with model width d_{model} and expert intermediate width d_{ff}, the matrices have shapes $W_u, W_g \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$ and $W_o \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$. A full expert has approximately

$$P_E \approx 3 d_{\text{model}} d_{\text{ff}} \tag{8}$$

parameters. A LoRA-style residual of rank r attached to the three expert matrices has approximately

$$P_\Delta(r) \approx 3r(d_{\text{model}} + d_{\text{ff}}) \tag{9}$$

parameters, giving a residual-to-full-expert ratio

$$\epsilon_r = \frac{P_\Delta(r)}{P_E} \approx r\left(\frac{1}{d_{\text{model}}} + \frac{1}{d_{\text{ff}}}\right). \tag{10}$$

This notation avoids overloading the hidden state $h_\ell(x)$ with the expert's internal width. For common LLM dimensions, ranks $r \in \{4, 8, 16\}$ are a small percentage of a full expert. This is what makes virtual experts attractive: many router-addressable tails can be retained for the storage cost of a few full experts.

The equations below treat $\Delta_{\ell,e}$ as an output residual that is added after the centroid expert output is computed. A LoRA-inside-the-expert implementation is also possible, but then $\Delta_{\ell,e}(h)$ denotes the induced output difference of the adapted nonlinear expert path, not simply a linear matrix delta. The same residual parameterization must be used during residual fitting and deployment.

4.3 Shared-base execution

The speed mechanism is not merely that fewer parameters are stored. The speed mechanism is shared-base execution. If token x selects virtual expert IDs $\mathcal{T}_\ell(x)$, then

$$\begin{aligned} \tilde{y}_\ell(x) &= \sum_{e \in \mathcal{T}_\ell(x)} \tilde{g}_{\ell,e}(x) \left[E_{\ell,c_\ell(e)}(h) + \Delta_{\ell,e}(h) \right] \\ &= \sum_{c \in \mathcal{U}_\ell(x)} \left(\sum_{e \in \mathcal{T}_\ell(x) : c_\ell(e) = c} \tilde{g}_{\ell,e}(x) \right) E_{\ell,c}(h) + \sum_{e \in \mathcal{T}_\ell(x)} \tilde{g}_{\ell,e}(x) \Delta_{\ell,e}(h), \end{aligned} \tag{11}$$

where

$$\mathcal{U}_\ell(x) = \{c_\ell(e) : e \in \mathcal{T}_\ell(x)\} \tag{12}$$

is the set of unique full centroids used by selected virtual experts. If $|\mathcal{U}_\ell(x)| < k$, then the layer executes fewer full experts than the original top-k layer and adds only small residual operations.

This is also the critical implementation requirement. A naive runtime that computes $E_{\ell,c(e)}$ separately for each selected virtual expert can lose the intended speedup. The grouped kernel must aggregate gates by centroid, compute each centroid once, compute residuals, and accumulate. A second requirement is routing hygiene: if top-k selection repeatedly chooses several virtual IDs that share the same centroid but have negligible residuals, the layer effectively wastes router slots. RAVE should therefore track duplicate-centroid selections during calibration and penalize virtual-ID fragmentation unless the corresponding residuals explain distinct errors.

4.4 Routing semantics and invariants

The compressed model has two namespaces. The *virtual expert namespace* is what the router scores. The *centroid namespace* is what the full expert kernels execute. For every non-pruned virtual expert e, RAVE stores a mapping $c_\ell(e)$. Full-kept experts use the convention $c_\ell(e) = e$ and $\Delta_{\ell,e} = 0$ unless an optional self-correction adapter is trained. A hard-pruned expert has no valid mapping. An absorbed expert either loses its router row or has its row remapped to the centroid row, depending on whether the implementation wants to preserve the original virtual ID for diagnostics.

The default routing rule is virtual-ID top-k:

$$\tilde{\mathcal{T}}_\ell(x) = \mathrm{TopK}(\tilde{r}_\ell(x), k), \qquad G_{\ell,c}(x) = \sum_{e \in \tilde{\mathcal{T}}_\ell(x) : c_\ell(e) = c} \tilde{g}_{\ell,e}(x), \tag{13}$$

where $G_{\ell,c}$ is the aggregated gate used for the shared centroid execution in Equation 11. This rule best preserves the teacher router's expert identities, but it can also select several virtual IDs that share one centroid. Such collisions are the mechanism that lowers $|\mathcal{U}_\ell(x)|$, but excessive collisions can reduce functional diversity. Therefore the active-centroid target must be treated as a quality–latency trade-off, not a free speedup.

A stricter latency variant is centroid-first routing: score virtual IDs, aggregate scores by centroid, select a smaller number $k_c < k$ of centroids, and then select residual IDs only within those centroids. This variant is more kernel-friendly and gives stronger latency control, but it changes the router semantics more aggressively. Both variants should be ablated.

4.5 Clean RAVE scoring

The base RAVE method uses clean calibration statistics. For each expert, compute:

$$s_{\ell,e} = \mathbb{E}_x \left[\mathbf{1}\{e \in \mathcal{T}_\ell(x)\} g_{\ell,e}(x) \| E_{\ell,e}(h_\ell(x)) \|_2 \right], \tag{14}$$

$$f_{\ell,e} = \mathbb{E}_x \left[\mathbf{1}\{e \in \mathcal{T}_\ell(x)\} \right], \tag{15}$$

$$a_{\ell,e} = \mathbb{E}_x \left[\mathbf{1}\{e \in \mathcal{T}_\ell(x)\} \| E_{\ell,e}(h_\ell(x)) \|_2^2 \right]. \tag{16}$$

Then choose a diverse set of centroid experts $\mathcal{C}_\ell$ using high saliency and coverage over routing regions. A simple objective for centroid choice is

$$\max_{\mathcal{C}_\ell : |\mathcal{C}_\ell| = m_\ell} \sum_{c \in \mathcal{C}_\ell} s_{\ell,c} + \lambda_{\mathrm{div}} \sum_{c \neq c' \in \mathcal{C}_\ell} d_{\mathrm{route}}(c, c'), \tag{17}$$

where d_{route} measures distance between gate-logit profiles, co-activation profiles, or random projections of gated outputs. This avoids choosing only frequent experts from one domain.

For each non-centroid expert e, evaluate a candidate set of centroids $\mathcal{N}_\ell(e)$ by similarity and residual reconstruction. Because the residual in Equation 7 is multiplied by the router gate at inference time, the residual should fit the ungated expert difference, with gate values used only as weights. Let

$$w_{\ell,e}(x) = \mathbf{1}\{e \in \mathcal{T}_\ell(x)\} g_{\ell,e}(x)^2. \tag{18}$$

Then define

$$R_{\ell,e \to c}(r) = \min_{\Delta \in \mathcal{R}_r} \mathbb{E}_{x \sim \mathcal{D}_{\mathrm{cal}}} \left[w_{\ell,e}(x) \, \| E_{\ell,e}(h_\ell(x)) - E_{\ell,c}(h_\ell(x)) - \Delta(h_\ell(x)) \|_2^2 \right], \tag{19}$$

where $\mathcal{R}_r$ is the class of rank-r residuals. The gate multiplies the reconstruction error; the residual function itself remains ungated, matching Equation 11. Candidate ranks may be $r \in \{0, 4, 8, 16\}$. Rank zero corresponds to absorption. An equivalent implementation may instead define a post-gate residual, but then the inference equation must not multiply that residual by the gate a second time.

The clean RAVE action cost can be written as

$$A_{\ell,e}(a) = \begin{cases} 0, & a = \text{full keep}, \\ \lambda_p s_{\ell,e}, & a = \text{hard prune}, \\ \lambda_m R_{\ell,e\to c}(0) + \lambda_s(1 - \mathrm{sim}_\ell(e,c)), & a = \text{absorb into } c, \\ \lambda_r R_{\ell,e\to c}(r) + \lambda_b \epsilon_r, & a = \text{virtual residual rank } r \text{ on } c. \end{cases} \tag{20}$$

This is a proxy, not a substitute for evaluation. Its purpose is to cheaply rank actions before optional adaptation.

4.6 Layer-wise assignment objective

Let $z^F_{\ell,e}$ indicate full keep, $z^P_{\ell,e}$ hard prune, $z^A_{\ell,e,c}$ absorption into centroid c, and $z^V_{\ell,e,c,r}$ virtual residual rank r on centroid c. Each expert receives one action:

$$z^F_{\ell,e} + z^P_{\ell,e} + \sum_c z^A_{\ell,e,c} + \sum_{c,r} z^V_{\ell,e,c,r} = 1. \tag{21}$$

Centroid feasibility requires

$$z^A_{\ell,e,c} \le z^F_{\ell,c}, \qquad z^V_{\ell,e,c,r} \le z^F_{\ell,c}, \tag{22}$$

so an expert can be absorbed into, or made virtual around, only a retained full centroid. If an absorbed expert keeps a router row with zero residual, it should be reported as a rank-zero virtual expert; if its row is deleted, its gate mass must be either dropped and renormalized or remapped to the centroid. These choices affect top-k competition and should not be silently conflated.

The assignment objective is

$$\begin{aligned} \min_z \quad & \sum_{\ell,e} \left[z^P_{\ell,e} A_{\ell,e}(P) + \sum_c z^A_{\ell,e,c} A_{\ell,e}(A_c) + \sum_{c,r} z^V_{\ell,e,c,r} A_{\ell,e}(V_{c,r}) \right] \\ & + \lambda_{\mathrm{mem}} \mathrm{Mem}(z) + \lambda_{\mathrm{act}} \mathbb{E}_{x,\ell}\left[|\mathcal{U}_\ell(x)|\right] + \lambda_{\mathrm{bal}} \Omega_{\mathrm{load}}(z). \end{aligned} \tag{23}$$

The term $\mathbb{E}[|\mathcal{U}_\ell(x)|]$ is essential for speed. Without it, RAVE may reduce storage but not active full-expert compute. Ω_{load} discourages pathological assignments that overload a few centroids or create distributed-serving bottlenecks. In practice, Equation 23 should be solved either with explicit memory and active-centroid constraints or by tuning the Lagrange multipliers to a target budget. Without such a budget, the zero-distortion solution is to keep every expert full.

4.7 One-shot versus sequential RAVE

The cheapest RAVE implementation is one-shot: collect all statistics from the original model and compress all layers independently. This is the closest analogue of a simple REAP pass. However, compressing early MoE layers changes the hidden states seen by later MoE layers, so original-model statistics may become stale. A sequential RAVE variant can mirror the motivation of sequential REAM: compress layer ℓ, run the calibration stream through the partially compressed model, and recompute the statistics used for layer $\ell+1$. This increases search time but reduces distribution shift in the assignment objective. Experiments should report whether sequential recomputation helps RAVE enough to justify its cost.

4.8 Algorithmic summary and accounting

A minimal RAVE implementation should use the following sequence.

1. Run one calibration pass and collect router logits, top-k expert IDs, gates, expert-output norms, and compact random projections of selected expert outputs.

2. Select full centroids per layer using saliency plus diversity over routing and activation regions. The term "full-expert reduction" should mean the reduction in retained full expert MLPs; virtual IDs are accounted for separately through residual storage and active-centroid count.
3. For each non-centroid expert, preselect a small set of candidate centroids, estimate rank-0/4/8/16 reconstruction quality, and assign one of the four actions: full keep, hard prune, absorb, or virtual residual.
4. Construct the compressed router map. Hard-pruned experts are masked before top-k selection or removed with gate renormalization. Absorbed experts either alias a centroid with rank zero or are deleted, depending on whether their router identity helps quality. Virtual residual experts keep a virtual ID and share centroid execution.
5. Optionally adapt router biases and residual adapters under teacher distillation, then evaluate benchmark quality and measured latency. If no grouped virtual-expert kernel is used, report RAVE as memory compression rather than speed compression.

5 QeRAVE: Quantization-Enhanced RAVE

QeRAVE adds quantization-aware perturbation to RAVE. It should be treated as an extension rather than the base method. No reinforcement learning is performed; the connection to QeRL is the use of quantization noise as an informative perturbation rather than merely as an efficiency trick. The hypothesis is that experts that remain unimportant under multiple perturbations are safer to prune or absorb, while experts whose saliency or top-k status changes under small perturbations are brittle and should be protected.

5.1 Perturbation ensemble

Let Π be a set of calibration perturbations. A perturbation $\pi \in \Pi$ may include:
1. expert weight quantization, such as FP8, INT8, INT4, or NVFP4-style formats;
2. activation quantization or clipping;
3. router-logit temperature changes;
4. small router-logit noise;
5. calibration bootstrap resampling;
6. pseudo-ablation or dropout of low-gate candidate experts.

For each perturbation, compute

$$s_{\ell,e}^{(\pi)} = \mathbb{E}_{x\sim\mathcal{D}_{\text{cal}}^{(\pi)}}\left[\mathbf{1}\{e \in \mathcal{T}_\ell^{(\pi)}(x)\} g_{\ell,e}^{(\pi)}(x) \left\| E_{\ell,e}^{(\pi)}(h_\ell^{(\pi)}(x)) \right\|_2\right]. \tag{24}$$

Then define

$$\mu_{\ell,e} = \mathbb{E}_{\pi\sim\Pi}[s_{\ell,e}^{(\pi)}], \qquad \sigma_{\ell,e} = \sqrt{\mathrm{Var}_{\pi\sim\Pi}[s_{\ell,e}^{(\pi)}]}. \tag{25}$$

A noise-stable importance score is

$$I_{\ell,e} = \mu_{\ell,e} + \alpha\sigma_{\ell,e} + \beta D_{\ell,e}^{\text{drop}} + \gamma B_{\ell,e}^{\text{route}}, \tag{26}$$

where $D_{\ell,e}^{\text{drop}}$ estimates hidden-state or loss distortion when expert e is removed, and $B_{\ell,e}^{\text{route}}$ measures top-k brittleness under perturbation. QeRAVE replaces the clean saliency $s_{\ell,e}$ in Equation 20 with $I_{\ell,e}$ and may fit residuals against quantized centroids rather than BF16 centroids.

5.2 When quantization should help

Quantization can help in three different ways:
1. **Search robustness.** Multiple low-bit passes approximate a local neighborhood around the model. Experts that are cold throughout the neighborhood are safer compression targets than experts that are cold only in one clean pass.
2. **Noise regularization.** Residual/router adaptation under quantized centroids can learn corrections that are robust to the final deployment format.
3. **Deployment bandwidth.** Quantized centroids and residuals reduce memory traffic, which may matter more than arithmetic FLOPs in decode-heavy serving.

Quantization can also hurt. Router top-k ranking is discrete; small numerical changes can alter selected experts. Therefore the default should keep routers in BF16/FP16 until experiments show that lower precision is safe.

5.3 RAVE versus QeRAVE

Table 1 summarizes the intended distinction. The first implementation should be RAVE unless the goal is specifically to study quantization noise.

Aspect	RAVE	QeRAVE
Core idea	Shared centroids plus low-rank virtual residual experts.	Same structural method.
Quantization required	No.	Yes for search perturbations and/or final deployment.
Calibration passes	One clean pass plus optional bootstrap.	B noisy/quantized passes, typically $B \in [4, 8]$.
Search cost	Lower and easier to reproduce.	Higher, unless low-bit passes are much cheaper.
Scientific role	Main method and cleanest ablation.	Extension testing noise robustness and low-bit deployment.
Deployment risk	Lower; standard precision is possible.	Higher; router and residual quantization must be validated.
Best use case	Establish that virtual experts beat prune/merge at equal memory.	Improve robustness, bandwidth, or adaptation once RAVE works.

Table 1: RAVE should be the core method; QeRAVE is an optional quantization-enhanced variant.

A fair QeRAVE evaluation must include post-hoc quantized baselines: quantized original, quantized REAP, quantized REAM, and RAVE followed by the same deployment quantization. Otherwise, an apparent QeRAVE gain could simply be a generic low-bit bandwidth gain rather than a benefit from quantization-noise-guided compression search.

6 Comparison with REAP and REAM

6.1 Conceptual comparison

REAP, REAM, RAVE, and QeRAVE occupy different points in the compression design space.

Method	Main action	Router independence	Tail knowledge	Speed mechanism
REAP	Delete low-saliency experts.	High for surviving experts.	Lost for pruned experts.	Memory reduction; speed only if active-k or serving overhead falls.
REAM	Group and merge experts.	Reduced within merged groups.	Partly preserved in merged weights.	Memory reduction; active full-expert count often unchanged.
RAVE	Keep centroids plus virtual residual experts.	Preserved for selected virtual IDs.	Preserved when residual rank is enough.	Fewer unique full centroids plus cheap residuals.
QeRAVE	RAVE with quantized/noisy search.	Same as RAVE.	Same, with low-bit robustness pressure.	Same as RAVE plus lower memory bandwidth if quantized.

Table 2: High-level comparison. The central advantage of RAVE is that it avoids both hard deletion and full functional collapse.

Rank-0 RAVE should not be described as REAM. In rank-0 RAVE, compressed experts are absorbed into unmodified centroids or remapped at the router level; no averaged or aligned expert weights are created. REAM instead explicitly constructs merged expert weights. This distinction matters because rank-0 RAVE tests router remapping and centroid sharing, while REAM tests whether weight merging preserves information better than deletion.

6.2 Expected behavior for full MoE validation

The working hypotheses are:

1. At equal storage, RAVE should lose less tail behavior than REAP because some low-saliency experts become residual virtual experts instead of being deleted.
2. At equal storage, RAVE should preserve more router-controllable specialization than REAM because virtual experts can keep separate router identities even when they share a centroid.
3. At equal latency, RAVE must explicitly reduce $\mathbb{E}[|\mathcal{U}_\ell(x)|]$. Without grouped execution and an active-centroid objective, it may be only a memory-quality method.
4. QeRAVE should help most when calibration decisions are unstable across domains or when the final deployment will be low-bit. If clean RAVE is already stable, QeRAVE may not justify its extra search cost.

These remain hypotheses for full MoE compression and serving validation; Section 7 supports only the narrower SU-01-family direct-decoding pilot.

7 Internal Olympiad-Style Evaluation

7.1 Benchmark and judging protocol

To make the method paper falsifiable before a full public serving study, we ran a strict proof-judge pilot on seven SU-01-family outputs. The reference baseline is the released `Simplified-Reasoning/SU-01` model, a 30B-A3B olympiad reasoning model whose public report emphasizes long-horizon proof solving and test-time scaling [25, 26]. Our pilot intentionally evaluates the harder direct-decoding setting: no multi-round generate–verify–revise loop, no external tools, and an 8192-token cap. It therefore should not be interpreted as a reproduction or contradiction of the full SU-01 test-time-scaling pipeline.

The benchmark, internally named `ai-imo-2`, contains 40 internally generated IMO-style synthetic mathematics problems. The problems were not used in our compression, prompting, or model-selection process. Because no external laboratory can inspect every pretraining corpus used by every upstream model, the publishable claim is that this is an internal held-out benchmark rather than a public benchmark-contamination claim. Each model output was graded as a contest proof against a supplied reference solution using a 0–7 olympiad-proof rubric: 0 means no meaningful progress, 6 means a complete correct proof, and 7 means a complete, clear, robust proof. Scores of 6 or 7 count as passes.

All seven variants produced non-empty outputs on all 40 problems. However, 274 of 280 generations reached the 8192-token cap. The judge therefore scored only the visible mathematical work and did not give credit for proof steps that might have appeared beyond truncation. This caveat is central: the pilot measures proof closure under a strict direct-decoding budget, not unconstrained long-context reasoning ability.

The model keys in Table 3 denote compression family and target expert budget for the same 128-expert SU-01 base model. `original` is the uncompressed `Simplified-Reasoning/SU-01` direct-decoding baseline. `reap_64` and `reap_77` are REAP pruning runs that retain 64 or 77 of 128 experts, corresponding to roughly 50% and 60% expert-storage settings. `ream_64` and `ream_77` use the same 64/77 target budgets but replace deletion with router-aware expert merging. `rave_64` and `rave_77` are the corresponding RAVE runs: experts are assigned to 64 or 77 retained centroids, while selected removed experts are represented as virtual residual experts around those centroids rather than being simply deleted or fully merged. In these pilot runs, the RAVE centering and residual-assignment statistics were computed on our internal mathematics calibration mixture, which we call OlympiadNet; this calibration data is separate from the 40-problem proof-judge evaluation set and was used to choose/center compressed expert structure, not as an evaluation answer source.

7.2 Leaderboard

Table 3 gives the headline result. `rave_64` is the overall winner by total judged score, with 154/280 points and an average score of 3.85/7. It also ties the best pass count, 9/40, with `reap_64`. The margin is not a pointwise sweep: compared with `reap_64`, `rave_64` gains 8 total points, wins 11 problems, ties 23, and loses 6. Compared with the original SU-01 direct run, `rave_64` gains 11 total points, wins 13 problems, ties 20, loses 7, and adds four more passes. The strongest conclusion is therefore precise: `rave_64` leads the aggregate partial-credit leaderboard in this capped direct-decoding pilot, while `reap_64` remains equally strong by raw pass count.

Rank	Variant	Score	Avg. /7	Passes	Judge synopsis
1	`rave_64`	154/280	3.85	9/40	Best overall; strongest mix of complete proofs and near-solutions, especially invariant-set, coordinate, and conic-substitution tasks.
2	`reap_64`	146/280	3.65	9/40	Tied the best pass count and produced several clean short solutions, but stalled more often after finding the right idea.
3	`rave_77`	146/280	3.65	5/40	Tied REAP-64 in total points but had fewer complete passes; often close on analytic/vector problems.
4	`original`	143/280	3.58	5/40	Strong direct-decoding baseline, but below RAVE-64 in total score and pass count under this 8K cap.
5	`reap_77`	139/280	3.48	7/40	Volatile: some excellent passes, but more low partial scores than the best 64-expert variants.
6	`ream_77`	105/280	2.62	2/40	Occasionally solved short counting/gap tasks, but far behind REAP/RAVE on proof closure.
7	`ream_64`	76/280	1.90	0/40	Weakest run; no passes and many exploratory or incomplete outputs.

Table 3: Internal `ai-imo-2` leaderboard under direct 8192-token decoding. A pass is a judge score of 6 or 7 on a 0–7 olympiad-proof rubric.

7.3 Judge opinion and failure modes

The judge report's qualitative ruling was that `rave_64` had the strongest mix of complete elementary proofs and near-solutions. Its best areas were invariant-set arguments, coordinate solutions, and conic-substitution problems. It passed nine problems across number theory, double counting, finite invariant sets, interval selection, coordinate geometry, and finite-exception arguments; the full problem list and score matrix are reserved for supplementary material rather than the main paper. It also had five 5-point near misses, suggesting that its advantage came from both full proof completions and higher-quality partial progress.

The main failure mode across all variants was not a total absence of ideas. Many outputs found the intended motif but failed to close the decisive lemma, equality case, or final bounding step before truncation. The hardest clusters were synthetic geometry, convex/side-width arguments, clique balancing, and polynomial-method grid-cover lower bounds. The easiest clusters were short number-theory, double-counting, finite-invariant-set, weighted interval-selection, orthocentre-vector, conic-substitution, and local-gap problems. This pattern supports using the internal benchmark as a stress test for proof closure, but it does not replace public math, code, instruction-following, memory, or latency evaluation.

8 Residual Fitting and Optional Adaptation

8.1 One-shot residual fitting

The cheapest RAVE variant fits residuals layer by layer from cached calibration pairs. For a virtual expert e attached to centroid c, collect inputs h_i and ungated expert differences

$$d_i = E_{\ell,e}(h_i) - E_{\ell,c}(h_i), \tag{27}$$

then fit $\Delta_{\ell,e}(h_i) \approx d_i$ with weights $w_i = g_{\ell,e}(x_i)^2$ or, equivalently, minimize $\|g_{\ell,e}(x_i)(d_i - \Delta_{\ell,e}(h_i))\|_2^2$. The gate should weight the fitting loss, not be baked into the residual target; otherwise the inference-time multiplication by $\tilde{g}_{\ell,e}$ would double-count the gate. A rank-r residual can be estimated by short LoRA optimization, ridge regression on a linearized expert block, or randomized SVD of projected residuals. Exact fitting of nonlinear SwiGLU residuals is unnecessary for the first prototype; the assignment decision only needs a useful estimate of whether rank r captures a meaningful fraction of the residual energy. A later closed-loop variant should recompute downstream activations after each layer is compressed, analogous in spirit to sequential recomputation in REAM.

8.2 Distillation objective

For higher quality, train only small components: residual adapters, router biases, optional router low-rank adapters, and optional centroid correction adapters. The base dense attention and full centroid weights can

remain frozen. Let p_T and p_S be teacher and student output distributions, and let h_ℓ^T, h_ℓ^S be hidden states. A practical loss is

$$\begin{aligned}\mathcal{L} = \tau^2 \mathrm{KL}\left(p_T^\tau(\cdot|x) \| p_S^\tau(\cdot|x)\right) + \lambda_h \sum_{\ell \in \mathcal{S}_h} \|h_\ell^T - h_\ell^S\|_2^2 \\ + \lambda_r \sum_{\ell \in \mathcal{S}_r} \mathrm{KL}\left(\rho_\ell^T(\cdot|x) \| \rho_\ell^S(\cdot|x)\right) + \lambda_c \mathrm{Cost}(\tilde{M}),\end{aligned} \tag{28}$$

where ρ_ℓ is the router distribution or top-k distribution. Because teacher and student expert IDs may no longer match, the router KL should compare distributions after applying the compression map $m_\ell(e)$ from original experts to retained centroids or virtual IDs. Teacher mass assigned to deleted or absorbed experts should be aggregated according to the same map used by the student router; raw expert-ID KL is not meaningful after pruning, absorption, or virtual remapping. The cost penalty can include residual rank, number of active full centroids, load imbalance, and measured latency.

8.3 Router adaptation

Router adaptation is likely necessary when many virtual IDs map to fewer centroids. Three low-cost changes are useful:

1. **Bias correction:** adjust router biases so that surviving full centroids and virtual experts maintain intended activation rates.
2. **Virtual-ID regularization:** discourage excessive fragmentation among virtual experts that share a centroid unless their residuals are distinct. A practical penalty is the expected number of duplicate selected IDs per centroid, weighted down when residual norms or reconstruction gains are large.
3. **Centroid-load regularization:** prevent all tokens from collapsing onto a small number of centroids, which can damage quality and distributed load balance.

For QERAVE, router adaptation should be trained under the intended quantization format but evaluated with strict top-k stability diagnostics.

9 Memory, Compute, and Latency Model

9.1 Storage

Let ρ_F be the fraction of original experts kept full, ρ_V the fraction converted to virtual residual experts, ρ_A absorbed, and ρ_P pruned, with

$$\rho_F + \rho_V + \rho_A + \rho_P = 1. \tag{29}$$

Ignoring router rows and metadata, the expert storage ratio for RAVE is

$$\frac{\mathrm{Mem}_{\mathrm{experts}}(\mathrm{RAVE})}{\mathrm{Mem}_{\mathrm{experts}}(M)} \approx \rho_F + \rho_V \bar{\epsilon}, \tag{30}$$

where $\bar{\epsilon}$ is the average residual-to-expert parameter ratio. If virtual residuals keep separate router rows, the router metadata cost is small compared with expert MLP weights for large models.

For QERAVE with centroid bitwidth b_F and residual bitwidth b_V relative to original bitwidth b_0, the ratio becomes

$$\frac{\mathrm{Mem}_{\mathrm{experts}}(\mathrm{QERAVE})}{\mathrm{Mem}_{\mathrm{experts}}(M)} \approx \rho_F \frac{b_F}{b_0} + \rho_V \bar{\epsilon} \frac{b_V}{b_0}. \tag{31}$$

This is the deployment advantage of QERAVE, but the router and residual quantization risks must be tested.

9.2 Active full-expert compute

Let k be the original number of active experts. Let $u_\ell(x) = |\mathcal{U}_\ell(x)|$ be the number of unique full centroids executed by RAVE for token x. A first-order active expert compute ratio is

$$\frac{C_{\mathrm{active}}(\mathrm{RAVE})}{C_{\mathrm{active}}(M)} \approx \frac{\mathbb{E}_{x,\ell}[u_\ell(x)] + \mathbb{E}_{x,\ell}[v_\ell(x)]\bar{\epsilon}_{\mathrm{flop}}}{k}, \tag{32}$$

where $v_\ell(x)$ is the number of selected virtual residuals and $\bar{\epsilon}_{\text{flop}}$ is the residual compute ratio. This equation explains the main latency gap between memory compression and speed compression. If $\mathbb{E}[u_\ell(x)] \approx k$, RAVE may not be faster. If $k = 8$, $\mathbb{E}[u] \approx 5$, and residual overhead is $0.05k$, then the expert-MLP portion is roughly $5.4/8 \approx 0.675$ of the original. End-to-end speedup will be smaller because attention, sampling, communication, and framework overhead remain. A crude Amdahl-style bound is

$$S_{\text{end2end}} \lesssim \frac{1}{(1 - p_{\text{moe}}) + p_{\text{moe}}\, C_{\text{active}}(\text{RAVE})/C_{\text{active}}(M)}, \tag{33}$$

where p_{moe} is the fraction of baseline latency spent in expert execution and expert memory movement. This bound is optimistic unless dispatch and communication also improve.

9.3 Bandwidth and serving latency

Decode-time MoE serving is often bandwidth- and scheduling-sensitive. A useful latency proxy is

$$\text{Lat} \approx T_{\text{attn}} + T_{\text{router}} + T_{\text{dispatch}} + \max\{T_{\text{expert compute}}, T_{\text{expert memory}}, T_{\text{comm}}\} + T_{\text{residual}}. \tag{34}$$

RAVE helps only if it lowers one of the max terms enough to offset T_{residual} and extra routing bookkeeping. QERAVE can lower $T_{\text{expert memory}}$ through quantization, but may add dequantization or kernel complexity.

10 Computation Needed to Make the Method Useful

This section estimates the work needed to produce a meaningful experimental result. The units are planning estimates, not measured claims.

Let $F(N)$ denote one full calibration forward pass through the original model on N tokens while collecting router and expert statistics. REAM uses a calibration set of 3072 sequences of 512 tokens, or about 1.57M tokens, and reports that on Qwen3-30B-A3B-Instruct-2507 its non-sequential merging takes about 1 hour while sequential merging takes about 1.5 hours with 30 GB of VRAM [6]. This provides a useful external scale for planning but not a guarantee for RAVE, because residual fitting and grouped-kernel evaluation add different costs.

10.1 Relative search cost

Method	Calibration/search cost	Adaptation cost	What it buys
REAP	$\approx 1.0F(N)$ plus sorting.	None in the one-shot version.	Cheapest expert-level baseline; strong pruning reference.
REAM	≈ 1.0–$1.5F(N)$ plus grouping, alignment, and sequential recomputation.	None in the reported one-shot setup.	Strong merge/prune hybrid baseline.
RAVE one-shot	≈ 1.2–$2.5F(N)$ depending on residual candidate count and cache design.	None.	Tests whether virtual residuals help without training.
RAVE adapted	Same search plus residual/router training.	20M–100M tokens for prototype; 100M–2B for serious recovery.	Tests quality ceiling of virtual experts.
QERAVE	$\approx C_{\text{RAVE}} + (B-1)\eta_q F(N)$ for perturbation scoring, where B includes the clean pass and η_q is the relative noisy/low-bit pass cost.	Same as RAVE, ideally under deployment quantization.	Tests robustness and low-bit deployment.

Table 4: Relative compute estimates. $F(N)$ is one full calibration pass; C_{RAVE} denotes the chosen clean RAVE search cost; η_q is the relative cost of one noisy or low-bit pass and can exceed one without optimized kernels.

The table implies a recommended sequence. First implement RAVE one-shot and compare it to REAP and REAM at equal memory. Only after this is informative should QERAVE be added. Otherwise, the

project risks spending compute on quantization-noise machinery before proving that virtual experts matter. The factor $\eta_q < 1$ should be used only when the calibration stack actually uses faster low-bit kernels; in many research implementations, noisy or fake-quantized passes will cost approximately $F(N)$ and should be budgeted conservatively.

10.2 Residual fitting cost drivers

The main extra cost in RAVE is not the saliency pass; it is evaluating whether a compressed expert can be represented by a small residual around a candidate centroid. Suppose a layer has n experts, top-k routing, N calibration tokens, candidate-centroid count q, and a compressed-expert fraction $\rho_C = \rho_V + \rho_A + \rho_P$. The calibration pass observes about Nk selected expert-token pairs per layer, or Nk/n per expert under a balanced router. If candidate centroid outputs are not already cached, residual scoring can require roughly $q\rho_C Nk$ additional expert evaluations per layer. Relative to the active-expert portion of the original calibration pass, this is about $q\rho_C$ extra MoE-expert work. Thus the advertised 1.2–2.5$F(N)$ one-shot envelope assumes small q (for example $q \le 4$), compact projected caches, and early rejection of low-saliency experts. A naive all-pairs implementation can be much more expensive and should not be used for the first study.

Two cheap approximations should be implemented before expensive residual fitting. First, preselect candidate centroids using router-logit profiles and random projections of gated outputs. Second, use a low-dimensional sketch of $E_{\ell,e}(h) - E_{\ell,c}(h)$ to decide whether rank-r fitting is promising. Full LoRA-style fitting should be reserved for experts near the keep/absorb/prune decision boundary.

Caching also needs an explicit budget. Storing selected BF16 expert outputs costs approximately

$$NLkd\, b_{\text{bytes}}, \tag{35}$$

where L is the number of MoE layers and d is the model width. For $N = 3072 \times 512$, $L = 48$, $k = 8$, $d = 2048$, and BF16 outputs, this is about 2.5 TB in decimal units. Replacing full outputs with 64-dimensional random projections reduces the same cache to about 77 GB, before metadata. This is why RAVE should use streaming moments, compact sketches, and candidate-level caches rather than all expert outputs.

For wall-clock planning, let θ_{cal} be calibration throughput in tokens/s on the chosen hardware and let m be the search multiplier from Table 4. Then

$$T_{\text{search}} \approx \frac{mN}{\theta_{\text{cal}}}. \tag{36}$$

Adapter recovery over N_d distillation tokens costs roughly

$$T_{\text{adapt}} \approx \frac{\chi N_d}{\theta_{\text{train}}}, \tag{37}$$

where χ is a forward-equivalent multiplier. Online teacher-plus-student distillation often has $\chi > 1$; caching teacher logits or hidden states can reduce compute but increases storage.

10.3 Concrete planning budgets

Budget A: minimal useful prototype. Use one 128-expert MoE model, one calibration mixture, and no adaptation. Collect $N \approx 1.57$M calibration tokens, matching the REAM-scale calibration setting over C4, NuminaMath, and The Stack-style code data [6, 27–29]. Run REAP, REAM, and RAVE one-shot at 25% and 50% full-expert reduction. Fit residual ranks $r \in \{0, 4, 8, 16\}$ only for the top few centroid candidates per compressed expert. The expected planning envelope is roughly a few multiples of REAM's one-shot compression time, dominated by residual candidate fitting and validation. This budget is enough to answer whether RAVE is worth further investment.

Budget B: publishable 30B-scale study. Use the same 128-expert model class but sweep calibration mixtures across general, math, and code data, for example C4, NuminaMath, and The Stack-style code corpora [27–29]. Evaluate original, frequency pruning, REAP, REAM, RAVE rank-0, RAVE rank-4/8/16, RAVE with active-centroid penalty, and QeRAVE with $B = 4$ perturbations. Also include quantized original, quantized REAP, quantized REAM, and post-hoc quantized RAVE whenever QeRAVE uses low-bit deployment. Add at least short residual/router distillation runs of 20M and 100M tokens. This budget is needed to separate four effects: virtual experts, active-centroid thinning, generic quantization, and quantization-noise robustness.

Budget C: large 512-expert validation. Validate on a 512-expert model such as Qwen3-Next-80B-A3B or Qwen3-Coder-Next-style systems, where the number of redundant or tail-specialized experts is larger [6, 30]. Candidate search must be approximate: store compact projections, preselect centroid neighbors, and avoid all-pairs expert output caching. At this scale, RAVE is useful only if it shows either near-REAM quality at lower active full-centroid count or better generative quality at the same storage.

10.4 Evaluation compute

Compression search is only part of the cost. Generative benchmarks dominate wall-clock time for large models. A useful study should reserve enough compute for:

1. multiple-choice benchmarks to compare with prior work;
2. generative math/code/instruction benchmarks, because REAP and REAM emphasize that MC and GEN can disagree;
3. latency traces for prefill and decode at batch sizes 1, 4, and 16;
4. ablations over rank, active-centroid target, calibration mixture, and adaptation length.

A method that improves average benchmark score but cannot run faster in a real grouped kernel should be reported as memory-quality compression, not speed compression.

10.5 Search complexity and cache footprint

The naive pairwise expert comparison cost is $O(LE^2N)$ if every layer has E experts and every expert pair is evaluated on all N calibration tokens. This is acceptable only for small debugging models. For 128 experts, all-pairs comparison already implies 16,384 expert pairs per layer; for 512 experts, it implies 262,144 pairs per layer. A practical implementation should therefore use two-stage filtering: first rank centroids by cheap router-profile and saliency features, then evaluate residual reconstruction only for the top 4–8 centroid candidates per compressed expert. Cached tensors should also be projections or sufficient statistics rather than full expert outputs for all pairs.

10.6 Forward-pass-equivalent accounting

The calibration scale above implies $N_{\text{cal}} = 3072 \cdot 512 = 1{,}572{,}864$ tokens. Adaptation quickly dominates search. A 20M-token recovery run is about $12.7N_{\text{cal}}$; a 100M-token run is about $63.6N_{\text{cal}}$; a 2B-token run is about $1272N_{\text{cal}}$. If online distillation requires a teacher forward, a student forward, and a backward pass through residual/router parameters, its cost can be several forward-pass equivalents per token. Caching teacher logits or hidden states lowers online compute but can require substantial storage. Thus a fair compute report should separate (i) one-time compression search, (ii) residual/router adaptation, and (iii) benchmark evaluation.

For QERAVE, $B = 4$ perturbations with a low-bit pass cost $\eta_q \in [0.5, 0.8]$ would add roughly 2.0–3.2$F(N)$ to the clean calibration pass before residual fitting. This may still be cheaper than even a short adaptation run, but it is not free. QERAVE is useful only if those extra passes improve cross-mixture robustness, reduce the amount of adaptation needed, or enable lower-bandwidth deployment.

10.7 Fair comparison rules

Comparisons should be reported in at least two regimes. In the *memory-quality* regime, methods are matched by expert-storage ratio and evaluated for quality. In the *latency-quality* regime, methods are matched by measured tokens/s, measured decode latency, or a strict proxy such as $\mathbb{E}[u_\ell(x)]/k$ under the same serving kernel. A method that keeps top-k active full experts unchanged should not be presented as faster solely because it stores fewer experts. Conversely, a method that reduces active centroids but sacrifices storage less aggressively should be compared at equal latency as well as equal memory.

11 Full Experimental Protocol for Public Validation

11.1 Models

A staged plan is preferable:

1. a small MoE model for implementation debugging;
2. a 128-expert MoE model comparable to the main REAM setting;
3. a 512-expert model for stress testing candidate search and serving behavior;
4. optionally, a code-specialized MoE model where tail experts may matter more.

11.2 Baselines

At minimum, compare:
1. original uncompressed model;
2. frequency pruning;
3. REAP;
4. REAM;
5. RAVE rank-0, equivalent to centroid absorption with virtual remapping;
6. RAVE rank-4/8/16 one-shot;
7. RAVE with active-centroid penalty;
8. RAVE adapted with residual/router distillation;
9. RAVE with post-hoc quantization;
10. quantized original, quantized REAP, and quantized REAM;
11. QERAVE perturbation scoring and low-bit deployment.

The most important ablation is RAVE versus QERAVE, because it determines whether quantization noise is essential or merely helpful. The quantized baselines are required to avoid attributing ordinary post-training quantization gains to the QERAVE search procedure.

11.3 Benchmarks

A useful benchmark mix should include both discriminative and generative tasks. Prior compression work warns that multiple-choice and generative quality can move differently under MoE compression [5, 6]. The minimum suite should include:
- multiple-choice: MMLU, ARC-Challenge, HellaSwag, BoolQ, PIQA, WinoGrande, and OpenBookQA [31–38];
- math generation: GSM8K and, where licensing permits, competition-style math [28, 39];
- code generation: HumanEval and LiveCodeBench [40, 41];
- instruction following and structured generation;
- latency and memory traces under realistic serving configurations.

11.4 Ablations

The core ablations are:
1. RAVE clean scoring versus QERAVE perturbation scoring;
2. mean saliency only versus saliency plus residual reconstruction;
3. rank-0 absorption versus rank-4/8/16 residuals;
4. active-centroid penalty on and off;
5. shared-base grouped kernel versus naive execution;
6. calibration mixtures among general, math, and code;
7. sequential recomputation on and off;
8. no adaptation versus 20M/100M-token residual/router adaptation;
9. centroid quantization and residual quantization separately;
10. QERAVE versus post-hoc quantized RAVE with identical bitwidths.

11.5 Success criteria

The first clear win would be one of the following:
- **Memory-quality win:** at the same expert-storage budget as REAM, RAVE achieves better generative quality or lower calibration sensitivity.
- **Latency-quality win:** at the same quality as REAM, RAVE executes fewer unique full centroids and improves real decode throughput.

- **Robustness win:** QeRAVE perturbation scoring transfers better across calibration mixtures than clean RAVE scoring.
- **Deployment win:** QeRAVE low-bit centroids/residuals improve tokens/s or VRAM without materially hurting quality relative to RAVE.

12 Implementation Notes

Caching. Caching every expert output for every token is expensive and can quickly reach terabytes. Store output norms, router logits, compact random projections, and residual statistics for a limited set of candidate centroid pairs. For 512-expert models, use approximate nearest-neighbor search or two-stage candidate filtering, and avoid all-pairs expert-output materialization.

Centroid choice. Centroids should not be chosen solely by top saliency. Add coverage over routing regions, activation clusters, and domains. Otherwise, the centroid set may overrepresent high-frequency general-text experts and underrepresent math/code specialists.

Residual fitting. Start with rank-0, rank-4, rank-8, and rank-16. If rank-16 does not reduce residual error enough, the expert should probably remain full rather than be forced into a weak residual.

Router remapping. Virtual experts may retain their original router rows, but the router should be regularized so that several virtual IDs sharing a centroid do not fragment top-k in a way that destroys active-centroid speedups.

Quantization. For the base RAVE paper, quantization should be an ablation, not a dependency. For QeRAVE, freeze the intended centroid quantization format before final residual adaptation whenever possible. Keep routers in BF16/FP16 until lower precision is proven safe.

Kernel. The performance-critical component is a grouped virtual-expert kernel: route virtual IDs, aggregate gates by centroid, compute each full centroid once, compute residuals, and accumulate. Without this kernel, speed claims should be avoided.

Sequential recomputation. Implement both one-shot and sequential modes behind the same interface. The one-shot mode is the cheapest baseline; the sequential mode diagnoses whether stale later-layer statistics are a major source of quality loss.

Reproducibility. Report the exact calibration mixture, random seeds for centroid candidate sampling and perturbation bootstraps, router temperature/noise settings, residual rank allocation rule, and whether comparisons are matched by storage, active centroids, or measured latency.

13 Limitations and Risks

Internal pilot scope. The internal `ai-imo-2` study is useful because it is strictly proof judged and proof oriented, and includes the original SU-01 direct run. It is still only one 40-problem private benchmark with a severe 8192-token cap. It does not establish performance on public math benchmarks, code generation, instruction following, memory-quality MoE compression, or latency in a serving stack. The compute multipliers remain engineering estimates and should be replaced with wall-clock measurements once an implementation exists.

RAVE may not beat REAM without adaptation. One-shot residual fitting may be too crude, especially for nonlinear expert differences. If so, RAVE may need short residual/router distillation to become competitive.

Kernel dependence. The latency argument depends on grouped centroid execution. A naive implementation may be slower than the original model despite lower storage.

Calibration sensitivity. REAM shows that calibration mixture can strongly affect MC and GEN outcomes. RAVE inherits this risk, and QeRAVE only hypothesizes a mitigation through perturbation ensembles.

Residual rank limits. Some experts may not be well approximated by a low-rank delta around any centroid. The assignment algorithm must allow these experts to remain full.

Residual implementation choice. A residual can be implemented as LoRA deltas inside expert matrices or as a separate small residual branch. Matrix LoRA is storage-efficient but interacts with the nonlinear SwiGLU path; a separate branch is easier to fit but may be less kernel-friendly. This is an implementation choice that must be ablated.

Router instability. Small changes in router logits can change top-k sets. Router quantization should be treated conservatively. In addition, rank-zero absorption and virtual-ID remapping can change the effective router competition even without quantization, so all compressed models should report top-k overlap and centroid-load diagnostics.

Virtual-ID collisions. Selecting many virtual IDs that share one centroid can improve latency but reduce the diversity of full expert functions. The active-centroid penalty must be swept rather than fixed by assumption.

Distributed serving complexity. Reduced local compute can be offset by communication overhead if centroids and residuals are poorly placed across devices. Serving experiments should report expert load balance, capacity overflows or dropped tokens if applicable, and per-device communication volume.

14 Conclusion

RAVE proposes a structural middle path between pruning and merging for sparse MoE LLM compression. It keeps important experts full, prunes robustly cold experts, absorbs centroid-equivalent experts, and preserves ambiguous tail specialists as low-rank virtual residual experts. This makes RAVE the clean core method: it can be evaluated without quantization and directly compared to REAP and REAM at equal memory and latency budgets.

QeRAVE is best viewed as an extension. It adds quantization-noise-stability scoring and optional low-bit deployment, inspired by QeRL's observation that quantization noise can be useful rather than merely harmful. The recommended research path remains: implement clean RAVE first, validate that virtual experts improve the quality-memory or quality-latency frontier, and then add QeRAVE to test whether perturbation ensembles improve robustness and whether low-bit centroids/residuals improve deployment.

The internal SU-01-family pilot gives an encouraging first signal. Under strict direct 8192-token decoding on 40 held-out synthetic olympiad-style problems, `rave_64` achieved the highest total proof-judged score, 154/280, and tied the best pass count with 9/40 complete or essentially complete proofs. This is not yet a public benchmark claim or a serving-speed claim, but it is strong enough to justify the next stage: public, latency-matched validation with full ablations over pruning, merging, virtual residual rank, active-centroid grouping, quantization, and residual/router adaptation.

References

[1] Jacobs et al. Adaptive Mixtures of Local Experts. *Neural Computation*. 1991.

[2] Shazeer et al. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. *International Conference on Learning Representations*. 2017.

[3] Lepikhin et al. GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding. *arXiv preprint arXiv:2006.16668*. 2020.

[4] Fedus et al. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. *Journal of Machine Learning Research*. 2022.

[5] Lasby et al. REAP the Experts: Why Pruning Prevails for One-Shot MoE Compression. *arXiv preprint arXiv:2510.13999*. 2025.

[6] Jha et al. REAM: Merging Improves Pruning of Experts in LLMs. *arXiv preprint arXiv:2604.04356*. 2026.

[7] Jiang et al. Mixtral of Experts. *arXiv preprint arXiv:2401.04088*. 2024.

[8] Yang et al. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388*. 2025.

[9] Frantar and Alistarh. SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot. *International Conference on Machine Learning*. 2023.

[10] Sun et al. A Simple and Effective Pruning Approach for Large Language Models. *International Conference on Learning Representations*. 2024.

[11] Liu et al. Efficient Expert Pruning for Sparse Mixture-of-Experts Language Models: Enhancing Performance and Reducing Inference Costs. *arXiv preprint arXiv:2407.00945*. 2024.

[12] Bai et al. DiEP: Adaptive Mixture-of-Experts Compression through Differentiable Expert Pruning. *Advances in Neural Information Processing Systems*. 2025.

[13] Yang et al. MoE-I2: Compressing Mixture of Experts Models through Inter-Expert Pruning and Intra-Expert Low-Rank Decomposition. *Findings of the Association for Computational Linguistics: EMNLP*. 2024.

[14] Xie et al. MoE-Pruner: Pruning Mixture-of-Experts Large Language Model using the Hints from Its Router. *arXiv preprint arXiv:2410.12013*. 2024.

[15] Li et al. Merge, Then Compress: Demystify Efficient SMoE with Hints from Its Routing Policy. *arXiv preprint arXiv:2310.01334*. 2023.

[16] Chen et al. Retraining-Free Merging of Sparse MoE via Hierarchical Clustering. *arXiv preprint arXiv:2410.08589*. 2024.

[17] Li et al. Sub-MoE: Efficient Mixture-of-Expert LLMs Compression via Subspace Expert Merging. *Proceedings of the AAAI Conference on Artificial Intelligence*. 2026.

[18] Hu et al. LoRA: Low-Rank Adaptation of Large Language Models. *International Conference on Learning Representations*. 2022.

[19] Dettmers et al. QLoRA: Efficient Finetuning of Quantized LLMs. *Advances in Neural Information Processing Systems*. 2023.

[20] Frantar et al. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. *International Conference on Learning Representations*. 2023.

[21] Xiao et al. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. *International Conference on Machine Learning*. 2023.

[22] Huang et al. QeRL: Beyond Efficiency–Quantization-enhanced Reinforcement Learning for LLMs. *arXiv preprint arXiv:2510.11696*. 2025.

[23] Hinton et al. Distilling the Knowledge in a Neural Network. *arXiv preprint arXiv:1503.02531*. 2015.

[24] Vaswani et al. Attention Is All You Need. *Advances in Neural Information Processing Systems*. 2017.

[25] Li et al. Achieving Gold-Medal-Level Olympiad Reasoning via Simple and Unified Scaling. *arXiv preprint arXiv:2605.13301*. 2026.

[26] Simplified Reasoning. Simplified-Reasoning/SU-01. *Hugging Face model repository*. 2026.

[27] Raffel et al. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*. 2020.

[28] Li and others. NuminaMath: The Largest Public Dataset in AI4Maths with 860K Pairs of Competition Math Problems and Solutions. *Hugging Face Dataset*. 2024.

[29] Kocetkov et al. The Stack: 3 TB of Permissively Licensed Source Code. *Transactions on Machine Learning Research*. 2023.

[30] Cao et al. Qwen3-Coder-Next Technical Report. *arXiv preprint arXiv:2603.00729*. 2026.

[31] Hendrycks et al. Measuring Massive Multitask Language Understanding. *International Conference on Learning Representations*. 2021.

[32] Clark et al. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *arXiv preprint arXiv:1803.05457*. 2018.

[33] Zellers et al. HellaSwag: Can a Machine Really Finish Your Sentence?. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 2019.

[34] Clark et al. BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*. 2019.

[35] Bisk et al. PIQA: Reasoning about Physical Commonsense in Natural Language. *Proceedings of the AAAI Conference on Artificial Intelligence*. 2020.

[36] Sakaguchi et al. WinoGrande: An Adversarial Winograd Schema Challenge at Scale. *Proceedings of the AAAI Conference on Artificial Intelligence*. 2020.

[37] Mihaylov et al. Can a Suit of Armor Conduct Electricity? A New Dataset for Open Book Question Answering. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2018.

[38] Gao et al. A Framework for Few-Shot Language Model Evaluation. *Zenodo*. 2023.

[39] Cobbe et al. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168*. 2021.
[40] Chen et al. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374*. 2021.
[41] Jain et al. LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code. *International Conference on Learning Representations*. 2025.