COORDINATED MIXED-PRECISION KV CACHE QUANTIZATION FOR EFFICIENT LLM-BASED MULTI-AGENT SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM-based multi-agent systems solve complex tasks through problem decomposition, specialized tool use, and iterative verification. However, collaboration produces multiple growing key–value (KV) caches that can coexist in memory, making cache storage a bottleneck. KV-cache quantization reduces this footprint, but independently applying single-model policies ignores differences in agents' precision needs. This can over-compress sensitive computations while spending precision on more tolerant caches, potentially degrading collective reasoning. In this paper, we introduce **Co**ordinated **Mix**ed-Precision **KV** Cache Quantization (CoMixKV), a novel method that integrates online sensitivity estimation with joint bit-width allocation under a shared memory budget. To our knowledge, CoMixKV is the first mixed-precision KV-cache quantization framework to jointly allocate precision across simultaneously resident agent caches under such a budget. An online estimator derives each agent's sensitivity from a local attention-error bound using recent queries, keys, and values, without calibration data, trial quantization, or additional model passes. A shared controller combines these scores with remaining quantizable capacity and byte-level storage costs to minimize total estimated error. Allocations are updated as agents start or finish and fresh observations become available. Selected widths apply only to newly quantized blocks; recent states retain native precision and previously quantized blocks remain unchanged. Across four benchmarks and three LLM backbones at average 2-bit, 3-bit, and 4-bit KV-cache settings, CoMixKV achieves strong accuracy compared with the evaluated quantization baselines. At the average 3-bit setting on Gemma-4-31B-it, CoMixKV achieves 68.35% average accuracy, outperforming the strongest quantized baseline by 0.59%. On Qwen3-32B, CoMixKV further reduces peak KV-cache memory by 32.08% and achieves a 1.38× inference speedup over FullKV.

1 INTRODUCTION

Large language model (LLM)-based multi-agent systems solve complex tasks through collaboration among specialized agents (Guo et al., 2024; Li et al., 2024a; Chen et al., 2024). Agents take complementary roles, use tools, and exchange intermediate results (Yang et al., 2024; He et al., 2025; Wu et al., 2024; Hong et al., 2024; Fourney et al., 2024). This collaboration, however, increases memory consumption because each agent maintains a local history whose attention states are stored in a key–value (KV) cache. Distinct histories form separate cache branches, and overlapping agent lifetimes can make multiple caches resident simultaneously, substantially increasing aggregate KV storage (Pan et al., 2025; Wang et al., 2025; Gao et al., 2024a; Jin et al., 2025).

KV-cache quantization reduces this footprint by storing keys and values at lower precision (Hooper et al., 2024; Hariri et al., 2026; Zhang et al., 2025c). Representative methods include **KIVI** (Liu et al., 2024c), **PolarQuant** (Wu et al., 2025), **TurboQuant** (Zandieh et al., 2026), **PM-KVQ** (Liu et al., 2026), and **PatternKV** (Zhang et al., 2026a). These methods improve quantization representations or precision allocation within individual caches (Liu et al., 2024c; He et al., 2024; Hooper et al., 2024; Zhang et al., 2026b; Zuo et al., 2026; Shen et al., 2026), but do not coordinate precision across simultaneously resident agent caches.

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

Applying such policies independently leaves two challenges. First, different agents and layers process different queries and histories, so the same precision can produce substantially different attention errors. Uniform or fixed allocation may therefore waste precision on tolerant computations while under-protecting sensitive ones. Second, cache residency changes as agents start, pause, and finish. Since previously quantized blocks retain their precision, later decisions affect only future blocks and must account for both remaining quantizable capacity and already committed storage. Effective compression therefore requires coordinated precision allocation across changing agent–layer computations under a shared memory budget.

We introduce **Co**ordinated **Mix**ed-Precision **KV** Cache Quantization (CoMixKV), which combines online sensitivity estimation with agent–layer bit-width allocation across private resident caches. To our knowledge, CoMixKV is the first mixed-precision KV-cache quantization framework to coordinate precision across simultaneously resident agent caches under a shared memory budget. Agents exchange text while retaining private KV states, and the controller coordinates precision rather than transferring cache tensors.

CoMixKV estimates agent–layer quantization sensitivity from recent queries, keys, and values produced during ordinary inference. The score is derived from a local attention-error bound and requires no calibration data, trial quantization, or additional model passes. Based on these scores, a shared controller assigns lower-, target-, and higher-bit widths across layers while respecting the byte budget. More sensitive layers receive higher precision, while balanced assignments preserve the target mean packed-code width. Allocations are refreshed as agent residency and observations change, and new assignments apply only to newly quantized blocks; existing blocks retain their stored precision.

We evaluate CoMixKV on four benchmarks spanning knowledge search, agentic reasoning, mathematical reasoning, and scientific reasoning, using three LLM backbones and average 2-, 3-, and 4-bit KV-cache settings. Against three KV-cache quantization baselines, CoMixKV achieves strong accuracy across the evaluated settings. At the average 3-bit setting on Gemma-4-31B-it, CoMixKV achieves 68.35% average accuracy, outperforming the strongest quantized baseline by 0.59%. Moreover, on Qwen3-32B, CoMixKV reduces peak KV-cache memory by 32.08% and achieves a 1.38× inference speedup over FullKV, demonstrating improvements in both task accuracy and system efficiency.

2 Related Work

LLM-based multi-agent systems. LLM agents interleave reasoning, tool use, and environment feedback across turns (Yao et al., 2023; Masterman et al., 2024; Zhang et al., 2025d). Multi-agent systems extend this process through role-based collaboration (Hong et al., 2024; Wu et al., 2024; Li et al., 2023; Qian et al., 2024) and verification (Zhang et al., 2026c). Their growing local histories and overlapping cache lifetimes increase aggregate KV-cache costs. Recent methods address this burden through cross-context reuse (Ye et al., 2025), workflow-guided retention and prefetching (Pan et al., 2025), coordinated scheduling and offloading (Bian et al., 2025), or shared compressed pools (Patel & Joshi, 2026). CoMixKV instead coordinates precision across private resident caches under a shared memory budget. Agents exchange textual reports rather than KV states, while their starts and returns change which caches compete for memory.

KV cache quantization. KV cache compression (Zhang et al., 2023; Liu et al., 2023; Tang et al., 2024; Feng et al., 2025; Ge et al., 2024) is an effective method for reducing long-context inference costs (Zhang et al., 2024; Liu et al., 2024a; Chitty-Venkata et al., 2026; Hu et al., 2026); specifically, KV cache quantization (Kriuk & Ng, 2026; Honavar & GVSL, 2026) reduces memory use without removing tokens through outlier handling (Hooper et al., 2024), adaptive precision (He et al., 2024; Dong et al., 2024; Zhang et al., 2026b; Zuo et al., 2026), and alternative representations (Zhang et al., 2025b; Kang et al., 2024). KIVI (Liu et al., 2024c) quantizes keys channel-wise and values token-wise, retaining native-precision tails. PolarQuant (Wu et al., 2025) uses polar quantization for RoPE-coupled keys; TurboQuant (Zandieh et al., 2026) combines random rotations, scalar quantization, and residual correction. PM-KVQ (Liu et al., 2026) reduces precision with sequence growth using calibrated layer sensitivity; PatternKV (Zhang et al., 2026a) quantizes residuals from online-mined patterns. TriAxialKV (Shen et al., 2026) calibrates precision by recency, modality, and semantic

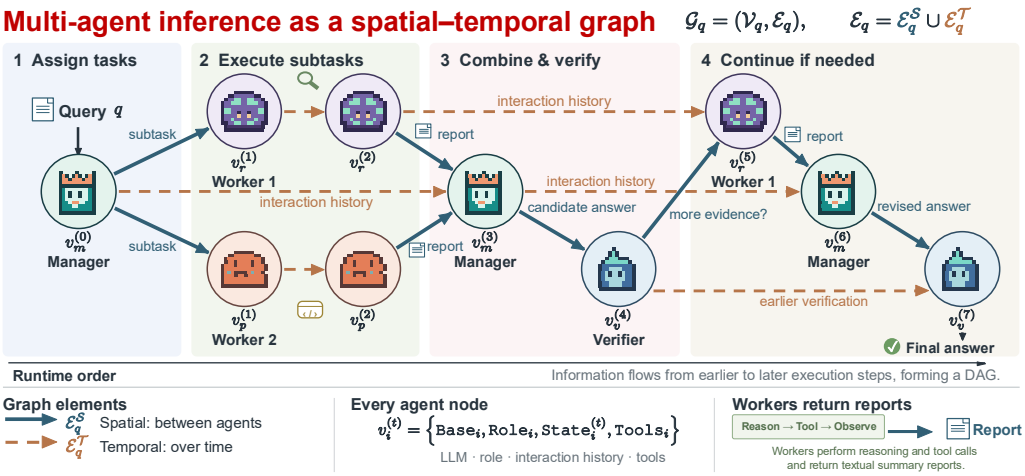


Figure 1: **Multi-agent inference as a spatial-temporal graph.** Solid blue arrows denote spatial communication ($\mathcal{E}_q^{\mathrm{S}}$), while dashed orange arrows denote temporal information flow ($\mathcal{E}_q^{\mathrm{T}}$). Node subscripts m, r, p, and v denote the manager, Worker 1, Worker 2, and verifier, respectively.

role. CoMixKV uses online Q/K/V sensitivity and exact per-invocation packed-code balance within a shared cache budget.

3 Preliminaries

We describe multi-agent inference as a spatial-temporal graph and use this view to formulate KV cache quantization over the resulting execution structure.

3.1 Multi-Agent Inference as a Spatial-Temporal Graph

Execution graph. Given a query $\boldsymbol{q}$, we represent multi-agent execution as a directed spatial-temporal graph (Zhang et al., 2025a; Zhuge et al., 2024; Liu et al., 2024b):

$$\mathcal{G}_{\boldsymbol{q}} = (\mathcal{V}_{\boldsymbol{q}}, \mathcal{E}_{\boldsymbol{q}}), \qquad \mathcal{E}_{\boldsymbol{q}} = \mathcal{E}_{\boldsymbol{q}}^{\mathrm{S}} \cup \mathcal{E}_{\boldsymbol{q}}^{\mathrm{T}}. \tag{1}$$

The node set $\mathcal{V}_{\boldsymbol{q}}$ contains execution states $v_i^{(t)}$, each representing agent i at step t:

$$v_i^{(t)} = \left\{\texttt{Base}_i, \texttt{Role}_i, \texttt{State}_i^{(t)}, \texttt{Tools}_i\right\}, \tag{2}$$

where $\texttt{Base}_i = \mathcal{M}$ is the language model, and $\texttt{Role}_i$, $\texttt{State}_i^{(t)}$, and $\texttt{Tools}_i$ specify its responsibility, local interaction history, and available tools, respectively (Wu et al., 2024). Spatial edges in $\mathcal{E}_{\boldsymbol{q}}^{\mathrm{S}}$ carry task assignments, worker reports, or verification feedback between agents (Wu et al., 2024; Gao et al., 2024b), while temporal edges in $\mathcal{E}_{\boldsymbol{q}}^{\mathrm{T}}$ carry information from earlier to later execution states. All edges follow runtime order, forming a directed acyclic graph (DAG) (Bondy et al., 1976).

Multi-agent communication. Each agent follows a tool-calling ReAct process (Yao et al., 2023), generating its output from the query, role, local history, and incoming temporal and spatial information:

$$\boldsymbol{y}_i^{(t)} \sim \mathcal{P}\left(\cdot \mid \boldsymbol{q}, \texttt{Role}_i, \texttt{State}_i^{(t)}, \boldsymbol{x}_i^{\mathrm{T},(t)}, \boldsymbol{x}_i^{\mathrm{S},(t)}\right), \tag{3}$$

where $\mathcal{P}$ is the model's conditional output distribution, and $\boldsymbol{x}_i^{\mathrm{T},(t)}$ and $\boldsymbol{x}_i^{\mathrm{S},(t)}$ denote earlier-state information and messages from other agents, respectively. The output $\boldsymbol{y}_i^{(t)}$ may include reasoning, tool calls, task assignments, reports, or a final answer.

In our hierarchical workflow, as shown in Figure 1, the manager delegates subtasks to specialized workers (Fourney et al., 2024), which reason and use tools while keeping detailed histories local

and returning only textual reports. The manager combines these reports into a candidate answer for the verifier. When more evidence is needed, the runtime starts another worker invocation and supplies its report, together with the original candidate answer, to a fresh verifier invocation. These exchanges form related execution branches rather than a single linear history.

3.2 Graph-Conditioned KV-Cache Quantization

Graph-induced cache branches. Each invocation a of agent i maintains a local history $\boldsymbol{h}_a$, formed by the inputs in Eq. (3) and its generated tokens. The corresponding keys and values are stored in cache (Kwon et al., 2023; Liu et al., 2024c; Hooper et al., 2024):

$$\mathcal{KV}_a = \left\{(\boldsymbol{K}_a^{(\ell)}, \boldsymbol{V}_a^{(\ell)})\right\}_{\ell \in \mathcal{L}}, \qquad \boldsymbol{K}_a^{(\ell)}, \boldsymbol{V}_a^{(\ell)} \in \mathbb{R}^{N_a \times D_\ell}. \tag{4}$$

Here, N_a counts cached tokens, $\mathcal{L}$ contains the attention layers to be quantized, and D_ℓ is the total feature dimension across KV heads in layer ℓ.

Temporal edges within an invocation extend its local history by retaining existing KV states and appending new entries (Yu et al., 2025; Gao et al., 2024a), whereas spatial edges transfer messages that are incorporated into the receiver's local history and encoded into its own KV states. As shown in Figure 2, the manager retains its cache while a nested worker runs. The worker's cache is released on return, but its textual report remains in the manager's history. The manager's cache is released before the separate verifier run begins. Thus, execution order determines which cache branches coexist and share the memory budget.

Problem formulation. Eq. (4) defines $\mathcal{KV}_a$ as the keys and values stored across attention layers for one agent invocation a. Each invocation maintains the KV states computed from its own local history. Several such caches can occupy memory at the same time, even when the agents are not generating tokens simultaneously. For example, the manager keeps its cache while waiting for a worker's report, and the worker maintains its own cache during this period. Both caches therefore need to be considered when allocating the available memory.

We jointly choose the bit widths for these caches and update the choices during execution. Each such decision is called an *allocation round*. A new round can occur when an invocation starts or finishes, or when new observations become available for estimating quantization sensitivity. At allocation round r, let $\mathcal{A}_r$ contain all invocations whose caches are still stored in memory, including invocations waiting for other agents to return. Collecting their individual caches $\mathcal{KV}_a$ gives

$$\mathcal{KV}^{(r)} = \left\{\mathcal{KV}_a : a \in \mathcal{A}_r\right\}. \tag{5}$$

Each element of $\mathcal{KV}^{(r)}$ is therefore the cache of one invocation, containing its keys and values across attention layers. These caches remain separate, but we select the bit widths for their newly quantized blocks jointly, accounting for their quantization sensitivities and storage requirements under a shared memory budget. The controller and budget are scoped to one question's agent team; unrelated questions use separate controller states.

For each invocation, we keep the KV states of the most recent R tokens at the model's original precision. In each layer, older states are grouped into blocks of P consecutive tokens for quantization. We set a target packed-code mean $b_{\text{avg}} \in \{2, 3, 4\}$ and allow three choices: $\mathcal{B} = \{b_{\text{avg}} - 1, b_{\text{avg}}, b_{\text{avg}} + 1\}$. Thus, the audited 2-, 3-, and 4-bit runs use candidates $\{1, 2, 3\}$, $\{2, 3, 4\}$, and $\{3, 4, 5\}$, respectively. The target and candidate set remain fixed throughout an experiment; the target is the mean packed-code width, while individual sealed blocks may use any candidate.

At allocation round r, we choose $b_{a,\ell,r} \in \mathcal{B}$ for each invocation a and KV layer ℓ. Until the next round, this width applies to newly quantized key and value blocks in that layer. Already quantized blocks, called *sealed* blocks, keep their existing widths. Let $\mathcal{B}_{a,j}^{(\ell)} = (\boldsymbol{K}_{a,j}^{(\ell)}, \boldsymbol{V}_{a,j}^{(\ell)})$ contain the keys and values of the j-th block from invocation a in layer ℓ. When first quantized using the width selected at round r, the block is stored as

$$\overline{\mathcal{B}}_{a,j}^{(\ell)} = \mathcal{Q}_{b_{a,\ell,r}}\left(\mathcal{B}_{a,j}^{(\ell)}\right). \tag{6}$$

Here, $\mathcal{Q}_b$ quantizes keys and values directly at the selected b-bit resolution and stores the codes and parameters needed to reconstruct them for attention. The packed implementation stores a fixed

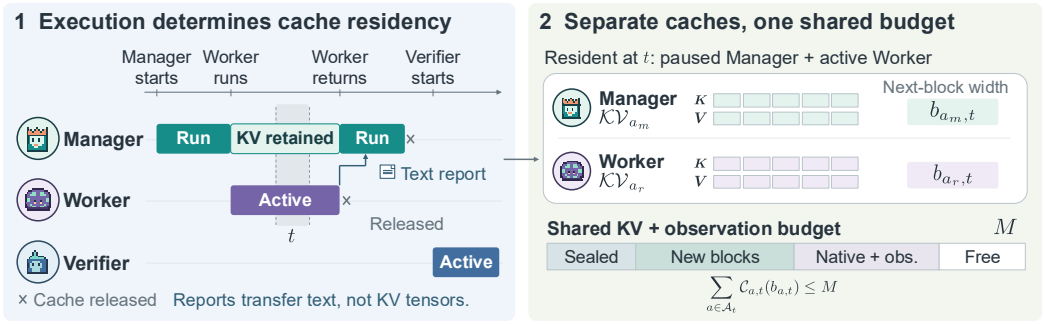


Figure 2: **Graph-conditioned cache residency and shared-budget quantization.** Left: cache lifetimes across manager, worker, and verifier runs. Right: agent–layer width choices for resident private KV caches under a shared byte envelope.

base payload plus refinement bitplanes for widths above that base; physical refinement pages are committed on demand. In the audited average 2-bit runs the base is one bit, allowing 1-bit blocks to offset the bytes of 3-bit blocks. The average 3- and 4-bit runs use 2- and 3-bit bases, respectively. Following KIVI (Liu et al., 2024c), keys are grouped across tokens within each feature, while values are grouped across features within each token. Let T_a be the cache capacity in tokens and $z_{a,\ell,r}$ the number of already quantized tokens in layer ℓ. The number of additional quantizable blocks in that layer is

$$n_{a,\ell,r} = \max\left(0, \left\lfloor \frac{T_a - R}{P} \right\rfloor - \frac{z_{a,\ell,r}}{P}\right). \tag{7}$$

This counts remaining quantizable capacity, not predicted generation length. Let $s_{a,\ell,r} \geq 0$ denote the sensitivity of layer ℓ computed from a small window of recent queries, keys, and values. The allocation surrogate for a candidate assignment $\boldsymbol{b} = (b_{a,\ell})$ is

$$\widehat{\mathcal{E}}_r(\boldsymbol{b}) = \sum_{a \in \mathcal{A}_r} \sum_{\ell \in \mathcal{L}} \frac{s_{a,\ell,r}}{(2^{b_{a,\ell}} - 1)^2}. \tag{8}$$

The estimate treats recent sensitivity as representative of future blocks; it does not measure their actual future error. The audited allocation chooses each agent–layer vector under exact per-invocation balance and full three-width coverage:

$$\boldsymbol{b}_r \in \operatorname*{arg\,min}_{\boldsymbol{b} \in \mathcal{B}^{|\mathcal{A}_r|L}} \widehat{\mathcal{E}}_r(\boldsymbol{b}) \quad \text{s.t.} \quad \sum_{\ell \in \mathcal{L}} b_{a,\ell} = L b_{\text{avg}},\ \{b_{a,\ell} : \ell \in \mathcal{L}\} = \mathcal{B}\ (\forall a),\ \sum_{a \in \mathcal{A}_r} \mathcal{C}_{a,r}(\boldsymbol{b}_a) \leq M_r^{\text{eff}}. \tag{9}$$

The implementation strengthens the mean constraint to equal counts of $b_{\text{avg}} - 1$ and $b_{\text{avg}} + 1$ layers for each invocation, leaving at least one layer at b_{avg}. Equal per-layer KV geometry makes this balance exact for each complete group written across layers. Sealed blocks contribute to memory cost but not the prospective objective because their precision cannot change. The selection rule and physical audit are specified below.

4 Methodology

At allocation round r, the formulation above requires a sensitivity score $s_{a,\ell,r}$ for each agent and layer, and a byte reservation $\mathcal{C}_{a,r}(\boldsymbol{b}_a)$ for each proposed layer vector. We obtain these quantities during ordinary inference, choose balanced layer vectors under the shared byte budget, and apply them to newly quantized cache blocks. Different agents may choose different layer vectors without a separate calibration dataset.

4.1 Compute Sensitivity from Recent Queries, Keys, and Values

We compute $s_{a,\ell,r}$, the sensitivity score used in Eq. (8), from recent queries, keys, and values of invocation a in layer ℓ. Instead of using the entire history, we collect a small recent observation

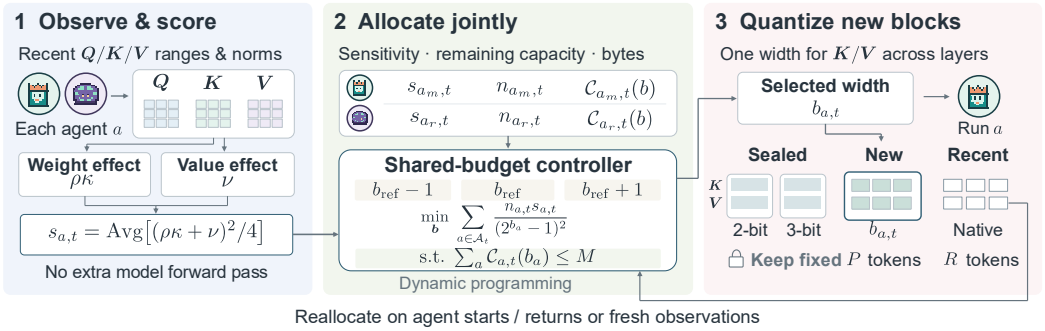


Figure 3: **Overview of CoMixKV. Observe and score:** recent $\boldsymbol{Q}/\boldsymbol{K}/\boldsymbol{V}$ observations provide online sensitivity scores by agent and layer. **Allocate:** the controller assigns paired lower- and higher-bit layers with the target packed-code mean for each invocation under the shared byte envelope. **Quantize new blocks:** selected widths apply only to newly quantized blocks; sealed blocks remain unchanged, and the most recent R tokens retain native precision. Allocations are updated when the resident set changes or fresh observations become available.

window. Recent queries provide a compact signal of the agent's current attention behavior, while restricting the computation to a short window keeps the online estimation inexpensive. Similar observation windows have been used in KV-cache compression to estimate token importance from recent attention patterns (Li et al., 2024b; Cai et al., 2025; Feng et al., 2025); here the score estimates sensitivity to quantization.

For one attention head, let $\boldsymbol{q} \in \mathbb{R}^d$ be an attention query vector and let $\boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{w\times d}$ contain w recent KV entries with d features each. The attention output over this window is

$$\boldsymbol{o} = \mathrm{Softmax}\left(\boldsymbol{K}\boldsymbol{q}/\sqrt{d}\right)^\top \boldsymbol{V}. \tag{10}$$

Softmax converts the query–key scores into weights that sum to one, and the resulting attention output is $\boldsymbol{o} \in \mathbb{R}^d$. Following KIVI (Liu et al., 2024c), we group keys across tokens within each feature and values across features within each token. A group's range is its maximum minus its minimum; a wider range allows larger rounding errors at the same bit width. We compute three scalars directly from $\boldsymbol{q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$:

$$\kappa = \frac{1}{\sqrt{d}}\sum_j |q_j| \max_{\mathcal{G}}\left(\max_{i\in\mathcal{G}}[\boldsymbol{K}]_{ij} - \min_{i\in\mathcal{G}}[\boldsymbol{K}]_{ij}\right), \nu = \max_i \sqrt{\sum_{\mathcal{G}} |\mathcal{G}|\left(\max_{j\in\mathcal{G}}[\boldsymbol{V}]_{ij} - \min_{j\in\mathcal{G}}[\boldsymbol{V}]_{ij}\right)^2}, \rho = \frac{1}{2}\sqrt{\sum_j \left(\max_i[\boldsymbol{V}]_{ij} - \min_i[\boldsymbol{V}]_{ij}\right)^2}. \tag{11}$$

Here, $[\boldsymbol{K}]_{ij}$ and $[\boldsymbol{V}]_{ij}$ denote matrix entries, with i indexing tokens and j indexing features, and q_j is the corresponding query entry. In κ, $\mathcal{G}$ ranges over token groups; in ν, it ranges over feature groups, with $|\mathcal{G}|$ counting the features in each group. The scalar κ combines query magnitudes and key ranges to capture how rounding can change the query–key scores. The scalar ν captures the possible effect of value rounding, while ρ measures how much the value vectors differ across tokens.

A b-bit uniform quantizer divides each group's range into $2^b - 1$ equal intervals. With exact group endpoints and nearest rounding, each entry's error is at most half an interval. Using κ, ν, ρ from Eq. (11), we combine the effects of key and value rounding to obtain

$$\|\widetilde{\boldsymbol{o}}_b - \boldsymbol{o}\|_2 \le \frac{\rho\kappa + \nu}{2(2^b - 1)}. \tag{12}$$

Here, $\widetilde{\boldsymbol{o}}_b \in \mathbb{R}^{1\times d}$ is the attention output computed from reconstructed b-bit keys and values, with the query and visible positions unchanged. The numerator depends on the observed queries, keys, and values, while the denominator shows how increasing precision reduces the bound.

Squaring this bound separates the coefficient $(\rho\kappa+\nu)^2/4$ from the bit-dependent factor $(2^b-1)^{-2}$. We average the coefficient to obtain the sensitivity score for invocation a at allocation round r:

$$s_{a,\ell,r} = \mathrm{Average}_{h,u}\left[\frac{(\rho\kappa+\nu)^2}{4}\right]. \tag{13}$$

Here, h and u index query heads and positions in layer ℓ. Within the average, κ uses the corresponding query and key head, while ρ and ν use the associated value head; their indices are omitted for brevity. Query heads sharing one KV head use the same keys and values. A larger $s_{a,\ell,r}$ indicates larger estimated attention error at the same bit width. Summing $s_{a,\ell,r}/(2^{b_{a,\ell}}-1)^2$ over layers and resident invocations gives Eq. (8). The bound itself applies only to the observed window; using it for future blocks is a planning surrogate. Appendix E.1 gives the derivation and assumptions.

4.2 Allocate the Shared Memory Budget

At allocation round r, the memory reserved for invocation a when its remaining quantizable blocks use layer vector $\boldsymbol{b}_a$ is

$$\mathcal{C}_{a,r}(\boldsymbol{b}_a) = I_{a,r} + \sum_{\ell\in\mathcal{L}} n_{a,\ell,r}\mathcal{B}_{\mathrm{blk}}(b_{a,\ell}) + U_{a,r} + O_a, \tag{14}$$

where $I_{a,r}$ is the storage of already quantized blocks and $n_{a,\ell,r}$ is remaining block capacity in layer ℓ. The per-layer block cost $\mathcal{B}_{\mathrm{blk}}(b)$ includes packed codes, scales, and offsets. The terms $U_{a,r}$ and O_a cover unquantized KV states and sensitivity observations, respectively. Appendix E.2 gives the byte calculation.

The audited mixed-width runs enforce a cumulative physical-byte envelope at every allocation round. Let

$$\mathcal{C}^{\mathrm{tar}}_{a,r} = \sum_{\ell\in\mathcal{L}} \left(\frac{z_{a,\ell,r}}{P} + n_{a,\ell,r}\right)\mathcal{B}_{\mathrm{blk}}(b_{\mathrm{avg}}) + U_{a,r} + O_a, \qquad M^{\mathrm{eff}}_r = \min\left(M, \sum_{a\in\mathcal{A}_r}\mathcal{C}^{\mathrm{tar}}_{a,r}\right). \tag{15}$$

The target entitlement charges sealed and prospective blocks at the average width, whereas $\mathcal{C}_{a,r}(\boldsymbol{b}_a)$ charges sealed blocks at their immutable widths. Pairing one lower-bit and one higher-bit layer within the same invocation keeps prospective payload bytes at the average target for equal-sized layer blocks. This avoids depending on byte savings from another agent that might later leave. The controller records whether each proposed reservation fits M^{eff}_r, and the final audit also checks actual sealed-code widths and physical storage.

For each invocation, we sort layers by their current scores. For a fixed number k of paired exchanges, the k least-sensitive layers use $b_{\mathrm{avg}}-1$, the k most-sensitive use $b_{\mathrm{avg}}+1$, and the remaining layers use b_{avg}. We evaluate all $1 \le k \le \lfloor (L-1)/2 \rfloor$ and select the minimum surrogate in Eq. (8). The first pair guarantees all three widths; more pairs are selected only when they reduce the surrogate. If two invocations would receive the same vector, a minimum-cost swap between two differently coded layers makes their vectors distinct without changing either invocation's width counts. Appendix E.3 gives the selection details.

4.3 Apply and Update the Bit-Width Assignment

The selected width $b_{a,\ell,r}$ applies to newly quantized $\boldsymbol{K}$ and $\boldsymbol{V}$ blocks in layer ℓ. New KV entries are first appended at native precision. After each forward pass, older unquantized entries outside the recent R-token window are quantized in complete blocks of P tokens using the current assignment. Incomplete blocks remain unquantized. Each quantized block retains its stored width and codes, so later assignments affect only newly quantized blocks.

The controller updates assignments when an invocation starts or finishes, or when fresh sensitivity observations become available. Observations are collected during prefill and approximately every J newly generated tokens. Before an invocation has a valid observation for a layer in its current request, that layer uses the median valid score among resident invocations (or 1 if none exists) as a temporary planning score; its first observation replaces this value. Completed invocations release their caches. If a changed request invalidates a cached suffix, we discard it and recompute the assignment before rebuilding that suffix. Between updates, the current assignment is reused. The audited reruns use no score smoothing or switch hysteresis. The runtime records physically sealed widths by agent and layer for post-run checks, including the exact mean packed-code width.

Algorithm 2 presents the end-to-end procedure, integrating cache initialization and reuse, sensitivity estimation, joint bit-width allocation, block quantization, and cache release.

Table 1: Accuracy (%) across four benchmarks and two model backbones at an average 3-bit KV-cache bit-width budget.

Model	Method	Knowledge Search	Agentic Reasoning	Mathematical Reasoning	Scientific Reasoning	Average
		Bamboogle	GAIA	AIME25	GPQA	
Qwen3.8-27B	Vanilla	87.20	58.25	96.67	77.78	79.98
	KIVI	90.40	58.25	90.00	81.82	80.12
	TurboQuant	88.80	58.25	93.33	81.82	80.55
	PM-KVQ	88.80	56.31	86.67	77.78	77.39
	CoMixKV	91.20	57.28	93.33	81.82	**80.91**
Gemma-4-31B-it	Vanilla	69.60	29.13	93.33	81.82	68.47
	KIVI	69.60	28.16	90.00	80.81	67.14
	TurboQuant	68.00	27.18	90.00	85.86	67.76
	PM-KVQ	61.60	32.04	90.00	84.85	67.12
	CoMixKV	70.40	28.16	90.00	84.85	**68.35**

5 Experiments

5.1 Experimental Setup

Baselines. We compare CoMixKV with **KIVI** (Liu et al., 2024c), **TurboQuant** (Zandieh et al., 2026), and **PM-KVQ** (Liu et al., 2026). All the methods are representative KV-cache quantization baselines. For fair comparison, we use the same model and decoding configuration for all methods and set the same target bit width for every quantization baseline.

Models. We evaluate three open-weight backbones with dense and hybrid-attention architectures: Qwen3-32B (Yang et al., 2025), Qwen3.8-27B (Qwen Team, 2026), and Gemma-4-31B-it (Team et al., 2026). This selection lets us examine CoMixKV across distinct model families and attention architectures.

Benchmarks. We evaluate four benchmarks spanning four types of reasoning tasks: (1) *knowledge-intensive search*, including Bamboogle (Press et al., 2023); (2) *agentic reasoning*, including GAIA (Mialon et al., 2024); (3) *mathematical reasoning*, including AIME25 (Zhang & Team, 2025); and (4) *scientific reasoning*, represented by GPQA (Rein et al., 2024). Together, these benchmarks cover knowledge retrieval, multi-step agent interaction, mathematical problem solving, and challenging scientific reasoning.

Implementation Details. All experiments are run on a server equipped with 8× NVIDIA H20 96GB GPUs, and our implementation is built on SGLang (Zheng et al., 2024). For a fair comparison, we evaluate all methods under average KV-cache bit-width budgets of 2, 3, and 4 bits. Each method follows its original bit-allocation strategy while matching the corresponding average bit-width budget. Unless otherwise specified, CoMixKV uses a token block size of $P = 128$, a quantization group size of $G = 32$, a native-precision recent window of $R = 128$ tokens, an observation window of $W = 32$ tokens, and a sensitivity refresh interval of $J = 128$ newly generated tokens.

5.2 Main Results

Comparison with quantization baselines. Table 1 reports the results across four benchmarks and two model backbones under an average 3-bit KV-cache bit-width budget. On Qwen3.8-27B, CoMixKV achieves the highest average accuracy among all quantization methods, reaching 80.91% and outperforming the strongest baseline, TurboQuant, by 0.36%. On Gemma-4-31B-it, CoMixKV achieves an average accuracy of 68.35%, outperforming the strongest quantization baseline, TurboQuant, by 0.59%. Furthermore, CoMixKV remains competitive with the unquantized Vanilla reference, demonstrating the effectiveness of coordinated mixed-precision KV-cache quantization across different model architectures.

Different Bit-width budgets. Figure 4(a) summarizes the average accuracy of Qwen3-32B under average KV-cache bit-width budgets of 2 and 4 bits, while Table 2 in Appendix A reports the com-

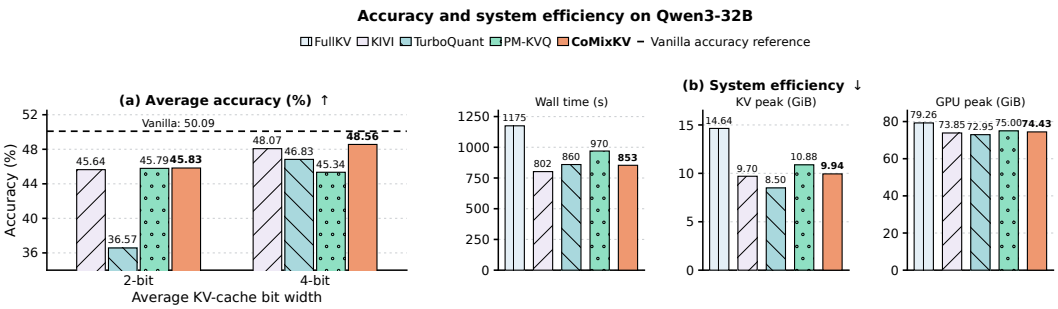


Figure 4: **Accuracy and system efficiency on Qwen3-32B.** (a) Average accuracy across four benchmarks under average 2-bit and 4-bit KV-cache bit-width budgets; the dashed line denotes the unquantized 16-bit Vanilla reference. (b) System efficiency on the controlled long-context agent workload, including batch wall time, physical KV-cache peak, and total GPU-memory peak. Lower is better for all three efficiency metrics.

plete benchmark-level results. CoMixKV achieves the highest average accuracy among all quantization methods at both bit-width budgets. At 2 bits, CoMixKV obtains 45.83% average accuracy, slightly outperforming the strongest baseline, PM-KVQ, by 0.04%. At 4 bits, CoMixKV reaches 48.56%, exceeding the strongest baseline, KIVI, by 0.49%. These results show that CoMixKV remains effective across different KV-cache precision budgets.

5.3 Memory and Latency Analysis

Controlled long-context agent workload. We evaluate system efficiency on Qwen3-32B using a controlled three-stage Manager–Analyst–Verifier workflow. Each call uses 32,768 input tokens and generates 2,048 output tokens, with five samples executed concurrently on one NVIDIA H20 96GB GPU. All methods use the same inputs and decoding configuration; quantized methods use the average 2-bit setting, while FullKV stores KV states in BF16.

Results. Figure 4(b) shows that CoMixKV reduces batch wall time from 1174.59 s to 853.44 s, corresponding to a 1.38× speedup over FullKV. It also reduces the KV-cache peak from 14.64 GiB to 9.94 GiB (32.08% reduction) and the total GPU-memory peak from 79.26 GiB to 74.43 GiB. These results indicate that CoMixKV substantially lowers KV-cache memory while improving end-to-end throughput under the same GPU-memory budget.

5.4 Ablation Studies

We further analyze the key design choices of CoMixKV in Appendix H.1 and Appendix H.2. For hyperparameter sensitivity, the default setting $(W, J) = (32, 128)$ ties for the best observed accuracy while avoiding a larger observation window or more frequent sensitivity updates, providing a favorable trade-off between accuracy and online overhead. For the allocation strategy, the online sensitivity-based policy consistently performs best among the evaluated variants, outperforming both uniform allocation and fixed sensitivity. These results demonstrate that dynamically updating layer sensitivity during execution and reallocating KV-cache precision accordingly is an important component of CoMixKV.

6 Conclusion

In this paper, we propose CoMixKV, a mixed-precision KV-cache quantization method for efficient LLM-based multi-agent systems. Our method assigns bit widths to agent–layer cache blocks under a shared memory budget. We derive a sensitivity score from a local attention-error bound and compute it using recent queries, keys, and values, without a calibration dataset or additional model passes. Paired lower- and higher-bit layer choices preserve each invocation's target mean packed-code width and cover all three candidates. The selected widths apply only to new blocks, while previously quantized blocks remain unchanged. Periodic sensitivity measurements refresh layer assignments as agents' states change.

GENERATIVE AI DISCLOSURE

Generative AI tools were used to assist with research brainstorming and providing feedback on methodology and experimental design, literature retrieval and summarization, drafting and editing parts of the manuscript, translation, code editing, and improving the clarity of technical descriptions and scientific figures. Generative AI tools were also used to assist with mathematical derivations and the development and verification of arguments supporting mathematical claims. All methodological decisions, implementations, experimental results, mathematical derivations, proofs, and scientific claims were independently reviewed and verified by the authors. The authors made all final research decisions and take full responsibility for the content of this work.

ETHICS STATEMENT

This work does not involve human subjects or private user data. We follow the ICLR Code of Ethics and do not identify any specific ethical concerns associated with the methods or experiments presented in this work.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of the experimental settings, models, datasets, baselines, and evaluation protocols in the Experiments section. The code will be made publicly available upon acceptance of the paper.

REFERENCES

Zhuohang Bian, Feiyang Wu, Zhuoran Li, Teng Ma, and Youwei Zhuo. Tokencake: A kv-cache-centric serving framework for llm-based multi-agent applications. *arXiv preprint arXiv:2510.18586*, 2025.

John Adrian Bondy, Uppaluri Siva Ramachandra Murty, et al. *Graph theory with applications*, volume 290. Macmillan London, 1976.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=ayi7qezU87.

Shuaihang Chen, Yuanxing Liu, Wei Han, Weinan Zhang, and Ting Liu. A survey on llm-based multi-agent system: Recent advances and new frontiers in application. *arXiv preprint arXiv:2412.17481*, 2024.

Krishna Teja Chitty-Venkata, Jie Ye, Siddhisanket Raskar, Anthony Kougkas, Xian Sun, Murali Emani, Venkatram Vishwanath, and Bogdan Nicolae. Pagedeviction: Structured block-wise kv cache pruning for efficient large language model inference. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 3207–3218, 2026.

Shichen Dong, Wen Cheng, Jiayu Qin, and Wei Wang. Qaq: Quality adaptive quantization for llm kv cache. *arXiv preprint arXiv:2403.04643*, 2024.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference. *Advances in Neural Information Processing Systems*, 38:113152–113188, 2025.

Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Erkang, Zhu, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, Peter Chang, Ricky Loynd, Robert West, Victor Dibia, Ahmed Awadallah, Ece Kamar, Rafah Hosn, and Saleema Amershi. Magentic-one: A generalist multi-agent system for solving complex tasks, 2024. URL https://arxiv.org/abs/2411.04468.

Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. Cost-efficient large language model serving for multi-turn conversations with cachedattention. In *2024 USENIX annual technical conference (USENIX ATC 24)*, pp. 111–126, 2024a.

Dawei Gao, Zitao Li, Xuchen Pan, Weirui Kuang, Zhijian Ma, Bingchen Qian, Fei Wei, Wenhao Zhang, Yuexiang Xie, Daoyuan Chen, et al. Agentscope: A flexible yet robust multi-agent platform. *arXiv preprint arXiv:2402.14034*, 2024b.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive kv cache compression for llms. In *International Conference on Learning Representations*, volume 2024, pp. 22975–22988, 2024.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. In Kate Larson (ed.), *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, pp. 8048–8057. International Joint Conferences on Artificial Intelligence Organization, 8 2024. doi: 10.24963/ijcai.2024/890. URL https://doi.org/10.24963/ijcai.2024/890. Survey Track.

Mohsen Hariri, Alan Luo, Weicong Chen, Tianyi Zhang, Qifan Wang, Xiaotian Han, and Vipin Chaudhary. Quantize what counts: More for keys, less for values. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 26393–26420, 2026.

Junda He, Christoph Treude, and David Lo. Llm-based multi-agent systems for software engineering: Literature review, vision, and the road ahead. *ACM Transactions on Software Engineering and Methodology*, 34(5):1–30, 2025.

Yefei He, Luoming Zhang, Weijia Wu, Jing Liu, Hong Zhou, and Bohan Zhuang. Zipcache: Accurate and efficient kv cache quantization with salient token identification. *Advances in Neural Information Processing Systems*, 37:68287–68307, 2024.

Pratik Honavar and Tejpratap GVSL. Qkvshare: Quantized kv-cache handoff for multi-agent on-device llms. *arXiv preprint arXiv:2605.03884*, 2026.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Steven Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, volume 2024, pp. 23247–23275, 2024.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun S Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems*, 37:1270–1303, 2024.

Long Hu, Nan Jia, Rui Wang, Jiahui Li, Qingyi Yang, Yixue Hao, Xianzhi Li, and Xiaofei Liao. TAKE: Task-aware chunked KV cache eviction for efficient long-context LLM prefill, 2026. URL https://openreview.net/forum?id=kMLfUshPwo.

Shuowei Jin, Xueshen Liu, Qingzhao Zhang, and Zhuoqing Mao. Compute or load KV cache? why not both? In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=WOyOtaO6lQ.

Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. Gear: An efficient kv cache compression recipe for near-lossless generative inference of llm. *arXiv preprint arXiv:2403.05527*, 2024.

Boris Kriuk and Logic Ng. Q-kvcomm: Efficient multi-agent communication via adaptive kv cache compression. In *2026 Second International Conference on Multi-Agent Systems for Collaborative Intelligence (ICMSCI)*, pp. 15–21. IEEE, 2026.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in neural information processing systems*, 36:51991–52008, 2023.

Haoyang Li, Yiming Li, Anxin Tian, Tianhao Tang, Zhanchao Xu, Xuejia Chen, Nicole Hu, Wei Dong, Qing Li, and Lei Chen. A survey on large language model acceleration based on kv cache management. *arXiv preprint arXiv:2412.19442*, 2024a.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024b.

Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. Minicache: Kv cache compression in depth dimension for large language models. *Advances in Neural Information Processing Systems*, 37:139997–140031, 2024a.

Tengxuan Liu, Shiyao Li, Jiayi Yang, Tianchen Zhao, Feng Zhou, Xiaohui Song, Guohao Dai, Shengen Yan, Huazhong Yang, and Yu Wang. Pm-kvq: Progressive mixed-precision kv cache quantization for long-cot llms. In *International Conference on Learning Representations*, volume 2026, pp. 107881–107900, 2026.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36:52342–52364, 2023.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic llm-powered agent network for task-oriented agent collaboration. In *First Conference on Language Modeling*, 2024b. URL https://openreview.net/forum?id=XII0Wp1XA9.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. In *International Conference on Machine Learning*, pp. 32332–32344. PMLR, 2024c.

Tula Masterman, Sandi Besen, Mason Sawtell, and Alex Chao. The landscape of emerging ai agent architectures for reasoning, planning, and tool calling: A survey. *arXiv preprint arXiv:2404.11584*, 2024.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *International Conference on Learning Representations*, volume 2024, pp. 9025–9049, 2024.

Zaifeng Pan, AJJKUMAR DAHYALAL PATEL, Yipeng Shen, Zhengding Hu, Yue Guan, Wan-Lu Li, Lianhui Qin, Yida Wang, and Yufei Ding. Kvflow: Efficient prefix caching for accelerating llm-based multi-agent workflows. *Advances in Neural Information Processing Systems*, 38: 126246–126265, 2025.

Ishan Patel and Ishan Joshi. Polykv: A shared asymmetrically-compressed kv cache pool for multi-agent llm inference. *arXiv preprint arXiv:2604.24971*, 2026.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 5687–5711, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.378. URL https://aclanthology.org/2023.findings-emnlp.378/.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 15174–15186, 2024.

Qwen Team. Qwen3.8-27B model card. Hugging Face, 2026. URL https://huggingface.co/Qwen/Qwen3.8-27B.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Hanzhang Shen, Haoran Wu, Yiren Zhao, and Robert Mullins. Triaxialkv: Toward extreme low-precision kv-cache quantization for agentic inference tasks. *arXiv preprint arXiv:2605.17170*, 2026.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference. In *International Conference on Machine Learning*, pp. 47901–47911. PMLR, 2024.

Gemma Team, Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, Cormac Brick, Victor Cărbune, Michelle Casbon, et al. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026.

Jiahao Wang, Jinbo Han, Xingda Wei, Sijie Shen, Dingyan Zhang, Chenguang Fang, Rong Chen, Wenyuan Yu, and Haibo Chen. {KVCache} cache in the wild: Characterizing and optimizing {KVCache} cache at a large cloud provider. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*, pp. 465–482, 2025.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=BAakY1hNKS.

Songhao Wu, Ang Lv, xiao feng, Yufei zhang, Xun Zhang, Guojun Yin, Wei Lin, and Rui Yan. Polarquant: Leveraging polar transformation for key cache quantization and decoding acceleration. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 50584–50607. Curran Associates, Inc., 2025. doi: 10.52202/085713-1688. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/48aa1c882f99f5cdd3c76f7c0ede44a7-Paper-Conference.pdf.

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

Yingxuan Yang, Qiuying Peng, Jun Wang, Ying Wen, and Weinan Zhang. Llm-based multi-agent systems: Techniques and business perspectives. *arXiv preprint arXiv:2411.14033*, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Hancheng Ye, Zhengqi Gao, Mingyuan Ma, Qinsi Wang, Yuzhe Fu, Ming-Yu Chung, Yueqian Lin, Zhijian Liu, Jianyi Zhang, Danyang Zhuo, et al. Kvcomm: Online cross-context kv-cache communication for efficient llm-based multi-agent systems. *Advances in Neural Information Processing Systems*, 38:17882–17928, 2025.

Lingfan Yu, Jinkun Lin, and Jinyang Li. Stateful large language model serving with pensieve. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 144–158, 2025.

Amir Zandieh, Majid Daliri, Majid Hadian, and Vahab Mirrokni. Turboquant: Online vector quantization with near-optimal distortion rate. In *International Conference on Learning Representations*, volume 2026, pp. 56418–56439, 2026.

Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Yu, and Tianlong Chen. Cut the crap: An economical communication pipeline for llm-based multi-agent systems. In *International Conference on Learning Representations*, volume 2025, pp. 75389–75428, 2025a.

Hailin Zhang, Xiaodong Ji, Yilin Chen, Fangcheng Fu, Xupeng Miao, Xiaonan Nie, Weipeng Chen, and Bin Cui. Pqcache: Product quantization-based kvcache for long context llm inference. *Proceedings of the ACM on Management of Data*, 3(3):1–30, 2025b.

Ji Zhang, Yiwei Li, Shaoxiong Feng, Peiwen Yuan, Xinglin Wang, Yueqi Zhang, Jiayi Shi, Chuyi Tan, Boyuan Pan, Yao Hu, and Kan Li. PatternKV: Flattening KV representation expands quantization headroom. In *Forty-third International Conference on Machine Learning*, 2026a. URL https://openreview.net/forum?id=ozV2cpEgnZ.

Jiebin Zhang, Dawei Zhu, Yifan Song, Wenhao Wu, Chuqiao Kuang, Xiaoguang Li, Lifeng Shang, Qun Liu, and Sujian Li. More tokens, lower precision: Towards the optimal token-precision trade-off in kv cache compression. In *EMNLP (Findings)*, pp. 8092–8105, 2025c.

Tao Zhang, Ziqian Zeng, Hao Peng, Huiping Zhuang, and Cen Chen. Mixkvq: Query-aware mixed-precision kv cache quantization for long-context reasoning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7189–7204, 2026b.

Xing Zhang, Yanwei Cui, Guanghui Wang, Wei Qiu, Ziyuan Li, Fangwei Han, Yajing Huang, Hengzhi Qiu, Bing Zhu, and Peiyang He. Verified multi-agent orchestration: A plan-execute-verify-replan framework for complex query resolution. *arXiv preprint arXiv:2603.11445*, 2026c.

Yabo Zhang, Yihan Zeng, Qingyun Li, Zhen Hu, Kavin Han, and Wangmeng Zuo. Tool-r1: Sample-efficient reinforcement learning for agentic tool use. *arXiv preprint arXiv:2509.12867*, 2025d.

Yifan Zhang and Math-AI Team. American invitational mathematics examination (aime) 2025, 2025. URL https://huggingface.co/datasets/math-ai/aime25.

Yuxin Zhang, Yuxuan Du, Gen Luo, Yunshan Zhong, Zhenyu Zhang, Shiwei Liu, and Rongrong Ji. Cam: Cache merging for memory-efficient llms inference. In *International Conference on Machine Learning*, pp. 58840–58850. PMLR, 2024.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Gptswarm: Language agents as optimizable graphs. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62743–62767. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/zhuge24a.html.

Fei Zuo, Zikang Zhou, Hao Cong, Xiaoyan Xi, and Ho Fai Leung. Ratequant: Optimal mixed-precision kv cache quantization via rate-distortion theory. *arXiv preprint arXiv:2605.06675*, 2026.

APPENDIX

A Additional Qwen3-32B Results at 2 and 4 Bits **15**

B Notation and Runtime Workflow **15**

C Agent Roles and Prompt Construction **17**

D Task and Recovery Prompt Templates **21**

E Derivations and Algorithm Details **25**

E.1 From Attention to the Estimated Quantization Error 25

E.2 Memory Reservation Calculation 28

E.3 Balanced Agent–Layer Allocation 29

F End-to-end Algorithm **29**

G Details of Baselines, Models and Benchmarks **29**

G.1 Details of Baselines 29

G.2 Details of Models 30

G.3 Details of Benchmarks 31

H Additional Experimental Analysis **31**

H.1 Hyperparameter Sensitivity 31

H.2 Online Allocation Ablation 32

A Additional Qwen3-32B Results at 2 and 4 Bits

Table 2 reports supplementary Qwen3-32B accuracy results under average KV-cache bit-width budgets of 2 and 4 bits on the four benchmarks. The unquantized Vanilla reference is repeated for comparison.

Table 2: Accuracy (%) of Qwen3-32B at average KV-cache bit-width budgets of 2 and 4 bits, with the unquantized 16-bit Vanilla reference.

Bits	Method	Knowledge Search	Agentic Reasoning	Mathematical Reasoning	Scientific Reasoning	Average
		Bamboogle	GAIA	AIME25	GPQA	
16	Vanilla	81.60	28.16	30.00	60.61	50.09
2	KIVI	71.20	26.21	36.67	48.48	45.64
	TurboQuant	53.60	21.36	23.33	47.98	36.57
	PM-KVQ	71.20	24.27	36.67	51.01	45.79
	CoMixKV	69.60	25.24	40.00	48.48	**45.83**
4	KIVI	75.20	32.04	30.00	55.05	48.07
	TurboQuant	73.60	28.16	30.00	55.56	46.83
	PM-KVQ	76.00	28.16	26.67	50.51	45.34
	CoMixKV	77.60	31.07	30.00	55.56	**48.56**

B Notation and Runtime Workflow

This section expands the execution graph in Section 3.1 and connects its agent states to the invocation-level caches in Section 3.2. We retain i for an agent's role and a for one invocation of that role; repeated invocations receive different identifiers even when they use the same role. The index t denotes an agent step within an invocation, whereas r denotes a cache-allocation round. Table 3 summarizes the notation. Bold lowercase symbols in these three sections represent serialized text sequences, and $\|$ concatenates them in order.

Local execution. All roles use the `ContextualToolCallingAgent` wrapper around the smolagents `ToolCallingAgent`. They share the model $\mathcal{M}$ from Eq. (2) but receive different role

Table 3: Agent-side notation and its connection to the main-text formulation.

Notation	Definition
$\boldsymbol{q}$	Original task query in Eq. (1); not the numerical attention query in Eq. (10).
$i \in \{\mathrm{m}, \mathrm{r}, \mathrm{p}, \mathrm{v}\}$	Fixed role labels for manager, researcher, analyst, and verifier. The upright label r denotes the researcher, not allocation round r.
a	One invocation of role i, owning the history $\boldsymbol{h}_a$ and cache $\mathcal{KV}_a$ defined in the main text.
$v_a^{(t)}$	Invocation-indexed refinement of $v_i^{(t)}$, distinguishing separate runs of the same role.
$\boldsymbol{h}_a^{(t)}$	The local history $\boldsymbol{h}_a$ before step t of invocation a.
$\boldsymbol{y}_a^{(t)}$	Serialized model output at that step, including its action, refining $\boldsymbol{y}_i^{(t)}$ in Eq. (3).
$\boldsymbol{x}_a^{\mathrm{obs},(t)}$	Text returned by the tool or managed-worker call issued at that step; not an attention-output vector.
$\mathcal{A}_r$	Invocations whose caches are resident at allocation round r, including paused invocations; not the configured set of roles.
$\mathcal{E}_{\boldsymbol{q}}^{\mathrm{T}}, \mathcal{E}_{\boldsymbol{q}}^{\mathrm{S}}$	Temporal and spatial edge sets from Eq. (1).
$\widehat{\boldsymbol{y}}, \boldsymbol{z}$	The manager's candidate answer and the final prediction accepted by the runtime.
$\boldsymbol{x}^{\mathrm{rep}}, \boldsymbol{x}^{\mathrm{rec}}$	Serialized worker evidence and additional recovery evidence, respectively.

instructions and action schemas. The history $\boldsymbol{h}_a^{(t)}$ represents the local interaction content recorded by `State` in the graph formulation. It begins with the invocation's instructions and input, including any explicitly supplied messages. For a nonterminal step, the model produces $\boldsymbol{y}_a^{(t)}$, the runtime executes its tool or managed-worker call, and the returned text is $\boldsymbol{x}_a^{\mathrm{obs},(t)}$. The history is updated as

$$\boldsymbol{h}_a^{(t+1)} = \boldsymbol{h}_a^{(t)} \parallel \boldsymbol{y}_a^{(t)} \parallel \boldsymbol{x}_a^{\mathrm{obs},(t)}. \tag{16}$$

This update preserves both the generated output and the returned observation. A tool result extends the invocation's local history; a worker report also carries spatial information from another invocation. The ordered history updates provide temporal information for later steps. A `final_answer` action returns the invocation's result and terminates the run, so it does not require another tool observation. Textual observations here are distinct from the numerical $\boldsymbol{Q}/\boldsymbol{K}/\boldsymbol{V}$ observations used to compute $s_{a,\ell,r}$.

Delegation and communication. The manager has no direct Search, Visit, or Python tool. For open-Web tasks, the researcher and analyst are exposed as managed-agent actions; for closed-book tasks, only the analyst is exposed. A manager-to-worker call contains a natural-language subtask. The manager pauses while the selected worker executes its local ReAct loop and retains its own cache during this period. The worker's textual report becomes the observation of that delegation action, after which the manager resumes. Workers do not call one another. The returned object is serialized text—a final report and, for most backbones, a textual run summary—rather than hidden states or KV tensors. The worker's cache is released when its invocation finishes. The verifier is not callable by the manager: the runtime invokes it after the manager has returned its candidate answer and released its cache.

Agent-level runtime. Algorithm 1 describes the role-level workflow, while Algorithm 2 manages the caches of its invocations. The workflow includes at least one worker invocation and one verifier invocation. If verification requests more evidence or produces no usable answer, the runtime can launch one targeted recovery invocation, using the researcher for open-Web tasks or the analyst for closed-book tasks. The recovery evidence is then supplied with the manager's candidate to a fresh verifier invocation; this branch does not resume the completed manager. Model-specific parsing and answer-format repairs are omitted.

In the algorithm, `TaskClass` specifies the task-dependent configuration, `Team` stores the configured roles and permissions, and `Worker` identifies the selected recovery role. These are configuration records, not numerical quantities or the resident set $\mathcal{A}_r$. The notation $\boldsymbol{h}_{\mathrm{m}}$ is shorthand for the completed manager transcript, and $\boldsymbol{y}_{\mathrm{v}}$ denotes the latest verifier response, which need not yet be a final answer. All worker-result routines return serialized evidence, so evidence is combined by concatenation rather than set union. The configured model and permissions are used by every run; each new invocation has a separate identifier a.

Algorithm 1 Agent-level execution used by the multi-agent runtime.

Require: Task query $\boldsymbol{q}$, `TaskClass`, model service $\mathcal{M}$
Ensure: Final prediction $\boldsymbol{z}$ in the requested format, or an explicit execution error

```
Team ← BUILDTEAM(TaskClass, M)
(ŷ, h_m) ← RUNMANAGER(q, Team)
x^rep ← WORKERREPORTS(h_m)
if h_m contains no managed-worker call then
    x^rep ← RUNANALYSTREVIEW(q, ŷ)
else if ŷ explicitly reports missing evidence then
    Worker ← SELECTRECOVERYWORKER(TaskClass)
    x^rep ← x^rep ‖ RUNWORKER(Worker, q, ŷ)
end if
y_v ← RUNVERIFIER(q, ŷ, x^rep)
if y_v requests more evidence or contains no usable answer then
    Worker ← SELECTRECOVERYWORKER(TaskClass)
    x^rec ← RUNTARGETEDRECOVERY(Worker, q, ŷ, x^rep, y_v)
    x^rep ← x^rep ‖ x^rec
    y_v ← RUNVERIFIER(q, ŷ, x^rep)
end if
if y_v still contains no usable answer then
    y_v ← FINALVERIFIERADJUDICATION(q, ŷ, x^rep, y_v)
end if
Accept the final answer from y_v as z, or raise an execution error
return z
```

From communication paths to cache branches. The implementation attaches a role-specific owner label of the form `role-r[rollout]-q[query-hash]` to each model request. This literal label identifies a role and rollout; it is not the allocation-round index r or a replacement for the invocation identifier a. Different roles do not use one another's session identity. Entering a managed-agent call switches the active context from the manager to the worker; returning its report restores the manager context. Recovery calls use `reset=True`, so they start fresh local histories even when they reuse a role. Such a reset begins a new invocation rather than continuing the earlier invocation's private cache. In the full-history mode described here, each turn supplies the invocation's current transcript, and SGLang can reuse an unchanged prefix through its Radix cache. The optional role-owned streaming-session path is not used for these runs.

Communication across a spatial edge is serialized into task or report text. A worker report is tokenized again as part of the receiver's prompt; the worker's KV tensors are not copied to the receiver. The serving backend may also reuse identical physical prefixes across requests, but different role instructions, subtasks, reports, or tool observations produce separate suffixes. This backend-level reuse does not merge the invocation identities or transfer an agent's private reasoning state. The set $\mathcal{A}_r$ and Eq. (14) describe invocation-level residency and reservations, not a count of unique shared physical allocations. An invocation leaves the resident set when its private cache is released, while its returned text may remain in another invocation's history.

Figure 5 illustrates how local action–observation histories, textual handoffs, and fresh recovery invocations give rise to separate cache branches.

C Agent Roles and Prompt Construction

The task class determines which workers are exposed to the manager and which tools each role can call. Open-Web tasks use the manager, researcher, analyst, and verifier. Mathematical and closed-book tasks use the manager, analyst, and verifier; the researcher and all Web tools are removed. Table 4 summarizes the actual action sets. The prompt clauses below use the portable JSON-action form described at the end of this section.

Figure 6 visualizes the task-conditioned role and tool configuration in Table 4, distinguishing managed-worker calls from runtime-only verification.

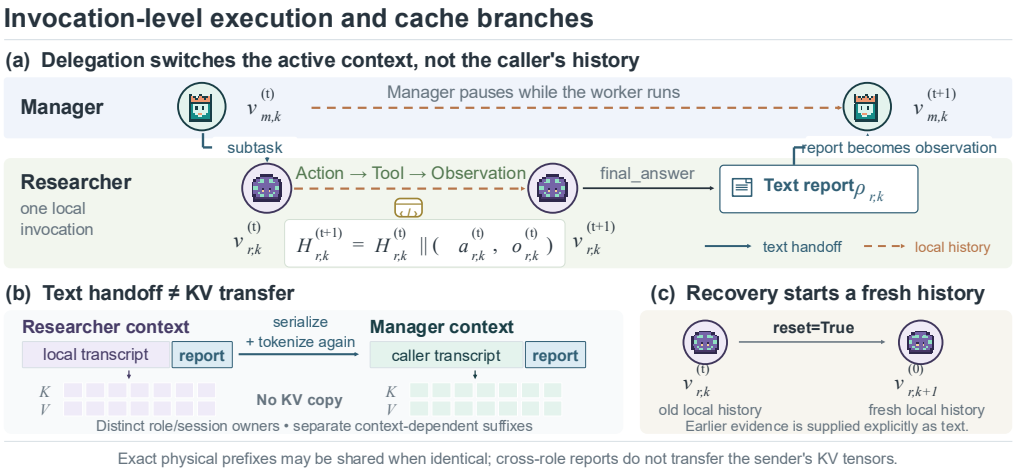


Figure 5: Invocation-level execution and cache branches. (a) The manager pauses during a managed-worker call while the worker extends its own local action–observation history; the returned textual report becomes the observation of the manager's delegation action. (b) Reports are serialized and tokenized as part of the receiver's prompt rather than transferred as KV tensors. Distinct context-dependent suffixes therefore form separate cache branches, although identical physical prefixes may be shared by the serving backend. (c) Recovery starts a fresh local history with `reset=True`; any earlier evidence is supplied explicitly as text.

Table 4: Role-specific actions in the multi-agent implementation. Every role also has `final_answer`. The verifier's `VerificationSearchTool` is exposed under the name `search`, but it returns a fixed message and never accesses the network.

Role	Open-Web tasks	Closed-book tasks
Manager	Calls researcher or analyst	Calls analyst only
Researcher	Search, Visit, Python	Not active
Analyst	Python, Search, Visit	Python only
Verifier	Python and guarded Search	Python only

Prompt assembly. For a fixed task class, the system prompt for role i concatenates four text components:

$$\boldsymbol{p}_i^{\mathrm{sys}} = \boldsymbol{p}_{\mathrm{ReAct}} \,\|\, \boldsymbol{p}_i^{\mathrm{schema}} \,\|\, \boldsymbol{p}_i^{\mathrm{role}} \,\|\, \boldsymbol{p}_i^{\mathrm{profile}}. \tag{17}$$

Here, $\boldsymbol{p}_{\mathrm{ReAct}}$ explains the shared action–observation loop; $\boldsymbol{p}_i^{\mathrm{schema}}$ lists the permitted tools and managed-worker actions; $\boldsymbol{p}_i^{\mathrm{role}}$ describes the responsibility specified by $\texttt{Role}_i$; and $\boldsymbol{p}_i^{\mathrm{profile}}$ gives the task-specific instructions. The schema follows the available $\texttt{Tools}_i$ and the task-dependent worker permissions in Table 4. These symbols denote text components, not additional model functions; $\mathcal{P}$ retains its main-text meaning as the conditional output distribution. The task query or delegated subtask is supplied separately as the run input, and each invocation maintains its own $\boldsymbol{h}_a$. Roles therefore share $\mathcal{M}$ but differ in instructions, permissions, and local histories.

Figure 7 illustrates the composition in Eq. (17) and the separation between system instructions, run input, and local history.

Manager. The manager coordinates the team and produces a candidate answer. In multi-agent mode it has no direct environment tools. For open-Web tasks it may delegate to the researcher or analyst; for closed-book tasks it may delegate only to the analyst. The manager is instructed to use at least one worker, to combine returned reports, and to follow the requested answer format. Within the manager loop, the analyst can be called at most once and the researcher at most twice; a later runtime recovery is a separate invocation.

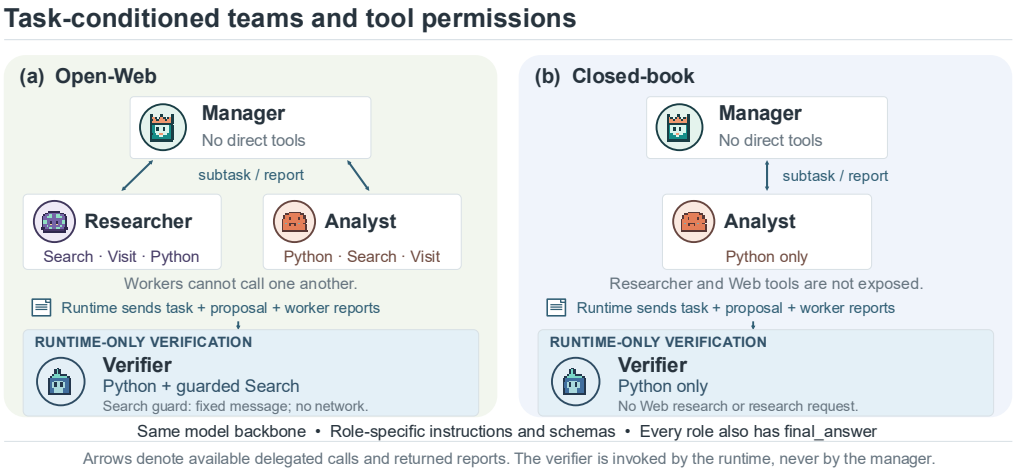


Figure 6: Task-conditioned teams and tool permissions. Open-Web tasks expose both researcher and analyst workers to the manager, whereas closed-book tasks expose only the analyst and disable Web tools. The verifier is a separate runtime stage, not a managed worker callable by the manager. Its guarded `search` action on open-Web tasks returns a fixed message without network access. Arrows show available task/report channels rather than a mandatory execution sequence, and every role also has `final_answer`.

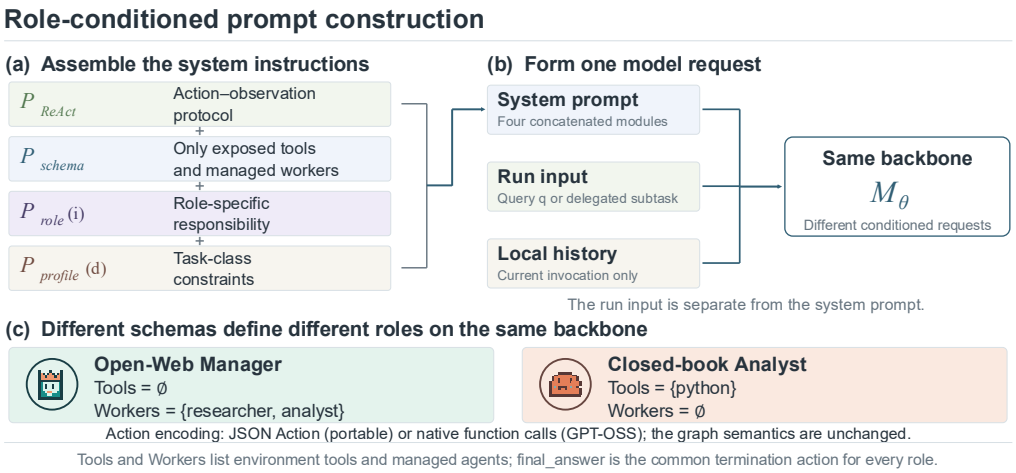


Figure 7: Role-conditioned prompt construction. The system prompt combines the shared ReAct protocol, the role's exposed action schemas, its responsibility, and the task-class profile. The query or delegated subtask is supplied separately as the run input, and each invocation retains its own local history. Roles use the same model backbone but differ in available actions, instructions, and visible histories.

Manager Role Clause for Open-Web Tasks

```
Coordinate the team to solve the user task; do not solve it
    entirely by yourself. You must delegate each task to at least
    one appropriate specialist. Use researcher for web evidence and
    source-grounded facts, and analyst for calculations, parsing,
    closed-book reasoning, or consistency checks.

After gathering the relevant worker result(s), synthesize one
    proposed exact answer. The runtime will send that proposal to
```

```
    an independent verifier before it becomes the prediction. Call
    the analyst at most once and synthesize returned evidence
    instead of reopening the same reasoning loop. You may call the
    researcher a second time only when the first attempt hits an
    access block, lacks the requested exact fact, or exposes a
    source conflict.

Follow explicit output-format instructions in the original task
    literally. Keep reasoning concise and emit exactly one valid
    JSON Action per step before the token limit.
```

Researcher. The researcher receives a focused subtask, gathers evidence through search or page visits, and returns a textual report. Python is available for lightweight processing. Its report, not its private local history or KV states, is inserted into the manager's history.

Researcher Role Clause

```
Focus on evidence, URLs, dates, and exact factual claims. Do not
    invent sources. Prefer the primary source named in the task,
    including its archived revision, PDF, API, dataset, or
    repository. Never use benchmark pages, answer dumps, or model
    traces that merely quote a benchmark question or claimed answer
    as evidence; solve the task from the underlying source.

Keep reasoning concise and emit one valid JSON Action before the
    token limit; escape quotes inside JSON strings. Use one focused
    query per search call because searches share a global rate
    limit. You only own search, visit, python, and final_answer;
    never call researcher, analyst, or verifier. Do not repeat
    equivalent searches or revisit a blocked URL. By step 8,
    synthesize the best-supported answer from the evidence already
    collected unless one different primary-source attempt is likely
    to resolve the task. Stop searching once the evidence supports
    the answer and finish with final_answer.
```

Analyst. The analyst handles calculations, structured extraction, and consistency checks. On open-Web tasks it owns Python, Search, and Visit; on mathematical and closed-book tasks only Python is exposed. It cannot call another agent.

Analyst Role Clause

```
Prioritize calculations, parsing, and consistency checks. Show the
    relevant result and assumptions. When using Python, compute
    from the task inputs and inspect stdout, stderr, and
    returncode. Do not use Python merely to print an assumed
    candidate answer. Repair syntax or runtime errors before
    finishing.

Track units through every calculation. If the task asks for
    thousands, millions, percentages, or another scaled unit,
    convert the raw result before proposing the final value. Apply
    the requested rounding and separators exactly. Use only the
    tools exposed in your schema, never call another agent, and
    finish with final_answer.
```

Verifier. The verifier is a separate runtime stage, not one of the manager's managed agents. It receives the original task, manager proposal, and worker evidence. For open-Web tasks it cannot perform new Web research: its `search` action is a non-network guard. It may return a checked answer or identify a focused evidence gap with `NEEDS_RESEARCH`; the runtime, rather than the verifier itself, launches any recovery worker. For closed-book tasks it independently reasons over

the problem, may use Python, and does not request Web research. A new verifier invocation can receive additional evidence without reusing the preceding verifier's private cache.

Verifier Role Clause for Open-Web Tasks

```
Independently check the proposed answer against the supplied worker
    evidence and task constraints. Reject evidence that repeats a
    benchmark question or another agent's output. Require the
    primary source named by the task when it is accessible. Treat a
    bare search snippet, unsupported claim, or single source for a
    multi-hop or counting task as insufficient evidence. Request
    the missing hop, underlying rows, historical revision, or
    corroborating source.

Check arithmetic, physical units, requested scale, rounding, date
    and time formats, ordering, and separators. Do not invent a
    correction or start new research. If the evidence supports an
    answer, return only the corrected exact answer using
    final_answer. If a decisive fact is missing or sources
    conflict, return NEEDS_RESEARCH: <one focused evidence gap or
    query> using final_answer. Add no other explanation.
```

Closed-Book Role Clauses

```
Manager: Coordinate the solution with the analyst. Delegate every
    task to the analyst for independent reasoning, calculation, or
    Python verification. Web search, page visits, and the
    researcher are unavailable. Synthesize one exact candidate
    answer for the verifier.

Analyst: Reason independently and use Python when useful. Check
    calculations, units, signs, option mapping, and output format.
    Return the exact requested answer and concise supporting
    reasoning.

Verifier: Independently solve and check the manager proposal
    against the original task and analyst report. Use Python when
    useful, never request Web research, and return only the
    best-supported exact answer.
```

Action format. For the portable smolagents protocol, an agent emits one JSON Action naming either an owned tool, a managed worker, or `final_answer`. A worker call creates a spatial edge, and a tool result extends the caller's temporal history.

D Task and Recovery Prompt Templates

This section records the task-dependent inputs and textual handoffs used by Algorithm 1. Table 5 lists the task profiles for the four benchmarks named in Section 5. Generic examples from the underlying smolagents prompt are omitted.

Table 5: Task profiles for the benchmark groups reported in the paper.

Benchmarks	Agent organization and task instruction
Bamboogle, GAIA	Use the open-Web team. Delegate source discovery to the researcher and calculations or structured extraction to the analyst.
AIME25	Use the closed-book team. Delegate derivation or Python verification to the analyst, disable Web access, and return only the requested exact numeric or symbolic expression.
GPQA	Use the closed-book team. Delegate scientific reasoning to the analyst, disable Web access, and return only the requested option, term, or short answer.

Messages passed between roles. Table 6 lists the textual payloads and their delivery routes. Runtime-mediated routes describe how messages are delivered, not additional agent vertices in $\mathcal{G}_q$. The payloads become inputs to the receiving invocation, consistent with $\boldsymbol{x}_i^{\mathrm{S},(t)}$ in Eq. (3). KV tensors are not transferred; a sender's private cache is released when its invocation ends.

Table 6: Textual payloads passed along the agent graph.

Message route	Payload visible to the receiver
Manager → worker	A delegated subtask and optional additional arguments.
Worker → manager	The worker's final textual report, optionally followed by a textual run summary.
Runtime → verifier	Original task, manager proposal, and extracted worker report(s).
Verifier → recovery worker (via runtime)	One focused evidence gap, together with the original task and earlier candidate/evidence.
Recovery worker → verifier (via runtime)	Recovery summary and the relevant Search, Visit, or Python observations.

Figure 8 summarizes the bounded verification and recovery workflow in Algorithm 1, providing an overview of when the following prompt templates are used.

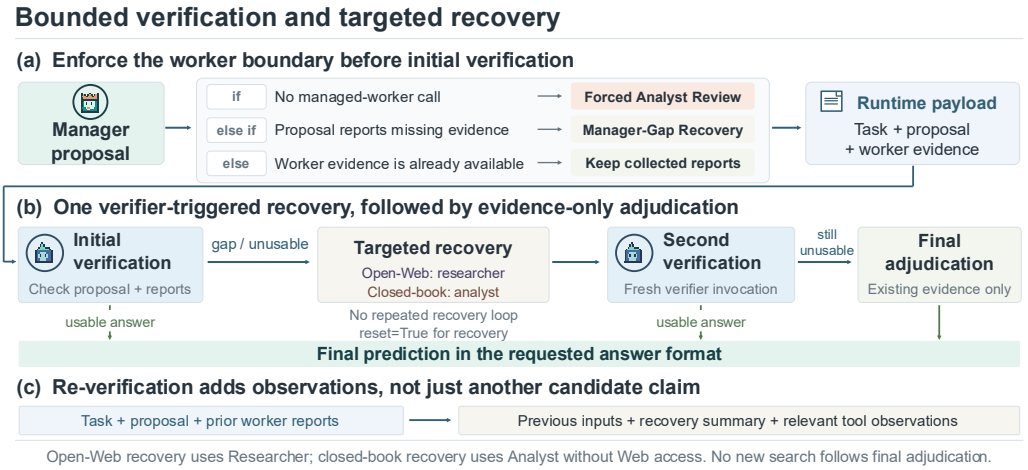


Figure 8: Bounded verification and targeted recovery. (a) Before initial verification, the runtime performs a forced analyst review if no worker was called; otherwise, an explicitly uncertain manager proposal can trigger manager-gap recovery. (b) An evidence request or unusable verification output triggers at most one targeted recovery in this branch, using a researcher for open-Web tasks or an analyst without Web access for closed-book tasks. A new verifier invocation checks the augmented evidence, with a final evidence-only adjudication if necessary and no further search branch. (c) Re-verification receives relevant tool observations together with the recovery summary, rather than treating repeated candidate claims as evidence.

Managed-worker call. The manager sends a worker a `task` and optional `additional_args` through the managed-agent schema. The library adds the worker's name and asks it to return a report for the manager. In simplified form, the handoff is:

Managed-Agent Handoff

```
You are the [RESEARCHER or ANALYST].

Task delegated by the manager:
[SUBTASK]
```

```
Use only the tools in your schema. Return a textual report that the
    manager can use.
```

Enforcing the worker boundary. If the manager returns a candidate without calling any worker, the runtime starts a fresh analyst invocation before verification. Its returned review forms $\boldsymbol{x}^{\mathrm{rep}}$ and is supplied to the verifier together with $\widehat{\boldsymbol{y}}$; the completed manager is not resumed:

Forced Analyst Review after Manager Bypass

```
Review the proposed answer for correctness, calculations,
    formatting, and compliance with the original task. Return a
    corrected exact answer if needed.

Original task:
[QUESTION]

Manager proposal:
[MANAGER PROPOSAL]
```

Recovery from an uncertain manager proposal. If the manager has already used a worker but its candidate explicitly reports missing evidence, the runtime invokes one worker again before verification: the researcher for open-Web tasks or the analyst for closed-book tasks. The returned text is appended to $\boldsymbol{x}^{\mathrm{rep}}$ and passed to the verifier with the existing candidate.

Manager-Gap Recovery Input

```
Revisit the uncertain manager proposal once.

For an open-Web task, use a different focused query or an
    alternative accessible source and return the exact answer with
    the shortest supporting evidence.

For a closed-book task, perform an independent reasoning and Python
    verification pass without Web access and return the exact
    best-supported answer.

Original task:
[QUESTION]

Uncertain manager proposal:
[MANAGER PROPOSAL]
```

Initial verification. The runtime starts verification after the manager and any required review have finished. The verifier receives the original task $\boldsymbol{q}$, candidate $\widehat{\boldsymbol{y}}$, and collected evidence $\boldsymbol{x}^{\mathrm{rep}}$, but no other invocation's KV tensors. Its response is denoted by $\boldsymbol{y}_{\mathrm{v}}$ until the runtime accepts a final prediction $\boldsymbol{z}$.

Open-Web Initial Verification Input

```
Independently verify the proposed answer against the original task
    and worker evidence. Return only the corrected exact answer
    when the evidence directly supports it. If one decisive fact is
    missing or conflicting, return NEEDS_RESEARCH: <one focused
    evidence gap or query>. Do not guess or include any other
    explanation.

Original task:
[QUESTION]
```

```
Manager proposal:
[MANAGER PROPOSAL]

Worker review:
[WORKER EVIDENCE]
```

Closed-Book Initial Verification Input

```
Independently solve and verify this closed-book problem against the
    Manager proposal and Analyst review. Use Python when useful,
    choose the best-supported option or value, and return only the
    exact answer requested. Do not search, request research,
    abstain, or include an explanation.

Original task:
[QUESTION]

Manager proposal:
[MANAGER PROPOSAL]

Analyst review:
[WORKER EVIDENCE]
```

Targeted recovery. For open-Web tasks, an evidence request or unusable verifier response can trigger a fresh researcher invocation. For closed-book tasks, an unusable response triggers a fresh analyst invocation without Web access. The runtime supplies the original task, candidate, earlier evidence, and verifier response to the selected worker. The returned recovery summary and relevant tool observations form $\boldsymbol{x}^{\mathrm{rec}}$.

Targeted Recovery Input

```
Resolve the verifier's specific evidence gap.

For an open-Web task, use a different focused query or source when
    earlier evidence is blocked or inconclusive, and inspect the
    named primary page, PDF, historical revision, dataset, or
    repository when possible.

For a closed-book task, perform an independent derivation and
    Python check without Web access.

Return the exact answer plus the shortest decisive quotation or
    calculation. Do not guess or abstain.

Original task:
[QUESTION]

Manager proposal:
[MANAGER PROPOSAL]

Earlier worker review:
[WORKER EVIDENCE]

Verifier evidence gap:
[VERIFIER OUTPUT]
```

Verification after recovery. The runtime appends $\boldsymbol{x}^{\mathrm{rec}}$ to the previously collected evidence and supplies the result, together with $\boldsymbol{q}$ and $\widehat{\boldsymbol{y}}$, to a fresh verifier invocation. The recovery report is not sent back to the completed manager. Relevant tool observations accompany the recovery summary so that the verifier can check its claims against the supplied evidence. The following template shows

the open-Web case. Closed-book verification retains the no-Web instruction given above and uses the additional analyst evidence.

Open-Web Verification after Recovery

```
Re-check the original task after targeted evidence recovery. Return
    only the exact final answer when supported. Treat the manager
    proposal and recovery summary as candidate claims. Validate
    them against the actual Search, Visit, or Python observations.
    If the decisive fact remains unavailable, return
    NEEDS_RESEARCH: <remaining evidence gap>.

Original task:
[QUESTION]

Manager proposal:
[MANAGER PROPOSAL]

Earlier worker evidence:
[WORKER EVIDENCE]

Targeted recovery evidence:
[RECOVERY EVIDENCE]
```

If the second verification is still unusable, the runtime invokes the verifier role once more for evidence-only adjudication over the collected messages; this does not start another search branch. The accepted answer becomes $\boldsymbol{z}$; an unresolved execution failure is not treated as a valid prediction. Bounded parser, terminal-error, and answer-format retries are implementation safeguards rather than additional agent roles, so their prompt text is not reproduced here.

E Derivations and Algorithm Details

This appendix gives the assumptions behind the attention bound, proves the allocation statements, and specifies the storage contract. No calibration dataset is used. The numerical observations come from ordinary forwards in the current invocation. The theoretical bound concerns ideal-grid quantization of an observed attention window; using its FP32 coefficient to choose precision for future blocks is a planning approximation.

E.1 From Attention to the Estimated Quantization Error

This subsection connects the attention computation in Eq. (10) to the allocation objective in Eq. (8). We first bound the effect of rounding keys and values within an observed window, then square and average this bound within each layer to obtain $s_{a,\ell,r}$. Finally, we explain the assumptions used to apply the local score to prospective blocks.

1. Local attention and assumptions. Fix invocation a and the observation used to compute its score at allocation round r. For one query and attention head, use $\boldsymbol{q} \in \mathbb{R}^d$ and $\boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{w \times d}$ as in the main text. Let $\widetilde{\boldsymbol{K}}_b$ and $\widetilde{\boldsymbol{V}}_b$ be the reconstructed b-bit matrices, where $b \geq 1$ is an integer. The query and visible positions remain unchanged. Define the query–key score vector $\boldsymbol{z} = \boldsymbol{K}\boldsymbol{q}/\sqrt{d}$ and attention weights $\boldsymbol{p} = \mathrm{Softmax}(\boldsymbol{z})$. Their quantized counterparts are $\widetilde{\boldsymbol{z}}_b = \widetilde{\boldsymbol{K}}_b\boldsymbol{q}/\sqrt{d}$ and $\widetilde{\boldsymbol{p}}_b = \mathrm{Softmax}(\widetilde{\boldsymbol{z}}_b)$. Thus,

$$\boldsymbol{o} = \boldsymbol{p}^\top \boldsymbol{V}, \qquad \widetilde{\boldsymbol{o}}_b = \widetilde{\boldsymbol{p}}_b^\top \widetilde{\boldsymbol{V}}_b, \qquad \boldsymbol{o}, \widetilde{\boldsymbol{o}}_b \in \mathbb{R}^{1 \times d}. \tag{18}$$

We assume real arithmetic, exact group endpoints, and nearest rounding to 2^b uniformly spaced levels. The attention weights are normalized over the observed visible positions, which must form a nonempty set. For a fixed mask, the proof applies to the visible positions while retaining the original quantization groups and their ranges.

Proposition 1 (window-local attention bound). Under these assumptions, Eq. (12) holds with κ, ν, ρ defined in Eq. (11). Steps 2–4 below give the proof.

2. Bound the errors in keys, values, and query–key scores. A b-bit quantizer divides a group's range into $2^b - 1$ equal intervals. Nearest rounding therefore changes an entry by at most half an interval. Using the same groups as in the main text gives

$$\begin{aligned} \big|[\widetilde{\boldsymbol{K}}_b - \boldsymbol{K}]_{ij}\big| &\le \frac{\max_{i' \in \mathcal{G}} [\boldsymbol{K}]_{i'j} - \min_{i' \in \mathcal{G}} [\boldsymbol{K}]_{i'j}}{2(2^b - 1)}, \\ \big|[\widetilde{\boldsymbol{V}}_b - \boldsymbol{V}]_{ij}\big| &\le \frac{\max_{j' \in \mathcal{G}} [\boldsymbol{V}]_{ij'} - \min_{j' \in \mathcal{G}} [\boldsymbol{V}]_{ij'}}{2(2^b - 1)}. \end{aligned} \tag{19}$$

In the first line, $\mathcal{G}$ is the key-token group containing i; in the second, it is the value-feature group containing j. The primed indices run over entries within that group. No separate variable is needed for either group range.

Since the query is fixed, each query–key score changes only through key rounding. Applying the triangle inequality and then the definition of κ gives

$$\begin{aligned} \|\widetilde{\boldsymbol{z}}_b - \boldsymbol{z}\|_\infty &\le \frac{1}{\sqrt{d}} \max_i \sum_j |q_j| \big|[\widetilde{\boldsymbol{K}}_b - \boldsymbol{K}]_{ij}\big| \\ &\le \frac{\sum_j |q_j| \max_{\mathcal{G}} \big(\max_{i' \in \mathcal{G}} [\boldsymbol{K}]_{i'j} - \min_{i' \in \mathcal{G}} [\boldsymbol{K}]_{i'j}\big)}{2(2^b - 1)\sqrt{d}} = \frac{\kappa}{2(2^b - 1)}. \end{aligned} \tag{20}$$

For values, every feature in a group has the same rounding-error bound. Summing squared entry bounds therefore contributes the group's actual size $|\mathcal{G}|$. With $[\boldsymbol{V}]_{i,:}$ denoting row i, we obtain

$$\begin{aligned} \max_i \big\|[\widetilde{\boldsymbol{V}}_b - \boldsymbol{V}]_{i,:}\big\|_2 &\le \frac{\max_i \sqrt{\sum_{\mathcal{G}} |\mathcal{G}| \big(\max_{j \in \mathcal{G}} [\boldsymbol{V}]_{ij} - \min_{j \in \mathcal{G}} [\boldsymbol{V}]_{ij}\big)^2}}{2(2^b - 1)} \\ &= \frac{\nu}{2(2^b - 1)}. \end{aligned} \tag{21}$$

These bounds use the complete observed groups. Restricting attention to fewer visible rows cannot increase the resulting maxima.

3. Bound the change in attention weights. To pass from query–key scores to attention weights, consider the softmax Jacobian $\boldsymbol{J}_{\boldsymbol{p}} = \operatorname{diag}(\boldsymbol{p}) - \boldsymbol{p}\boldsymbol{p}^\top$. For any perturbation vector $\boldsymbol{\delta}$, its i-th output entry is $p_i(\delta_i - \boldsymbol{p}^\top \boldsymbol{\delta})$. Since $p_i \ge 0$ and $\sum_i p_i = 1$,

$$\begin{aligned} \|\boldsymbol{J}_{\boldsymbol{p}} \boldsymbol{\delta}\|_1 &= \sum_i p_i \big|\delta_i - \boldsymbol{p}^\top \boldsymbol{\delta}\big| \\ &\le \sqrt{\sum_i p_i \big(\delta_i - \boldsymbol{p}^\top \boldsymbol{\delta}\big)^2} \\ &= \sqrt{\sum_i p_i \delta_i^2 - (\boldsymbol{p}^\top \boldsymbol{\delta})^2} \le \|\boldsymbol{\delta}\|_\infty. \end{aligned} \tag{22}$$

The first inequality is the Cauchy–Schwarz inequality with weights p_i. The last inequality follows from $\sum_i p_i \delta_i^2 \le \|\boldsymbol{\delta}\|_\infty^2$. Applying this Jacobian bound along the line segment from $\boldsymbol{z}$ to $\widetilde{\boldsymbol{z}}_b$ gives

$$\|\widetilde{\boldsymbol{p}}_b - \boldsymbol{p}\|_1 \le \|\widetilde{\boldsymbol{z}}_b - \boldsymbol{z}\|_\infty \le \frac{\kappa}{2(2^b - 1)}. \tag{23}$$

Thus, κ controls the possible change in attention weights caused by key rounding.

4. Combine the two effects on the attention output. Adding and subtracting $\widetilde{\boldsymbol{p}}_b^\top \boldsymbol{V}$ separates the output error into two terms:

$$\widetilde{\boldsymbol{o}}_b - \boldsymbol{o} = (\widetilde{\boldsymbol{p}}_b - \boldsymbol{p})^\top \boldsymbol{V} + \widetilde{\boldsymbol{p}}_b^\top (\widetilde{\boldsymbol{V}}_b - \boldsymbol{V}). \tag{24}$$

The first term changes the weights while holding values fixed; the second changes the values under the quantized weights. To bound the first term, let $\boldsymbol{c} \in \mathbb{R}^{1 \times d}$ contain the coordinate midpoints of the

value vectors:

$$c_j = \frac{\max_i [\boldsymbol{V}]_{ij} + \min_i [\boldsymbol{V}]_{ij}}{2}, \qquad \max_i \|[\boldsymbol{V}]_{i,:} - \boldsymbol{c}\|_2 \le \frac{1}{2}\sqrt{\sum_j \big(\max_i [\boldsymbol{V}]_{ij} - \min_i [\boldsymbol{V}]_{ij}\big)^2} = \rho. \tag{25}$$

Both attention-weight vectors sum to one, so $\sum_i (\widetilde{p}_{b,i} - p_i) = 0$. Subtracting the same midpoint from every value row therefore leaves the first term unchanged:

$$\begin{aligned} \|(\widetilde{\boldsymbol{p}}_b - \boldsymbol{p})^\top \boldsymbol{V}\|_2 &= \left\| \sum_i (\widetilde{p}_{b,i} - p_i)([\boldsymbol{V}]_{i,:} - \boldsymbol{c}) \right\|_2 \\ &\le \sum_i |\widetilde{p}_{b,i} - p_i| \, \|[\boldsymbol{V}]_{i,:} - \boldsymbol{c}\|_2 \le \rho \|\widetilde{\boldsymbol{p}}_b - \boldsymbol{p}\|_1. \end{aligned} \tag{26}$$

For the second term, the quantized attention weights remain nonnegative and sum to one. Its norm is therefore at most the largest value-row rounding error. Combining both terms yields

$$\begin{aligned} \|\widetilde{\boldsymbol{o}}_b - \boldsymbol{o}\|_2 &\le \rho \|\widetilde{\boldsymbol{p}}_b - \boldsymbol{p}\|_1 + \sum_i \widetilde{p}_{b,i} \big\| [\widetilde{\boldsymbol{V}}_b - \boldsymbol{V}]_{i,:} \big\|_2 \\ &\le \rho \|\widetilde{\boldsymbol{p}}_b - \boldsymbol{p}\|_1 + \max_i \big\| [\widetilde{\boldsymbol{V}}_b - \boldsymbol{V}]_{i,:} \big\|_2 \\ &\le \frac{\rho\kappa + \nu}{2(2^b - 1)}. \end{aligned} \tag{27}$$

This proves Eq. (12) without assuming independent or zero-mean quantization errors. The factor ρ is needed because changes in attention weights matter only through differences between the value vectors. If all value rows are identical, $\rho = 0$ and the first error term vanishes.

5. Square and average to obtain the sensitivity score. Squaring the nonnegative bound gives

$$\|\widetilde{\boldsymbol{o}}_b - \boldsymbol{o}\|_2^2 \le \frac{(\rho\kappa + \nu)^2}{4(2^b - 1)^2}. \tag{28}$$

The coefficient $(\rho\kappa + \nu)^2/4$ depends on the observed computation, whereas $(2^b - 1)^{-2}$ depends only on precision. We compute the coefficient for each observed layer ℓ, query head h, and query position u, and average within the layer as in Eq. (13). Let $\mathcal{H}(h)$ identify the KV head used by query head h. Then $\kappa_{\ell,h,u}$ uses that query and its associated key head, while $\rho_{\ell,\mathcal{H}(h)}$ and $\nu_{\ell,\mathcal{H}(h)}$ use the corresponding value head. The layer score is

$$s_{a,\ell,r} = \frac{\sum_{h,u} \big(\rho_{\ell,\mathcal{H}(h)} \kappa_{\ell,h,u} + \nu_{\ell,\mathcal{H}(h)}\big)^2}{4 \sum_{h,u} 1}. \tag{29}$$

The sums enumerate the observed head–query pairs in layer ℓ; the factor $1/4$ comes from the squared bound. Query heads sharing one KV head reuse its key and value statistics but retain their own query-dependent κ. For one layer and candidate width b, the bit-dependent factor can be taken outside the average:

$$\mathrm{Average}_{h,u}\big[\|\widetilde{\boldsymbol{o}}_{\ell,h,u,b} - \boldsymbol{o}_{\ell,h,u}\|_2^2\big] \le \frac{s_{a,\ell,r}}{(2^b - 1)^2}. \tag{30}$$

Squaring precedes averaging. This bounds the mean of the squared local output errors, not the squared error after combining all heads or propagating through later layers.

6. Construct the estimate over remaining blocks and invocations. The inequality above concerns the recorded observation window. To use it for allocation, we treat $s_{a,\ell,r}$ as a surrogate for prospective blocks in that layer and evaluate $s_{a,\ell,r}/(2^{b_{a,\ell}} - 1)^2$. This extrapolates from current observations; a future block need not have the same entries or quantization groups.

Eq. (7) gives remaining quantizable capacity per layer; it is used in the byte reservation, not to predict generation length. The audited controller ranks layer scores and minimizes the additive surrogate

$$\widehat{\mathcal{E}}_r(\boldsymbol{b}) = \sum_{a \in \mathcal{A}_r} \sum_{\ell \in \mathcal{L}} \frac{s_{a,\ell,r}}{(2^{b_{a,\ell}} - 1)^2}. \tag{31}$$

This is an allocation objective, not a bound on final-answer error. Already sealed blocks are excluded from its adjustable component because their precision cannot change; their bytes still count toward the reservation. A layer with no current observation receives the median valid resident score, or 1 when no score exists.

The bound applies to attention computed within the observed window under exact group endpoints and nearest rounding. Assuming that recent observations remain representative of the remaining blocks, we use this bound to construct the additive allocation objective in Eq. (8), rather than a bound on final-answer error.

E.2 Memory Reservation Calculation

We detail the four terms in Eq. (14). All costs are measured in bytes, and the L quantized layers are assumed to share the same cache geometry. The formula reserves memory for existing storage and further cache growth, rather than measuring only the current occupancy.

Storage of one quantized block. Let H and H_{q} denote the numbers of KV heads and query heads. Each block contains P tokens per layer, with d_{K} key features and d_{V} value features per KV head. With group size G, keys are grouped across tokens within each feature, and values across features within each token. A block stored at b bits occupies

$$\begin{aligned}\mathcal{B}_{\mathrm{blk}}(b) = H\left(\left\lceil\frac{Pd_{\mathrm{K}}b}{8}\right\rceil + \left\lceil\frac{Pd_{\mathrm{V}}b}{8}\right\rceil\right) \\ + 4H\left(\left\lceil\frac{P}{G}\right\rceil d_{\mathrm{K}} + P\left\lceil\frac{d_{\mathrm{V}}}{G}\right\rceil\right).\end{aligned} \tag{32}$$

The first line counts packed key and value codes, with ceilings accounting for the final partial byte in each head's payload. The implementation divides each code across a fixed base payload and zero or more one-bit refinement planes. The audited 2-bit target uses a physical 1-bit base, so a 1-bit block consumes fewer packed-code bytes than a 2-bit block and can earn byte credit for a 3-bit block. The 3- and 4-bit targets use 2- and 3-bit bases. Refinement storage is committed on demand; the reservation formula counts the selected code width, while the physical audit also reports committed refinement storage. The second line counts $H\lceil P/G\rceil d_{\mathrm{K}}$ key groups and $HP\lceil d_{\mathrm{V}}/G\rceil$ value groups. Each group stores a two-byte scale and a two-byte offset, giving four bytes per group.

Existing and additional quantized storage. The term $I_{a,r}$ sums codes, scales, and offsets in already quantized blocks across layers at their stored widths. Given capacity T_a, a native-precision recent window of R tokens, and $z_{a,\ell,r}$ already quantized tokens in layer ℓ, Eq. (7) gives its remaining block count $n_{a,\ell,r}$. The additional quantized reservation is $\sum_\ell n_{a,\ell,r}\mathcal{B}_{\mathrm{blk}}(b_{a,\ell})$.

Unquantized states and observations. Let u be the native bytes per scalar, A the maximum tokens appended in one forward pass, W the observation-window limit, and $\tau_{a,r}$ the current unquantized tail length. We reserve

$$\begin{aligned}U_{a,r} &= LuH(d_{\mathrm{K}} + d_{\mathrm{V}})\min\{T_a - z_{a,r}, \max(\tau_{a,r}, P - 1 + R + A)\}, \\ O_a &= Lu\left[H_{\mathrm{q}}d_{\mathrm{K}}\min(W, A, T_a) + H(d_{\mathrm{K}} + d_{\mathrm{V}})\min(W, T_a)\right].\end{aligned} \tag{33}$$

In $U_{a,r}$, $R + P - 1 + A$ allows for recent tokens, an incomplete block, and one forward pass's new entries. The maximum also covers the current tail, while the minimum limits the allowance to capacity not already quantized. Multiplying by $LuH(d_{\mathrm{K}} + d_{\mathrm{V}})$ converts the token allowance to bytes. In O_a, the first term covers current query observations and the second covers recent key and value observations. This space remains reserved between observations.

Total reservation. Adding existing quantized storage, additional quantized capacity, unquantized states, and observations gives Eq. (14). The additional packed capacity and unquantized allowance intentionally overlap because entries are stored at native precision before quantization; the reservation therefore includes both forms. Observation tensors are counted once. The strict average-width envelope in Eq. (15) compares this packed-KV reservation with the same reservation evaluated at the target width for all sealed and prospective blocks. It is a KV-storage budget, not a bound on total process GPU memory. Model weights, temporary attention and codec workspace, Python records, and allocator overhead are excluded.

E.3 Balanced Agent–Layer Allocation

For equal-sized KV layers, a lower-bit and a higher-bit choice have opposite, equal packed-code byte deviations from the target. We pair these choices *within each invocation*, so the exact target packed-code mean does not depend on the lifetime of another agent. Let $s_{a,(1),r} \le \cdots \le s_{a,(L),r}$ be the layer sensitivities in ascending order. At a fixed pair count k, the optimal arrangement for Eq. (8) assigns $b_{\text{avg}} - 1$ to the k smallest scores and $b_{\text{avg}} + 1$ to the k largest scores. This follows by exchanging any inverted pair: the bit-dependent error coefficient decreases as the bit width increases, so assigning the larger width to the larger score cannot increase the objective. The remaining $L - 2k$ layers use b_{avg}. We evaluate $k = 1, \ldots, \lfloor (L-1)/2 \rfloor$ and choose the lowest estimated error, breaking ties toward fewer exchanges. The lower bound guarantees all three candidate widths, and the upper bound keeps at least one target-width layer. Scores tied before any observation are ordered by a deterministic invocation–layer hash. If a new vector duplicates one already assigned or physically sealed by a teammate, we choose the distinct vector reachable by one swap of differently coded layers with the smallest increase in the surrogate. This tie repair preserves the width counts and byte budget.

Historical sealed blocks retain their widths. The reservation in Eq. (14) includes those historical bytes, future blocks at the selected layer widths, the raw tail, and observations. The controller compares the reservation with the configured team budget and the average-width entitlement at every allocation event. The physical audit separately checks all sealed widths, the exact mean packed-code width, and compressed KV storage; an allocation proposal by itself does not certify a valid result. This constrained selection is the audited end-to-end policy. The earlier agent-level Pareto solver remains available in code for controls and ablations but is not used to generate the audited agent–layer result cells.

F End-to-end Algorithm

Algorithm 2 presents the end-to-end procedure of CoMixKV, combining online sensitivity estimation, joint bit-width allocation under a shared memory budget, and quantization of newly eligible blocks. It also describes cache initialization, prefix reuse, and cache release as agent invocations progress. Updated bit widths apply only to newly quantized blocks, while retained quantized blocks remain unchanged.

In Algorithm 2, selected widths are prospective: a score refresh changes only subsequent block writes, while sealed blocks remain immutable. Each invocation's balanced layer vector has all three candidate widths and mean b_{avg}. The runtime checks the effective byte envelope at allocation events and records the widths actually written by the packed codec. Accuracy cells are reported only after full-cohort execution and the post-run quantization audit pass.

G Details of Baselines, Models and Benchmarks

G.1 Details of Baselines

We describe the three quantization baselines in Tables 1 and 2, together with the unquantized Vanilla reference.

- **Vanilla (FullKV)**. Vanilla stores the KV cache without quantization and provides the native-precision reference. Table 2 includes the Qwen3-32B reference with 16-bit KV states, and the controlled memory and latency analysis uses BF16 KV storage under the label FullKV.
- **KIVI** (Liu et al., 2024c). KIVI is a tuning-free asymmetric KV-cache quantization method. It quantizes keys per channel to isolate the error caused by persistent channel-wise outliers and values per token to reduce attention-output error. KIVI retains a small recent window at the model's native precision and quantizes older entries in groups, supporting streaming updates during autoregressive decoding without model retraining.
- **TurboQuant** (Zandieh et al., 2026). TurboQuant is a data-oblivious online vector quantizer. It first applies a random rotation that concentrates the coordinate distributions and

Algorithm 2 End-to-end agent–layer KV-cache quantization in CoMixKV

Require: Multi-agent runtime, resident set $\mathcal{A}$, target b_{avg}, layer set $\mathcal{L}$
Require: Recent window R, observation window W, refresh interval J, shared budget M
Ensure: Final runtime prediction $\boldsymbol{z}$, or an explicit execution error

for each scheduled prefill or decode forward of invocation a **do**
  Reconcile a's private cache and reusable prefix; retain sealed block widths
  if the resident set changed or a fresh observation is available **then**
    Compute $s_{a,\ell}$ from ordinary-forward $\boldsymbol{Q}/\boldsymbol{K}/\boldsymbol{V}$ for each observed layer
    Impute missing layer scores from the resident median, or 1 if none exists
    for each resident invocation a **do**
      Sort its L layers by score, with deterministic tie breaking
      for $k = 1, \ldots, \lfloor (L-1)/2 \rfloor$ **do**
        Assign k lowest layers $b_{\text{avg}} - 1$, k highest $b_{\text{avg}} + 1$, and the rest b_{avg}
        Evaluate the score surrogate in Eq. (8)
      end for
      Retain the minimum-surrogate layer vector; charge its reserved bytes
    end for
    Record the selected reservation and average-width byte envelope
  end if
  Run the ordinary forward with sealed blocks at their stored widths and the recent raw tail
  Append new keys and values at native precision
  for each newly eligible complete block j and KV layer ℓ **do**
    Seal $\mathcal{B}_{a,j}^{(\ell)}$ using current $b_{a,\ell}$ and record its physical width
  end for
  Continue agent actions; release completed invocations and their private caches
end for
Audit candidate-width coverage, sealed-code mean, and physical KV compression
return the final prediction $\boldsymbol{z}$

then uses distribution-aware scalar quantizers. To correct bias in inner-product estimation, TurboQuant stores a one-bit Quantized Johnson–Lindenstrauss (QJL) sketch of the quantization residual. The method requires neither model training nor calibration data and targets both KV-cache storage and attention-score computation.

- **PM-KVQ** (Liu et al., 2026). PM-KVQ targets the cumulative error that low-bit quantization introduces during long chain-of-thought decoding. It begins with 16-bit KV states and progressively reduces the precision of the existing cache through 16-to-8-bit, 8-to-4-bit, and 4-to-2-bit transitions as the sequence grows. An equivalent-right-shift operator performs each transition without full dequantization. A calibration stage profiles transformer-block sensitivity, assigns block-specific final bit widths under a memory budget, and uses positional interpolation to approximate long-context key distributions from short sequences.

G.2 Details of Models

We evaluate three open-weight model backbones covering dense and hybrid-attention architectures: Qwen3-32B, Qwen3.8-27B, and Gemma-4-31B-it.

- **Qwen3-32B** (Yang et al., 2025). Qwen3-32B is a 32B-parameter dense model from the Qwen3 family. Qwen3 unifies thinking and non-thinking modes within a single model and supports adjustable thinking budgets, allowing inference-time computation to be adapted to task difficulty. The family is trained for broad capabilities including mathematical reasoning, coding, instruction following, multilingual understanding, and agent tasks.
- **Qwen3.8-27B** (Qwen Team, 2026). Qwen3.8-27B is a 27B-parameter native vision-language model with a hybrid-attention backbone. Its 64 language-model layers follow a repeated structure in which three Gated DeltaNet linear-attention layers are followed by one gated full-attention layer, resulting in 48 linear-attention layers and 16 full-attention layers. The model supports a native context length of 262K tokens, multi-token prediction, configurable reasoning effort, and long-horizon agentic execution.
- **Gemma-4-31B-it** (Team et al., 2026). Gemma-4-31B-it belongs to the Gemma 4 family of open-weight, natively multimodal language models. Gemma 4 is designed to improve rea-

soning, long-context capability, and inference efficiency, and introduces an explicit thinking mode that allows the model to generate internal reasoning traces before producing its response. We use the 31B instruction-tuned variant as an additional large-scale backbone from a model family distinct from Qwen.

G.3 Details of Benchmarks

We evaluate four benchmarks covering knowledge-intensive search, agentic reasoning, mathematical reasoning, and scientific reasoning.

Knowledge-intensive Search.

- **Bamboogle** (Press et al., 2023). Bamboogle was introduced to study the compositionality gap of language models. Its questions require multiple inferential steps and are deliberately constructed so that the required facts are unlikely to have appeared together during pretraining. Solving the benchmark therefore requires a model to retrieve or infer individual facts and correctly compose them into the final answer.

Agentic Reasoning.

- **GAIA** (Mialon et al., 2024). GAIA evaluates general AI assistants using real-world questions that require combinations of reasoning, web browsing, multimodal understanding, and tool use. The benchmark contains tasks of varying difficulty and is designed around problems that are straightforward for humans but difficult for AI systems without reliable multi-step interaction with external environments.

Mathematical Reasoning.

- **AIME25** (Zhang & Team, 2025). AIME25 consists of problems from the 2025 American Invitational Mathematics Examination. These competition-level problems require multi-step mathematical reasoning across areas such as algebra, geometry, number theory, and combinatorics, with each problem having an exact integer answer.

Scientific Reasoning.

- **GPQA** (Rein et al., 2024). GPQA (Graduate-Level Google-Proof Q&A) contains expert-written multiple-choice questions in biology, physics, and chemistry. The questions are intentionally designed to be difficult to answer through simple web search and instead require graduate-level domain knowledge and multi-step scientific reasoning.

H Additional Experimental Analysis

H.1 Hyperparameter Sensitivity

We study the observation-window length W and sensitivity-refresh interval J on the AIME25 dataset using Qwen3-32B with an average 4-bit KV-cache budget. Each candidate changes one parameter at a time. Table 7 reports the results for all four settings.

Table 7: Hyperparameter sensitivity on AIME25.

W	J	Accuracy (%)
16	128	33.33
32	**128**	**40.00**
64	128	**40.00**
32	64	**40.00**

We use $(W, J) = (32, 128)$ as the default setting because it achieves the highest observed accuracy of 40.00% while avoiding unnecessary online overhead. Increasing the observation window from

$W = 32$ to $W = 64$ does not improve accuracy, indicating that 32 recent tokens already provide sufficient information for estimating layer sensitivity. Likewise, reducing the refresh interval from $J = 128$ to $J = 64$ produces the same accuracy but requires sensitivity estimation and allocation updates twice as frequently. In contrast, reducing the observation window to $W = 16$ decreases accuracy to 33.33%, suggesting that an overly short window provides a less reliable estimate of the current attention behavior. We therefore adopt $(W, J) = (32, 128)$ throughout the main experiments as it matches the best observed accuracy with a shorter observation window and less frequent sensitivity updates.

H.2 Online Allocation Ablation

We isolate the contribution of online sensitivity-based allocation on AIME25 by comparing the default online policy with uniform allocation and fixed sensitivity scores. All runs use Qwen3-32B, an average 4-bit KV-cache budget, and the same byte constraint. Table 8 reports the completed and audited results.

Table 8: Online-allocation ablation on AIME25.

Allocation policy	Accuracy (%)
Online sensitivity updates	40.00
Uniform allocation	36.67
Fixed sensitivity	36.67

The online sensitivity-based policy achieves 40.00% accuracy, outperforming both uniform allocation and fixed sensitivity by 3.34% under the same average bit budget and byte constraint. Uniform allocation ignores differences in quantization sensitivity across layers, while fixed sensitivity preserves an initial allocation but cannot respond to changes in the agent's queries, histories, and attention behavior during execution. The consistent degradation of these two ablations shows that the benefit of CoMixKV does not arise solely from using mixed precision; dynamically updating sensitivity estimates and reallocating precision according to the current computation is important for preserving task accuracy. These results support the central design of CoMixKV, in which online Q/K/V observations continually guide where higher and lower KV-cache precision should be assigned.