FEWER BITS, MORE TOKENS: RETHINKING KV CACHE QUANTIZATION IN AGENTIC CODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Agentic coding has emerged as a prominent application of LLMs, but its multi-turn interaction trajectories continually expand the KV cache, imposing substantial memory demands during inference. KV cache quantization reduces per-token memory usage, but do these savings translate into more efficient agent trajectories? We find that low-bit quantization disproportionately amplifies three failures: repetitive action loops, improper tool use, and insufficient debugging, while increasing the actions and tokens required per task. To identify which attention heads are most sensitive to quantization around these failures, we measure changes in attention distributions and head outputs under controlled KV cache perturbations. Highly sensitive heads comprise only about 5% of all heads and cluster in a few layers. Guided by this localization, we introduce a simple and effective mixed-precision strategy that restores higher precision in selected layers while retaining low precision elsewhere. On SWE-bench Verified with Qwen3-Coder-30B-A3B-Instruct, increasing average precision from 3.0 to 3.35 bits substantially reduces the three behavioral failures and brings task resolution close to BF16. Despite higher per-token precision, observed KV cache usage per case decreases by 12% versus uniform 3-bit quantization. These results shift the focus of agent quantization from storing each token more cheaply to completing tasks more reliably with less memory.

1 INTRODUCTION

Agentic coding extends large language models (LLMs) from isolated code generation to multi-turn problem-solving in software repositories (Jimenez et al., 2024; Yang et al., 2024). To resolve an issue, an agent must inspect code, invoke tools, implement changes, and verify their effects over multiple turns. As the context grows, the growing KV cache imposes a substantial inference-time memory burden and can dominate memory usage in long-context workloads (Hooper et al., 2024). KV cache quantization offers a direct way to reduce this burden. Unlike token eviction and selection methods (Xiao et al., 2024; Li et al., 2024), it lowers the precision of cached representations while retaining the full interaction history (Liu et al., 2024b; Zandieh et al., 2026). Sensitivity-aware mixed-precision approaches further improve compression–quality trade-offs by allocating higher precision to vulnerable components (Li et al., 2025; Zhao et al., 2026).

However, existing KV quantization methods are primarily designed and evaluated on *non-interactive* tasks, leaving their effects on *interactive* agent execution underexplored. Recent studies show that even successfully resolved coding tasks can involve blind retries, regression cycles, and other inefficient behaviors (Shu et al., 2026; Mehtiyev & Assunção, 2026; Majgaonkar et al., 2025; Sahoo et al., 2026), highlighting the importance of sustained execution control (Wang et al., 2026). Quantization may exacerbate these execution inefficiencies, changing not only how the interaction history is stored, but also how it grows: an invalid tool call or ineffective edit may trigger additional attempts. These extra actions and tokens may offset per-token memory savings. This raises our central question: *How does KV cache quantization affect agent behavior and trajectory-level efficiency?*

We address this question through a *behavior-guided diagnosis and mitigation* of KV cache quantization in coding agents. The study begins by characterizing how low-bit quantization affects behavioral failures, actions, and token generation alongside task resolution (Section 3). We then localize quantization sensitivity in failure-associated execution contexts by replaying recorded token sequences and quantizing one attention head's KV cache at a time. Fixing the tokens isolates quantization-induced

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.

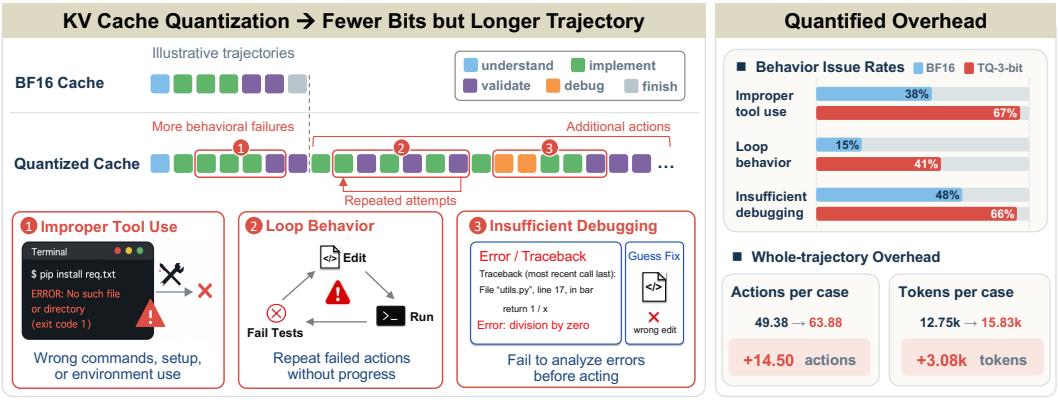


Figure 1: Overview of how KV cache quantization affects agent behavior and trajectory efficiency: higher behavioral failure rates and more actions and tokens per case.

changes in attention from differences in generated sequences (Section 4). The resulting sensitivity patterns guide layer selection for precision restoration. Finally, we compare this allocation with alternatives at the same precision budget to test whether our layer selection improves behavioral reliability and trajectory-level memory efficiency (Section 5).

Specifically, our results reveal **selective behavioral degradation** under low-bit quantization. Across a taxonomy of 13 behavioral failure classes (Wang et al., 2026), the largest increases occur in *repetitive action loops*, *improper tool use*, and *insufficient debugging* (Figure 1). These failures concern execution control: agents repeat ineffective actions, misuse tools, or fail to adjust to execution feedback. In our primary setting, uniform 3-bit quantization requires 29.3% more turns and generates approximately 3.1k additional tokens per case, while achieving a lower task resolve rate. Diagnosis in failure-associated contexts identifies high quantization sensitivity in about 5% of attention heads, concentrated in a few layers, providing candidate targets for precision restoration. Restoring precision in these layers reduces the three highlighted failures and raises the resolve rate from 38.8% to 41.6%, approaching BF16's 42.2%. Crucially, **higher per-token precision yields lower per-case cache usage** in this setting: although average precision increases from 3.0 to 3.35 bits, observed KV cache usage falls from 1.56 to 1.38 GB per case, a reduction of approximately 12%. Comparisons with adapted mixed-precision baselines at the same precision budget support the value of the selected layers, while additional evaluations with Devstral-Small-2-24B and on SWE-bench Multilingual show consistent behavioral recovery and task-resolution gains.

Overall, our main contribution is to reveal, characterize, and mitigate the trajectory-level effects of KV cache quantization in agentic coding. Our behavior-guided diagnosis connects trajectory-level failures to quantization-sensitive layers and guides selective precision restoration. These findings motivate future agent quantization methods to treat precision not only as a storage cost, but also as a resource for reliable execution.

2 Related Work

Agentic Coding and Trajectory Analysis. Agentic coding interleaves reasoning, action, observation, and revision, building on agent paradigms such as ReAct and Toolformer and feedback-driven methods such as Reflexion and Self-Refine (Yao et al., 2023; Schick et al., 2023; Shinn et al., 2023; Madaan et al., 2023). Repository-level systems, including SWE-agent, Agentless, OpenHands, and AutoCodeRover, apply these ideas to issue localization, repair, and validation (Yang et al., 2024; Xia et al., 2024; Wang et al., 2025; Zhang et al., 2024), making sustained execution reliability essential to task success. Final task resolve rates alone do not reveal how solutions are reached. Recent trajectory analyses show that context gathering, validation, and debugging strategies shape outcomes (Mehtiyev & Assunção, 2026; Majgaonkar et al., 2025; Bouzenia & Pradel, 2025). AgentLens and rubric-based critics further identify blind retries, regression cycles, and missing verification even in successful trajectories (Sahoo et al., 2026; Wang et al., 2026). These studies primarily examine agent behavior under a given inference configuration. We instead investigate how KV cache quantization changes these behaviors and use failure-centered internal measurements to guide mitigation.

Efficient Inference and KV Cache Quantization. Efficient LLM inference benefits from IO-aware attention, paged KV cache management, and weight or activation quantization (Dao et al., 2022; Dao, 2024; Kwon et al., 2023; Dettmers et al., 2022; Frantar et al., 2023; Xiao et al., 2023; Lin et al., 2024; Shao et al., 2024). For long contexts, growing KV caches motivate token eviction and selection (Zhang et al., 2023; Xiao et al., 2024; Ge et al., 2024; Li et al., 2024; Cai et al., 2025). KV cache quantization instead lowers the precision of cached keys and values while retaining tokens, as explored by KIVI, KVQuant, TurboQuant, and related methods (Liu et al., 2024b; Hooper et al., 2024; Zandieh et al., 2026; Liao & Wen, 2026; Son et al., 2025). Evaluations commonly emphasize numerical fidelity, perplexity, long-context tasks, and aggregate task performance. These measures do not directly characterize execution-control failures or the additional actions and tokens they may induce. In agentic workflows, the generated trajectory itself can change under quantization, making per-case cache usage depend on both representation precision and execution behavior. Our analysis complements conventional evaluations by examining this trajectory-level trade-off.

Sensitivity-Aware Precision Allocation. Mixed-precision approaches allocate precision according to heterogeneous sensitivity. KVTuner (Li et al., 2025) uses reconstruction error to assess quantization sensitivity; CoopQ (Zhao et al., 2026) considers activation magnitude; and AMQ (Lee et al., 2025) uses layer-wise perplexity degradation to identify components requiring higher precision. Our approach instead measures sensitivity in execution contexts where behavioral failures occur, linking precision allocation to the behaviors targeted for recovery. This makes the diagnosis directly relevant to the behaviors targeted for recovery. Its advantage therefore lies not in mixed precision alone, but in identifying where additional precision most effectively improves agent execution.

3 Trajectory-Level Behavioral Analysis

Execution trajectories connect model capabilities to practical software engineering outcomes, shaping both the reliability and resource cost of agentic coding. Trajectory-level behavior is therefore a key dimension of agent quality, and understanding how KV cache quantization affects this behavior is essential to assessing its practical value.

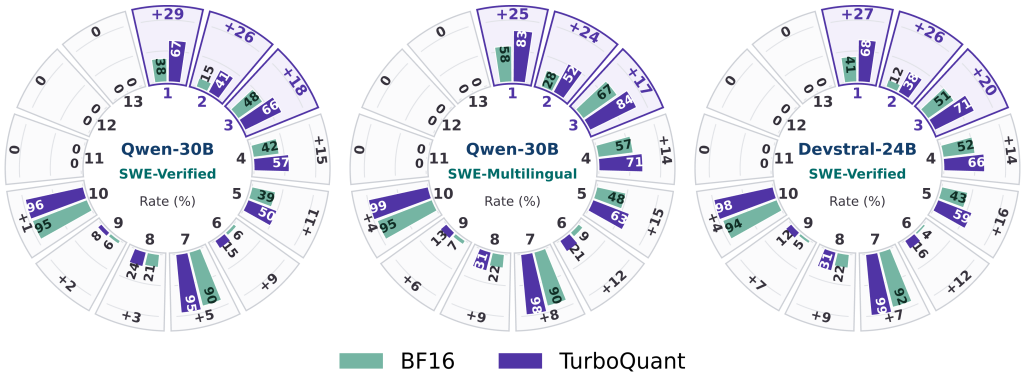
Execution trajectory. Agentic coding proceeds through iterative interactions between an LLM and a repository environment. Given an issue description, the agent first inspects the repository to gather relevant context for subsequent actions. It then repeatedly interacts with the environment by taking actions (e.g., inspecting files, editing code, or running tests) and receiving corresponding observations (e.g., file contents, execution errors, or test results). The accumulated interaction history and environment feedback guide subsequent decisions and action selection. This iterative process forms an execution trajectory. We represent an execution trajectory as an ordered sequence of task context, agent actions, and environment observations:

$$\tau = (u, a_1, o_1, a_2, o_2, \ldots, a_T, o_T), \tag{1}$$

where u denotes the user-provided task context, a_t denotes the agent action at interaction step t, and o_t denotes the corresponding environment observation.

Behavioral failure taxonomy. Following prior work (Wang et al., 2026), we characterize each trajectory using 13 binary indicators of behavioral failures: misunderstanding user intent, ignoring instructions, insufficient code analysis, acting on ambiguous requirements, improper tool use, repetitive action loops, insufficient testing, insufficient debugging, incomplete implementation, file management errors, scope creep, risky actions without permission, and other issues. An LLM auditor annotates the presence of each failure type in a trajectory. An independent judge model produces strongly correlated estimates of the observed degradation patterns (Pearson's $r = 0.92$). Definitions, annotation procedures, and evaluator validation are provided in Appendix B.

Evaluation setting. Our primary analysis uses Qwen3-Coder-30B-A3B-Instruct with OpenHands on SWE-bench Verified, comparing BF16 inference with uniform KV cache quantization using TurboQuant (Zandieh et al., 2026), a state-of-the-art method for online vector quantization. Further implementation details and the rationale for this setting are provided in Appendix A. To assess whether the observed patterns extend beyond this setting, we also evaluate Qwen3-Coder-30B-A3B-Instruct on SWE-bench Multilingual and Devstral-Small-2-24B on SWE-bench Verified.



1 **Improper Tool Use or Setup** 2 **Loop Behavior** 3 **Insufficient Debugging** 4 Incompl. impl. 5 Insuff. analysis
6 Other issues 7 Insuff. test. 8 Instr. viol. 9 Intent err. 10 File err. 11 Insuff. clar. 12 Scope creep 13 Risky acts

Figure 2: Behavioral failure rates under BF16 and TurboQuant across three settings. Highlighted sectors 1–3 mark the largest increases; outer labels show percentage-point changes from BF16.

Quantization selectively amplifies three execution-control failures. In Figure 2, across the 13 behavioral failure classes, the largest increases occur in *improper tool use*, *repetitive action loops*, and *insufficient debugging*: 29, 26, and 18 percentage points in our primary setting, respectively. Action loops rise from 15% under BF16 to 41% under quantization, approximately 2.7 times the BF16 rate. The same three failure types increase substantially across model and benchmark evaluations.

These failures reflect difficulties in issuing valid tool calls, avoiding repeated unsuccessful actions, and investigating errors before proceeding. In contrast, high-level interpretation failures, such as misunderstanding user intent, show smaller changes. This pattern identifies execution control as particularly vulnerable to quantization, without implying that task understanding is unaffected.

This behavioral degradation is accompanied by additional execution overhead: quantized agents require 29.3% more turns and generate 3.1k additional tokens per case on average, while achieving a lower task resolve rate than BF16. Per-token memory savings therefore come with longer interaction histories and more actions, partially offsetting the intended trajectory-level efficiency gains.

Case study: loop behavior. Figure 3 illustrates this failure class using the case of `django__django-11211` from SWE-Bench Verified. Under BF16, the agent responds to a failed command by inspecting the error and adjusting its debugging strategy. In contrast, the quantized agent repeatedly edits the same file during turns 59–60, prepares a rollback at turn 63, encounters a missing pytest dependency at turn 64, and continues an edit loop during turns 65–75. Despite receiving the same failure signal, it neither diagnoses the root cause nor meaningfully changes its approach. This example illustrates how a failure to incorporate execution feedback can turn an unsuccessful action into a prolonged loop, consuming additional turns without making progress toward a solution. Additional case studies appear in Appendix B.3.

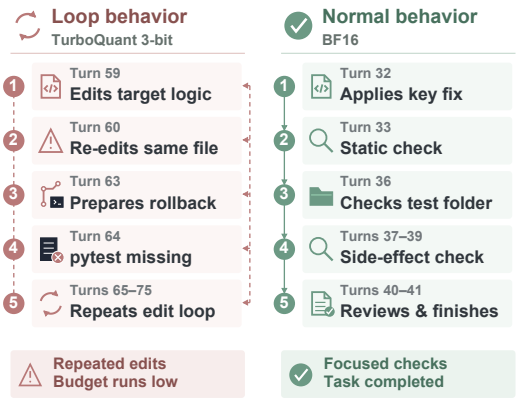


Figure 3: A representative case of loop behavior.

4 Failure-Centered Attention Diagnosis

The behavioral findings in Section 3 raise a natural question: *which attention computations are most sensitive to KV cache quantization around failure onset?* We investigate this question to identify targets for precision restoration without increasing precision throughout the cache.

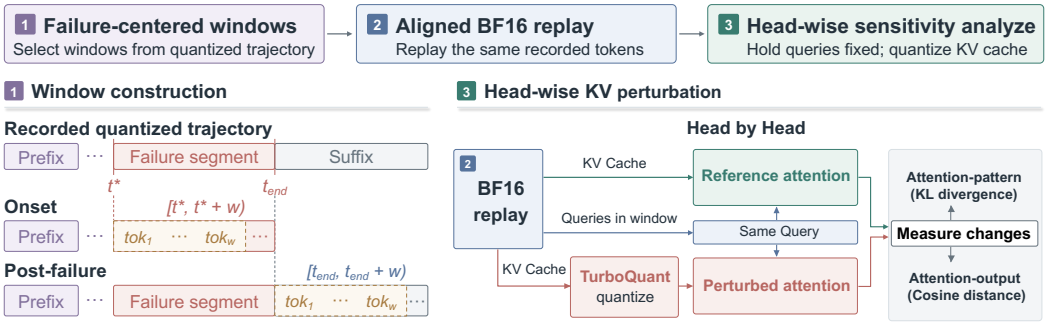


Figure 4: **Framework of failure-centered attention diagnosis.** Aligned BF16 replay and head-wise KV quantization measure attention and head-output changes in onset and post-failure windows.

4.1 Localization Framework

Failure-centered replay and controlled perturbation. The three failures identified in Section 3 highlight two vulnerable aspects of execution: issuing valid actions (*improper tool use*) and adjusting subsequent actions after failure (*repetitive action loops* and *insufficient debugging*). These behaviors span multiple tokens, motivating window-based rather than single-token diagnosis.

For each annotated failure segment in a quantized calibration trajectory, let $t^\star$ and t^{end} denote its start and exclusive end positions. We examine two windows of length w (Figure 4): a *failure-onset window* $[t^\star, t^\star + w)$, which probes computations as the problematic behavior begins, and a *post-failure window* $[t^{\text{end}}, t^{\text{end}} + w)$, which probes the continuation in which the agent may reassess or repeat its actions. Both windows are applied to all three failure categories. They select query positions for diagnosis; each query still attends to its full causally visible prefix.

We replay identical token sequences under BF16 and controlled quantization, quantizing one target head's KV cache at a time while retaining BF16 elsewhere. This comparison measures local sensitivity to the targeted perturbation without confounding it with differences in generated trajectories.

Sensitivity metrics and aggregation. For layer ℓ, head h, and query position q, let $p^{\text{FP}}_{\ell,h,q}$ and $p^{\text{Q}}_{\ell,h,q}$ denote the BF16 and quantized attention distributions, and let $o^{\text{FP}}_{\ell,h,q}$ and $o^{\text{Q}}_{\ell,h,q}$ denote the corresponding head contributions after the output projection. We measure

$$d^{\text{attn}}_{\ell,h,q} = \mathrm{KL}\left(p^{\text{FP}}_{\ell,h,q} \,\middle\|\, p^{\text{Q}}_{\ell,h,q}\right), \quad d^{\text{out}}_{\ell,h,q} = 1 - \cos\left(o^{\text{FP}}_{\ell,h,q}, o^{\text{Q}}_{\ell,h,q}\right). \tag{2}$$

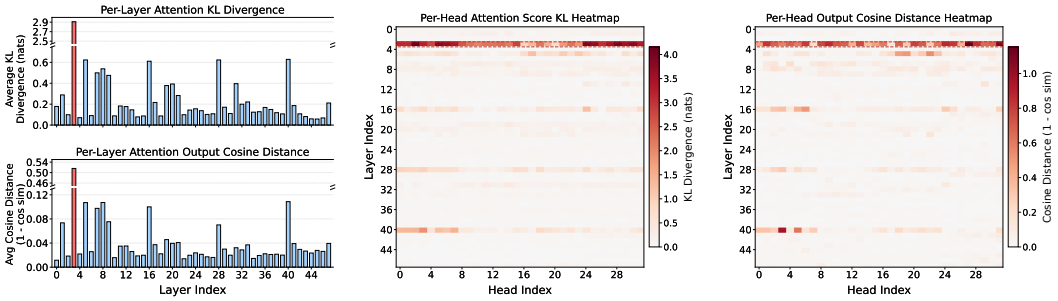
Attention KL captures routing changes induced by `key` quantization, while output cosine distance captures directional changes arising from both `key` and `value` quantization. The following bound motivates these complementary measurements.

Theorem 1 (Local KV quantization error)**.** *For a fixed head and query, let $p, \tilde{p}$ denote the BF16 and quantized attention distributions, with corresponding values $v_j, \tilde{v}_j$. Define the pre-projection outputs $z = \sum_j p_j v_j$ and $\tilde{z} = \sum_j \tilde{p}_j \tilde{v}_j$. If $\|v_j\|_2 \le M$ for all j, then*

$$\|z - \tilde{z}\|_2 \le M\sqrt{2\ \mathrm{KL}(p\|\tilde{p})} + \sum_j \tilde{p}_j \|v_j - \tilde{v}_j\|_2. \tag{3}$$

The first term is controlled by attention KL and captures routing drift driven primarily by `key` quantization, while the second reflects `value` distortion not captured by attention KL alone. We therefore complement attention KL with the head-output cosine distance, which jointly measures whether the information written forward by the head changes after quantization. The detailed proof is provided in Appendix D.

We average each diagnostic over query positions and calibration examples within each window, then give equal weight to the onset and post-failure windows to obtain head-level scores. Averaging over heads yields layer-level scores. We use attention KL to rank layers for precision restoration and retain output cosine distance as a complementary diagnostic.



(a) Layer-wise sensitivity. (b) Head-level attention disruption. (c) Head-level output distortion.

Figure 5: **Quantization sensitivity across layers and heads.** (a) Layer-level sensitivity aggregated from head-wise measurements. (b) Attention-routing changes measured by attention KL. (c) Directional changes in projected head outputs measured by cosine distance.

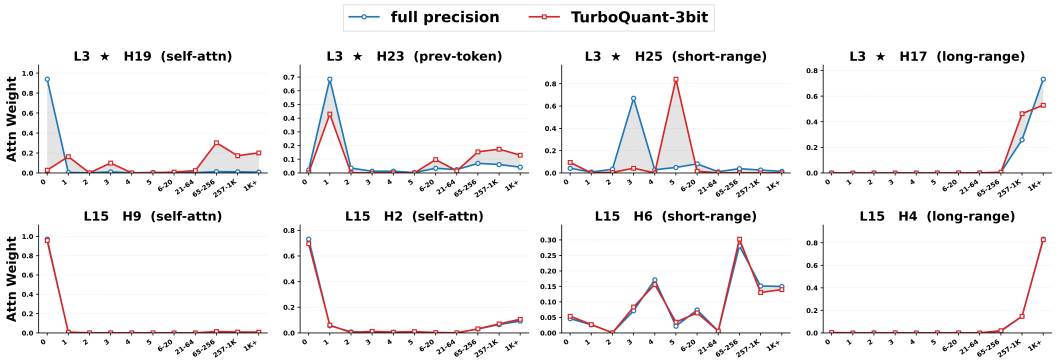


Figure 6: Attention patterns before and after quantization for heads in the sensitive Layer 3 and low-sensitive Layer 15. The horizontal axis denotes the relative distance between queries and keys.

4.2 Sensitivity Patterns

Finding 1: Sensitivity is sparse and layer-localized. Figure 5a shows that quantization sensitivity concentrates in a small subset of the 48 attention layers. Layer 3 is the strongest outlier. The head-level maps in Figures 5b and 5c reveal corresponding layer-wise structure: sensitive heads cluster within a few layers, although heads in the same layer can differ in their response to quantization. Further statistical analyses of layer sensitivity are provided in Appendix G.1. The output-distance map broadly agrees with the attention-KL localization, highlighting Layer 3 and additional peaks at Layers 5, 16, 28, and 40. The identified sensitivity is therefore reflected in both attention routing and projected head outputs under controlled replay, although these local measurements alone do not establish which perturbations cause downstream behavioral failures.

Similar concentration patterns appear across five backbones ranging from 8B to 123B, spanning dense and MoE architectures and including hybrid-attention models; this consistency concerns the concentration of sensitivity, not identical sensitive layers or heads across models. Full cross-model results and primary-model confidence intervals are in Appendices E.2.

Finding 2: Sensitive layers exhibit disrupted local attention. To characterize the attention patterns associated with high sensitivity, we examine attention distributions as a function of relative token distance (Figure 6; additional examples appear in Figure 11). Under BF16, Head 19 in the sensitive Layer 3 exhibits a strong peak at the current token, while Head 23 focuses on immediately preceding tokens. Quantization weakens both peaks and shifts attention mass toward more distant positions. In contrast, the examined heads in the low-sensitivity Layer 15 retain comparatively stable distributions. This cross-layer comparison illustrates how quantization can disrupt local attention in a sensitive layer while leaving attention patterns in a less sensitive layer relatively unchanged. Together, these examples complement the layer-level sensitivity analysis by illustrating the attention disruptions observed in the examined heads and contexts, providing further motivation for selective precision restoration at the layer level.

Table 1: Behavioral failures, task resolution, and KV cache usage under different precision allocations on Qwen3-Coder-30B-A3B-Instruct and SWE-bench Verified. Pink rows mark the three execution-control failures most amplified by quantization.

Metric	BF16	TQ-3bit	TQ-Bd	KVTuner	CoopQ	AMQ	TQ-Mix
Improper Tool Use or Setup (↓)	38%	67%	64%	54%	55%	57%	**43%**
Loop Behavior (↓)	15%	41%	40%	31%	33%	30%	**20%**
Insufficient Debugging (↓)	48%	66%	62%	53%	56%	52%	**48%**
Incomplete Implementation (↓)	42%	57%	52%	47%	51%	58%	51%
Insufficient Analysis (↓)	39%	50%	47%	47%	40%	53%	46%
Other Agent Issue (↓)	6%	15%	13%	16%	12%	6%	8%
Insufficient Testing (↓)	90%	95%	93%	91%	88%	96%	93%
Did Not Follow Instruction (↓)	21%	24%	24%	23%	17%	29%	23%
Misunderstood Intention (↓)	6%	8%	7%	11%	6%	3%	5%
File Management Errors (↓)	95%	96%	97%	99%	91%	94%	95%
Insufficient Clarification (↓)	0%	0%	0%	0%	0%	0%	0%
Scope Creep (↓)	0%	0%	0%	0%	0%	0%	0%
Risky Actions or Permission (↓)	0%	0%	0%	0%	0%	0%	0%
Average Bits (↓)	16bit	3bit	4.1bit	3.35bit	3.35bit	3.35bit	**3.35bit**
KV Cache per Case (↓)	6.24 GB	1.56 GB	2.25 GB	1.45 GB	1.48 GB	1.53 GB	**1.38 GB**
Resolved Rate (↑)	42.2%	38.8%	39.4%	39.0%	40.0%	39.2%	**41.6%**

5 Sensitivity-Guided Precision Restoration

The preceding analyses identify three execution-control failures amplified by KV cache quantization and localize attention sensitivity to a small subset of layers. We now test whether restoring precision in these layers recovers behavioral reliability while retaining substantial memory savings.

5.1 Restoration Setup

Sensitivity-guided allocation. We convert the failure-centered layer ranking into a three-tier precision allocation: extreme sensitivity outliers receive BF16, the next highest-ranked layers receive 4-bit precision, and the remaining layers retain 3-bit precision. For Qwen3-Coder-30B-A3B-Instruct, this restores Layer 3 to BF16 and Layers 5, 16, 28, and 40 to 4 bits, averaging 3.35 bits. We denote this configuration as **TQ-Mix**; the complete allocation rule is provided in Appendix C.

Baselines. We compare TQ-Mix with six baselines. KVTuner, CoopQ, and AMQ denote adaptations of their respective sensitivity criteria to our KV cache restoration setting. Consistent with TQ-Mix, each restores one layer to BF16 and four to 4 bits, leaving the rest at 3 bits. This holds the restoration mechanism and 3.35-bit average precision budget fixed while varying layer selection. All quantized configurations use TurboQuant (Zandieh et al., 2026).

- **BF16:** all KV caches remain in `bfloat16` without quantization.
- **TQ-3bit:** TurboQuant-3bit-nc is applied uniformly to all KV caches.
- **TQ-Bd:** following the vLLM boundary-layer heuristic,[1] the first two and last two layers remain in BF16 and the rest use 3-bit precision, averaging 4.1 bits.
- **KVTuner** (Li et al., 2025): layers are ranked by *output reconstruction error*, measured as the normalized mean-squared error in attention outputs relative to BF16 when each layer's KV cache is quantized individually. It restores Layer 24 to BF16 and Layers 9, 26, 35, and 38 to 4 bits.
- **CoopQ** (Zhao et al., 2026): layers are ranked by *activation magnitude*, prioritizing larger root-mean-square magnitudes of cached key and value activations. It restores Layer 11 to BF16 and Layers 7, 15, 26, and 38 to 4 bits.

[1] https://docs.vllm.ai/en/latest/api/vllm/model_executor/layers/quantization/turboquant/

Table 2: Generalization across datasets and model families. Parentheses report percentage-point changes relative to TurboQuant-3bit. Failure rates are lower-is-better; resolved rate is higher-is-better. Blue shading indicates our default setting.

Setting	Metric	BF16	TQ-3bit	TQ-Bd	TQ-Mix
SWE-Multilingual Qwen3-30B	Improper Tool Use ↓	58%	83%	79% (−4 pp)	**61%** (−22 pp)
	Loop Behavior ↓	28%	52%	51% (−1 pp)	**34%** (−18 pp)
	Insufficient Debugging ↓	67%	84%	79% (−5 pp)	**66%** (−18 pp)
	Resolved Rate ↑	38.6%	33.3%	34.6% (+1.3 pp)	**37.3%** (+4.0 pp)
SWE-Verified Devstral-24B	Improper Tool Use ↓	41%	68%	63% (−5 pp)	**45%** (−23 pp)
	Loop Behavior ↓	12%	38%	37% (−1 pp)	**18%** (−20 pp)
	Insufficient Debugging ↓	51%	71%	65% (−6 pp)	**52%** (−19 pp)
	Resolved Rate ↑	66.2%	62.6%	63.4% (+0.8 pp)	**65.8%** (+3.2 pp)

- **AMQ** (Lee et al., 2025): layers are ranked by *perplexity sensitivity*, measured as the increase in validation perplexity relative to BF16 when each layer's KV cache is quantized individually. This extends the original weight-quantization criterion to KV caches. It restores Layer 30 to BF16 and Layers 10, 18, 29, and 36 to 4 bits.

Why layer-level restoration? Although sensitivity is measured at the head level, we aggregate head-level measurements into layer scores for precision allocation. The vLLM release used in our experiments does not natively support mixed KV cache precision across heads within a layer. Efficient head-wise allocation would require additional support for cache layouts, attention kernels, and batched execution. We therefore evaluate all restoration configurations at the layer level within the same serving stack (Kwon et al., 2023).

5.2 Behavioral Recovery and Memory Efficiency

Targeted restoration recovers execution control. In Table 1, TQ-Mix reduces the three execution-control failure rates identified in Section 3 by 18–24 percentage points relative to TQ-3bit. All three return to within 5 percentage points of BF16, with insufficient debugging matching BF16. The resolve rate also recovers from 38.8% to 41.6%, approaching BF16's 42.2%. Restoring a small subset of layers thus recovers much of the observed execution-control degradation, although recovery does not extend uniformly to all behaviors.

Precision allocation matters beyond the bit budget. TQ-Mix outperforms TQ-Bd on the three highlighted failures and task resolution despite using lower average precision (3.35 versus 4.1 bits). At the same 3.35-bit budget, it also outperforms the adapted KVTuner, CoopQ, and AMQ baselines, improving the resolve rate by 2.8 percentage points versus at most 1.2. These controlled comparisons support the value of failure-centered layer selection, rather than additional precision alone. Measurements around failures may expose execution-critical vulnerabilities less directly captured by reconstruction error, activation magnitude, or perplexity.

Higher precision can reduce per-case cache usage. Despite its higher average precision, TQ-Mix uses less KV cache per case than TQ-3bit (1.38 versus 1.56 GB), remaining well below BF16 (6.24 GB). Together with improved task resolution, this reinforces the distinction between per-token compression and trajectory-level efficiency. More reliable execution may limit repeated actions and history growth, offsetting the additional per-token storage. Although cache measurements alone do not establish this mechanism, they show why precision allocation should be evaluated beyond bit width alone.

5.3 Evaluation across Datasets and Models

Behavioral recovery extends across settings. In Table 2, TQ-Mix consistently outperforms TQ-3bit and TQ-Bd on the three execution-control failures and task resolution. Relative to TQ-3bit, it reduces failure rates by 18–22 percentage points on SWE-bench Multilingual and 19–23 percentage points on Devstral-Small-2-24B. The corresponding resolve rates improve by 4.0 and 3.2 percentage points, reaching within 1.3 and 0.4 percentage points of BF16, respectively. TQ-Bd yields smaller

Table 3: Comparison of KV cache strategies on static coding benchmarks. Bold values denote the best result among compressed configurations. Blue shading indicates our default setting.

Configuration	RepoBench-EM	RepoBench-ES	HumanEval	MBPP	LiveCodeBench-100	Average
BF16	33.73	65.76	62.80	68.87	24.00	51.03
TQ-3bit	31.57	64.22	60.37	61.48	15.00	46.53
TQ-Bd	31.27	63.65	59.76	61.87	17.00	46.71
TQ-Mix	**32.23**	**64.53**	**60.98**	**64.59**	**19.00**	**48.27**

Table 4: Robustness analyses on SWE-bench Verified. Five-layer allocation refers to Layers 3, 5, 16, 28, and 40. Behavioral failure is averaged across the three execution-control classes. Local and Long-range denote the mean pre-quantization attention scores of sensitive heads. FP8 uses E4M3.

(a) Precision-allocation ablation

Precision allocation	Average bits	Resolved rate ↑	Behavioral failure ↓
TQ-Mix	3.35	41.6%	37.0%
Five-layer BF16	4.35	41.8%	34.3%
Five-layer 4-bit	3.10	40.2%	45.0%
Layer 3 BF16	3.27	40.8%	41.7%

(b) Cross-quantizer sensitivity

Quantizer config.	Sensitive heads	Local score	Long-range score
TQ-3bit	5.1%	0.77	0.09
TQ-4bit	3.9%	0.65	0.18
INT4	5.3%	0.62	0.22
FP8	2.5%	0.71	0.15

improvements in both settings. These results support the usefulness of failure-centered precision restoration beyond a single dataset or model. Results for all 13 behavioral failure categories are provided in Appendix Tables 8 and 9.

Restoration also benefits non-interactive coding tasks. Table 3 evaluates TQ-Mix on four additional benchmarks (e.g., RepoBench, HumanEval, MBPP, and LiveCodeBench) for LLaMA-3-8B-Instruct. TQ-Mix achieves the best results among the quantized configurations on all five reported metrics, raising the mean score from 46.53 under TQ-3bit to 48.27, compared with 51.03 under BF16. These results provide complementary evidence that selective restoration can improve coding performance outside multi-turn agent execution, although recovery remains partial and these benchmarks do not directly measure execution-control failures. Detailed settings are provided in Appendix F.

5.4 Robustness of Failure-Centered Diagnosis

Layer selection and restoration precision jointly shape behavioral recovery. In Table 4a, restoring all five sensitive layers to BF16 costs an additional 1.00 bit on average, yet improves the resolved rate by only 0.2%. Conversely, assigning all five layers to 4 bits or restoring only Layer 3 to BF16 yields modest precision savings but increases behavioral failure by 8.0% and 4.7%, respectively. TQ-Mix thus offers a favorable trade-off between behavioral recovery and precision cost.

Sparse sensitivity and local-attention preference persist across quantization configurations. In Table 4b, only 2.5%–5.3% of heads are classified as sensitive across TQ-3bit, TQ-4bit, INT4, and FP8. Before quantization, these heads exhibit higher mean local attention scores (0.62–0.77) than long-range scores (0.09–0.22). The localization analysis thus consistently identifies a sparse, locally focused subset of heads, suggesting that this pattern is not specific to TurboQuant.

6 Conclusion

Fewer bits per token do not necessarily mean fewer resources per task. More bits in the right layers can mean less memory. In agentic coding, directly applying low-bit KV cache quantization can amplify behavioral failures, prolong interactions, and increase execution overhead. These failures also provide a signal for pinpointing sensitive layers, making behavioral diagnosis a practical guide to precision allocation. Our results show that selectively restoring precision in these layers offers a simple way to improve trajectory reliability while retaining substantial memory savings. Precision is therefore both a storage cost and a resource for reliable execution.

AI USE STATEMENT

We used generative AI tools to assist with translation and interpretation of results. We also used large language models as behavioral auditors to annotate agent trajectories, as described in Appendix B.2. We did not use generative AI tools to generate synthetic datasets or to clean or reformat datasets.

Additionally, we used generative AI tools to improve the readability and organization of the manuscript and to suggest titles or keywords. The authors reviewed the AI-assisted manuscript content and take responsibility for the final content of this work, including its text, analyses, claims, and research artifacts.

ETHICS STATEMENT

This work uses publicly available models, coding benchmarks, and open-source repositories, and does not involve human subjects or private data. Although the proposed method improves the efficiency and reliability of coding agents, these agents may still generate incorrect or insecure code. Practical deployment should therefore include sandboxing, testing, access control, and human oversight.

REPRODUCIBILITY STATEMENT

The main text specifies the evaluated models, KV cache quantization settings, benchmarks, and evaluation metrics. Appendix A documents the detailed experimental setup, including agent configurations, behavioral failure definitions, evaluation protocols, and sensitivity-guided localization and precision restoration procedures. Additional results across models, datasets, and quantizers are provided to support the reported findings.

REFERENCES

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172/.

Islem Bouzenia and Michael Pradel. Understanding software engineering agents: A study of thought-action-result trajectories, 2025. URL https://arxiv.org/abs/2506.18824.

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, 2025. URL https://arxiv.org/abs/2406.02069.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Neil Chowdhury, James Aung, Chan Jun Shern, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, Carlos E. Jimenez, John Yang, Leyton Ho, Tejal Patwardhan, Kevin Liu, and Aleksander Madry. Introducing SWE-bench Verified. OpenAI Blog, August 2024. URL https://openai.com/index/introducing-swe-bench-verified/. Updated February 24, 2025.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 35549–35562, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/98ed250b203d1ac6b24bbcf263e3d4a7-Paper-Conference.pdf.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 16344–16359. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/67d57c32e20fd0a7a302cb81d36e40d5-Paper-Conference.pdf.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. GPT3.int8(): 8-bit matrix multiplication for transformers at scale. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 30318–30332. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/c3ba4962c05c49636d4c6206a97e9c8a-Paper-Conference.pdf.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers, 2023. URL https://arxiv.org/abs/2210.17323.

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 B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 22975–22988, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/639a9a172c044fbb64175b5fad42e9a5-Paper-Conference.pdf.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 1270–1303. Curran Associates, Inc., 2024. doi: 10.52202/079017-0040. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/028fcbcf85435d39a40c4d61b42c99a4-Paper-Conference.pdf.

Naman Jain, Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/94074dd5a072d28ff75a76dabed43767-Paper-Conference.pdf.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/edac78c3e300629acfe6cbe9ca88fb84-Paper-Conference.pdf.

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*, SOSP '23, pp. 611–626, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702297. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Sangjun Lee, Seung-taek Woo, Jun-gyu Jin, Changhun Lee, and Eunhyeok Park. AMQ: Enabling AutoML for mixed-precision weight-only quantization of large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 35532–35550. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.1799. URL https://aclanthology.org/2025.emnlp-main.1799/.

Xing Li, Zeyu Xing, Yiming Li, Linping Qu, Hui-Ling Zhen, Yiwu Yao, Wulong Liu, Sinno Jialin Pan, and Mingxuan Yuan. KVTuner: Sensitivity-aware layer-wise mixed-precision KV cache quantization for efficient and nearly lossless LLM inference. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 36451–36485. PMLR, 2025. URL https://proceedings.mlr.press/v267/li25dd.html.

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. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 22947–22970. Curran Associates, Inc., 2024. doi: 10.52202/079017-0722. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/28ab418242603e0f7323e54185d19bde-Paper-Conference.pdf.

Chengxi Liao and Zeyi Wen. Channel-aware mixed-precision quantization for efficient long-context inference. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=yjr2jX41q0.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. In P. Gibbons, G. Pekhimenko, and C. De Sa (eds.), *Proceedings of Machine Learning and Systems*, volume 6, pp. 87–100, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/file/42a452cbafa9dd64e9ba4aa95cc1ef21-Paper-Conference.pdf.

Tianyang Liu, Canwen Xu, and Julian McAuley. Repobench: Benchmarking repository-level code auto-completion systems. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 47832–47850, 2024a. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/d191ba4c8923ed8fd8935b7c98658b5f-Paper-Conference.pdf.

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 *Forty-first International Conference on Machine Learning*, 2024b. URL https://openreview.net/forum?id=L057s2Rq8O.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91edff07232fb1b55a505a9e9f6c0ff3-Paper-Conference.pdf.

Oorja Majgaonkar, Zhiwei Fei, Xiang Li, Federica Sarro, and He Ye. Understanding code agent behaviour: An empirical study of success and failure trajectories, 2025. URL https://arxiv.org/abs/2511.00197.

Tural Mehtiyev and Wesley Assunção. Beyond resolution rates: Behavioral drivers of coding agent success and failure, 2026. URL https://arxiv.org/abs/2604.02547.

Priyam Sahoo, Gaurav Mittal, Xiaomin Li, Shengjie Ma, Benjamin Steenhoek, Pingping Lin, and Yu Hu. AgentLens: Revealing the lucky pass problem in SWE-Agent evaluation, 2026. URL https://arxiv.org/abs/2605.12925.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach

themselves to use tools. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 68539–68551. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/d842425e4bf79ba039352da0f658a906-Paper-Conference.pdf.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Gao Peng, Yu Qiao, and Ping Luo. OmniQuant: Omnidirectionally calibrated quantization for large language models. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 45472–45496, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/c6483c8a68083af3383f91ee0dc6db95-Paper-Conference.pdf.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.

Rui Shu, Chun Yong Chong, Xin Zhou, Yun Peng, Zihan Wu, Xu Han, Zeyang Zhuang, Guowen Yuan, and Yuan Wang. What resolve rate hides: Trajectory structure diagnostics for coding agents. *CoRR*, abs/2607.06184, 2026. doi: 10.48550/ARXIV.2607.06184. URL https://doi.org/10.48550/arXiv.2607.06184.

Donghyun Son, Euntae Choi, and Sungjoo Yoo. Nsnquant: A double normalization approach for calibration-free low-bit vector quantization of kv cache. 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, pp. 43124–43159. Curran Associates, Inc., 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/3d8ee933c215fbb7b4d1948ff4906299-Paper-Conference.pdf.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=OJd3ayDDoF.

Xingyao Wang, Valerie Chen, Heng Ji, and Graham Neubig. A rubric-supervised critic from sparse real-world outcomes, 2026. URL https://arxiv.org/abs/2603.03800.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying LLM-based software engineering agents, 2024. URL https://arxiv.org/abs/2407.01489.

Guangxuan Xiao, Ji Lin, Mickaël Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: Accurate and efficient post-training quantization for large language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 38087–38099. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/xiao23c.html.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 21875–21895, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/5e5fd18f863cbe6d8ae392a93fd271c9-Paper-Conference.pdf.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 50528–50652. Curran Associates, Inc., 2024. doi: 10.52202/079017-1601. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/5a7c947568c1b1328ccc5230172e1e7c-Paper-Conference.pdf.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.

Amir Zandieh, Majid Daliri, Majid Hadian, and Vahab Mirrokni. TurboQuant: Online vector quantization with near-optimal distortion rate. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=tO3ASKZlok.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. AutoCodeRover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2024, pp. 1592–1604, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400706127. doi: 10.1145/3650212.3680384. URL https://doi.org/10.1145/3650212.3680384.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang "Atlas" Wang, and Beidi Chen. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 34661–34710. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/6ceefa7b15572587b78ecfcebb2827f8-Paper-Conference.pdf.

Junchen Zhao, Ali Derakhshan, Jayden Hyman, Junhao Dong, Sangeetha Abdu Jyothi, and Ian Harris. CoopQ: Cooperative game inspired layerwise mixed precision quantization for LLMs. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 7566–7578. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.373. URL https://aclanthology.org/2026.findings-acl.373/.

APPENDIX OVERVIEW

The appendix details the experimental setup, behavioral analyses, and TQ-Mix implementation, and provides theoretical support and additional evaluations. The overview below links to each section and subsection.

Contents		Page
I. Evaluation Setup and Behavioral Analysis		
A. Agentic Coding Evaluation Setup		16
A.1	Model and Quantizer Selection	16
A.2	Benchmark Selection	16
A.3	Agent Environment and Task Evaluation	16
B. Behavioral Failure Analysis		17
B.1	Behavioral Failure Taxonomy	17
B.2	Behavioral Auditing and Cross-Auditor Consistency	18
B.3	Case Studies of Behavioral Failures	18
B.4	Behavioral Failures Across Execution Phases	18
B.5	Localization Stability Across Replay Windows	19
II. Method, Theory, and Generalization		
C. Implementation Details of TQ-Mix		20
C.1	Alternative Layer-Ranking Criteria	21
D. Theoretical Analysis of KV Cache Quantization and Restoration		22
D.1	Proof of Theorem 1	22
D.2	Perturbation Propagation Under Selective KV Restoration	24
E. Generalization Across Datasets and Models		27
E.1	Generalization to SWE-bench Multilingual	27
E.2	Generalization Across Models and Architectures	27
III. Auxiliary Evaluation and Attention Diagnostics		
F. Auxiliary Evaluation on Static Benchmarks		29
F.1	Coding Benchmarks	29
F.2	Coding Evaluation Metrics	30
F.3	Long-Context Evaluation on LongBench-E	30
G. Statistical Analysis and Attention Visualizations		31
G.1	Confidence Intervals for Layer Sensitivity Scores	31
G.2	Comparison of Attention Heads in Sensitive vs Non-Sensitive Layers	31

A AGENTIC CODING EVALUATION SETUP

A.1 MODEL AND QUANTIZER SELECTION

To rigorously evaluate the impact of KV cache quantization on agentic coding, we carefully select an underlying large language model (LLM) that balances robust reasoning capabilities with practical computational constraints.

Why Qwen3-Coder-30B-A3B-Instruct? Smaller models often lack the advanced planning, context aggregation, and tool-use capabilities required for long-horizon agentic coding tasks, whereas substantially larger models incur prohibitive inference costs. Moreover, our agentic coding setting requires a context window of nearly 128K, a capability that is not consistently supported by several more recent alternatives. Therefore, we adopt Qwen3-Coder-30B-A3B-Instruct as our primary agentic backbone. It provides a favorable balance between model capacity, long-context support, and inference efficiency, while also being **publicly** available and specifically optimized for coding tasks.

Why TurboQuant KV cache? For our evaluation, we treat the uncompressed `bfloat16` (BF16) KV cache as the full-precision reference. To study aggressive memory compression, we adopt TurboQuant (Zandieh et al., 2026) with 3-bit quantization as our primary compressed baseline. We favor TurboQuant over heuristic quantization approaches because of its theoretical guarantees and online applicability. In particular, its provable distortion bounds and data-oblivious design make it well suited to agentic coding settings, where the KV cache grows dynamically over long and potentially unpredictable interaction trajectories.

A.2 BENCHMARK SELECTION

In our primary setting, for evaluating agentic coding performance, we utilize the SWE-bench dataset series, specifically anchoring our analysis on SWE-bench Verified (Chowdhury et al., 2024). While the original SWE-bench provides a highly realistic simulation of real-world GitHub issue resolution, it is known to contain unresolvable issues, flaky tests, and underspecified requirements that can confound evaluation.

Why SWE-bench Verified? SWE-bench Verified provides a curated subset of SWE-bench in which problematic instances have been manually screened to improve task solvability and evaluation reliability. This makes it particularly suitable for our trajectory-level analysis, where we aim to isolate behavioral changes induced by KV cache quantization. By reducing failures caused by ambiguous tasks, unstable evaluation, or environment-related issues, SWE-bench Verified helps minimize dataset-level confounding factors. This allows behaviors such as repetitive loops or skipped verification steps to be more reliably associated with changes in KV cache precision.

A.3 AGENT ENVIRONMENT AND TASK EVALUATION

SWE-bench Verified is a human-validated subset of 500 instances from SWE-bench (Jimenez et al., 2024), a benchmark of real-world software engineering tasks drawn from issues and corresponding pull requests in open-source Python repositories, including `django`, `sympy`, and `scikit-learn`. The Verified subset was curated through expert review of issue clarity and test suitability, providing a more reliable basis for evaluating repository-level problem solving.

In our setup, each instance is executed in an isolated Docker environment with the repository checked out at the specified `base_commit`. Given the issue description (`problem_statement`), the agent interacts with the repository through shell commands and file-editing tools to inspect code, implement changes, and validate a candidate patch. These multi-turn interactions produce the trajectories used in our behavioral analysis.

Task success is evaluated by executing the benchmark tests after applying the generated patch. The `FAIL_TO_PASS` tests check whether the patch resolves failures associated with the issue, while the `PASS_TO_PASS` tests check whether previously passing tests remain successful. An instance is considered resolved only when all required tests in both groups pass. The benchmark's reference patch and evaluation test patch are withheld from the agent during trajectory generation.

Table 5: Definitions of the 13 behavioral failure categories used in our trajectory-level analysis of agentic coding.

Behavior Category	Definition
Improper tool use or setup	The agent misuses tools, commands, APIs, dependencies, or environment configurations, such as using incorrect command syntax, inappropriate tools, or malformed setup procedures.
Loop behavior	The agent repeatedly performs the same or substantially similar failed action, typically three or more times, without changing strategy or making meaningful progress.
Insufficient debugging	The agent fails to adequately investigate, isolate, or resolve observed errors, such as ignoring stack traces, proceeding while failures persist, or not reducing the failing case.
Incomplete implementation	The agent delivers unfinished, partial, non-functional, or non-runnable code, including cases with unresolved TODOs, stubs, missing components, or broken functionality.
Insufficient analysis	The agent acts without sufficiently inspecting relevant existing code, documentation, examples, logs, or contextual information that should inform the solution.
Other agent issue	Any other agent-side behavioral failure that is observable from the trace but not covered by the other categories.
Insufficient testing	The agent skips reasonable validation, testing, or sanity checks for non-trivial or risky code changes.
Did not follow instruction	The agent ignores or fails to comply with explicit user instructions, system constraints, or task requirements.
Misunderstood intention	The agent misunderstands the user's underlying goal or intent, leading it to pursue an incorrect objective or produce an output that does not address the requested task.
File management errors	The agent creates, modifies, deletes, or places files incorrectly, including writing to wrong paths, overwriting unintended files, creating unnecessary artifacts, or mismanaging project structure.
Insufficient clarification	The agent proceeds despite ambiguous or underspecified requirements, instead of asking necessary clarification questions before taking action.
Scope creep	The agent implements unrequested functionality, expands the task beyond the user's intent, or changes the scope without explicit approval.
Risky actions or permission issues	The agent performs potentially harmful or irreversible actions without explicit user permission, such as pushing to protected branches, deleting existing files, modifying credentials, or making unsafe repository changes.

This setting allows us to examine both final task success and the intermediate behaviours that determine execution cost. Human curation helps reduce ambiguity in task specifications and evaluation, while comparisons with BF16 under the same task instances, agent configuration, and execution budgets allow us to assess how KV-cache quantization changes behavioral failure rates and trajectory-level efficiency.

B Behavioral Failure Analysis

B.1 Behavioral Failure Taxonomy

We adopt the behavioral failure taxonomy from prior work on critic rubrics for coding agents (Wang et al., 2026). This taxonomy defines behavioral failures as segment-level patterns observable in the agent's interaction trace. A segment corresponds to a user-initiated unit of work, beginning with a user request and ending when the agent completes its actions. Each failure category is marked as present only when supported by concrete trace evidence, such as user messages, tool calls, command outputs, code edits, error logs, or final responses. When evidence is insufficient, the category is conservatively labeled as absent. This annotation protocol enables us to examine how KV cache quantization affects intermediate agent behaviors, complementing the final resolved rate. Table 5 provides detailed definitions of all 13 behavioral failure categories.

B.2 Behavioral Auditing and Cross-Auditor Consistency

To reduce potential model-family-related bias, we use Claude Opus 4.7 via API as the primary behavioral auditor rather than a Qwen-series model, since Qwen is our primary agent backbone. Following the prompt templates of Wang et al. (2026), we annotate interaction trajectories per configuration under BF16 and KV cache quantization. To assess whether the findings depend on the choice of auditor, we use Gemini-flash 3.1 to independently annotate the same trajectories. Both auditors identify consistent behavioral degradation patterns, with a Pearson correlation of $r \approx 0.92$ between their per-category changes in failure rates relative to BF16. This agreement supports the consistency of our findings across auditors.

Auditor protocol. Each auditor receives only the agent trace, without model identifiers, KV cache precision settings, or benchmark evaluation results. Annotations are returned as schema-constrained Boolean labels. We set the temperature to 0 and retain the default values for other decoding parameters.

Table 6 shows that the two models identify the same three dominant failure categories and consistent recovery patterns. Their estimates of per-category failure-rate changes are strongly correlated, with Pearson's $r \approx 0.92$ for degradation (TQ-3bit minus BF16) and $r \approx 0.88$ for recovery (TQ-3bit minus TQ-Mix). We quantify this cross-auditor consistency using Pearson's correlation coefficient across behavioral failure categories:

$$r = \frac{\sum_{k=1}^{K}(x_k - \bar{x})(y_k - \bar{y})}{\sqrt{\sum_{k=1}^{K}(x_k - \bar{x})^2}\sqrt{\sum_{k=1}^{K}(y_k - \bar{y})^2}}, \tag{4}$$

where x_k and y_k denote the effects estimated by Claude and Gemini, respectively, for behavior category k; $\bar{x}$ and $\bar{y}$ are their means across the $K = 10$ non-zero categories. For behavioral failure, the effect is the rate difference between TQ-3bit and BF16. For recovery, it is the rate difference between TQ-3bit and TQ-Mix.

B.3 Case Studies of Behavioral Failures

In addition to the loop-behavior case study in the main text, this appendix presents two further case studies: improper tool use or setup in `django-11477` and insufficient debugging in `sympy-18189`.

Figure 7 illustrates the first case. Under BF16, the agent responds to execution errors by installing missing dependencies and correcting the test invocation, successfully running the tests within Turns 10–13. Under TQ-3bit, the agent responds to a missing-`pytest` error by switching test runners and attempting incorrect file or module paths during Turns 20–24, leaving the dependency issue unresolved. This case illustrates how ineffective responses to execution feedback can prolong tool use without enabling verification.

Figure 8 presents the second case study, illustrating insufficient debugging. Under BF16, the agent narrows the issue to the relevant function at Turn 23 and adds a diagnostic script to inspect intermediate values during Turns 24–25. These observations help it identify the root cause at Turn 26, and the subsequent repair passes verification at Turn 38. Under TQ-3bit, the agent confirms the failure at Turn 47 but repeatedly edits the code and reruns tests during Turns 48–64 without adding diagnostic instrumentation. Across three attempts, it claims to have identified the issue, yet each diagnosis is incorrect, and verification still fails at Turn 65. This case illustrates how repeated edits without sufficient investigation can prolong debugging while leaving the underlying error unresolved.

B.4 Behavioral Failures Across Execution Phases

Phase-wise analysis of behavioral failures. We further analyze where the three behavioral failure categories highlighted in the main text occur across stages of task execution. Figure 9 shows distinct phase distributions across categories. In our primary setting, improper tool use or setup is most concentrated in *Test Creation* (P4, 45.3%), followed by *Verification* (P7, 28.0%). Loop behavior and insufficient debugging are most concentrated in *Verification* (47.7% and 42.9%, respectively), with substantial shares also occurring during *Fix Implementation* (P6, 24.2% and 25.0%). These

Table 6: Behavioral failure rates annotated by two independent auditors on SWE-bench Verified across all 13 categories. Parentheses report percentage-point changes relative to TQ-3bit; negative values indicate reductions in failure rates.

Behavior	Auditor	BF16	TQ-3bit	TQ-Mix
Improper Tool Use or Setup	Claude	38%	67%	43% (−24 pp)
	Gemini	40%	60%	38% (−22 pp)
Loop Behavior	Claude	15%	41%	20% (−21 pp)
	Gemini	16%	44%	17% (−27 pp)
Insufficient Debugging	Claude	48%	66%	48% (−18 pp)
	Gemini	47%	62%	48% (−14 pp)
Incomplete Implementation	Claude	42%	57%	51% (−6 pp)
	Gemini	34%	50%	46% (−4 pp)
Insufficient Analysis	Claude	39%	50%	46% (−4 pp)
	Gemini	35%	52%	37% (−15 pp)
Other Agent Issue	Claude	6%	15%	8% (−7 pp)
	Gemini	7%	16%	7% (−9 pp)
Did Not Follow Instruction	Claude	21%	24%	23% (−1 pp)
	Gemini	24%	23%	20% (−3 pp)
Insufficient Testing	Claude	90%	95%	93% (−2 pp)
	Gemini	95%	98%	96% (−2 pp)
File Management Errors	Claude	95%	96%	95% (−1 pp)
	Gemini	96%	98%	96% (−2 pp)
Misunderstood Intention	Claude	6%	8%	5% (−3 pp)
	Gemini	7%	9%	5% (−4 pp)
Insufficient Clarification	Claude	0%	0%	0% (0 pp)
	Gemini	0%	0%	0% (0 pp)
Scope Creep	Claude	0%	0%	0% (0 pp)
	Gemini	0%	0%	0% (0 pp)
Risky Actions or Permission	Claude	0%	0%	0% (0 pp)
	Gemini	0%	0%	0% (0 pp)

category-specific peaks persist across the additional evaluation settings, while the early phases (P1–P3) account for relatively small shares of the failures. The observed distributions place these failures primarily in stages that require tool use, code modification, and responses to execution feedback, further characterizing where execution-control difficulties arise in quantized trajectories.

B.5 Localization Stability Across Replay Windows

Layer selection is stable across moderate window lengths. In Table 7, all tested window lengths $w \in \{18, 20, 24, 26, 28\}$ select the same five layers as the default $w = 24$. Only the extremely short window, $w = 2$, changes three of the five selected layers. Thus, localization is stable across the tested moderate windows rather than tied to a single window choice. The different selection at $w = 2$ suggests that very short windows may provide insufficient evidence for stable sensitivity estimation; it does not by itself establish that the resulting allocation performs worse.

Table 7: Sensitive-layer selections across replay-window lengths on SWE-bench Verified. Blue shading marks the default replay window.

Replay window	Selected layers
$w = 24$	3, 5, 16, 28, 40
$w \in \{18, 20\}$	3, 5, 16, 28, 40
$w \in \{26, 28\}$	3, 5, 16, 28, 40
$w = 2$	3, 5, 11, 17, 32

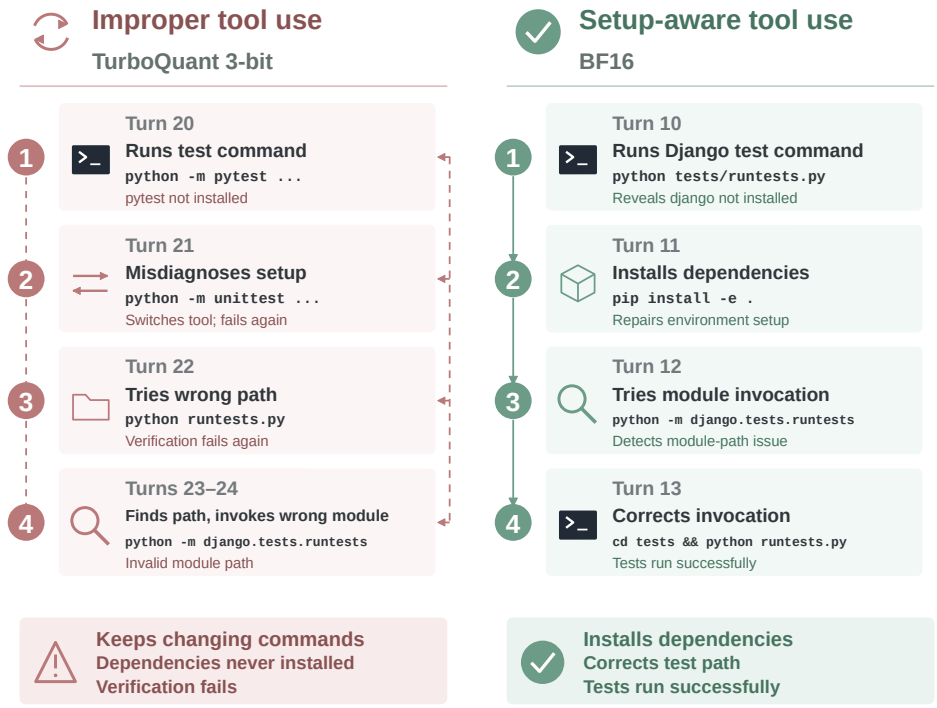


Figure 7: A case study of improper tool use or setup. TQ-3bit repeatedly changes test commands without resolving the dependency issue, whereas BF16 installs dependencies and corrects the invocation to run the tests successfully. Turn numbers refer to positions within each configuration's trajectory.

C Implementation Details of TQ-Mix

This section specifies the calibration and precision-allocation procedure used by TQ-Mix.

Calibration and evaluation pipeline. We use disjoint calibration and test sets. We sample $N_{\text{calib}} = 50$ calibration instances and execute them under TQ-3bit to collect calibration trajectories. The behavioral auditor identifies segments corresponding to the three execution-control failure categories studied in the main text. For each annotated failure segment, we construct a failure-onset window $[t^\star, t^\star + w)$ and a post-failure window $[t^{\text{end}}, t^{\text{end}} + w)$, with $w = 24$. Following Section 4.1, we replay the same recorded tokens under BF16 and controlled head-wise KV cache perturbations, holding queries fixed and retaining BF16 elsewhere. We average attention KL over query positions and calibration examples, assign equal weight to the two window types, and average across heads to obtain the layer-ranking scores $\{S_\ell\}_{\ell=1}^{L}$. Head-output cosine distance is retained as a complementary diagnostic. Let μ_S and σ_S denote the mean and standard deviation of the layer scores, and define the standardized score

$$z_\ell = \frac{S_\ell - \mu_S}{\sigma_S}. \tag{5}$$

Layers more than three standard deviations above the mean form the extreme-outlier tier

$$\mathcal{R}_{\text{BF16}} = \{\ell : z_\ell > 3\}. \tag{6}$$

After identifying the BF16 tier, we recompute the mean and standard deviation of the sensitivity scores over the remaining layers, denoted by $\mathcal{C} = \{1, \ldots, L\} \setminus \mathcal{R}_{\text{BF16}}$. Let $\mu_{\mathcal{C}}$ and $\sigma_{\mathcal{C}}$ denote these statistics. Layers more than two standard deviations above the remaining-layer mean form the 4-bit tier:

$$\mathcal{R}_4 = \{\ell \in \mathcal{C} : S_\ell > \mu_{\mathcal{C}} + 2\sigma_{\mathcal{C}}\}. \tag{7}$$

All remaining layers retain 3-bit precision.

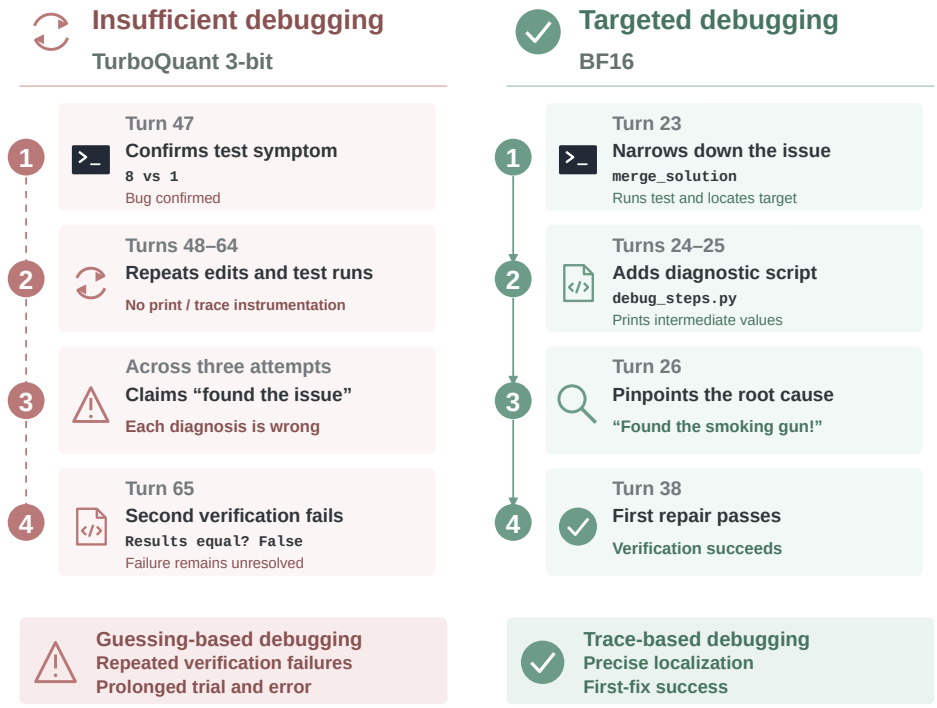


Figure 8: A case study of insufficient debugging. TQ-3bit repeatedly edits code and reruns tests without collecting diagnostic evidence. BF16 uses a diagnostic script to inspect intermediate values, identify the root cause, and produce a repair that passes verification.

The resulting per-layer KV-cache precision is

$$b_\ell = \begin{cases} 16, & \ell \in \mathcal{R}_{\mathrm{BF16}}, \\ 4, & \ell \in \mathcal{R}_4, \\ 3, & \text{otherwise}. \end{cases} \tag{8}$$

For Qwen3-Coder-30B-A3B-InstructWith $L = 48$, the rule yields $\mathcal{R}_{\mathrm{BF16}} = \{3\}$ and $\mathcal{R}_4 = \{5, 16, 28, 40\}$. The nominal average precision is therefore

$$\bar{b} = \frac{16 + 4 \times 4 + 43 \times 3}{48} = 3.354 \approx 3.35 \text{ bits}. \tag{9}$$

Notation. z_ℓ is its standardized value across layers. The sets $\mathcal{R}_{\mathrm{BF16}}$ and $\mathcal{R}_4$ contain the layers assigned BF16 and 4-bit KV caches, respectively, while all remaining layers use 3-bit caches. The policy is calibrated once for a model and then fixed for every evaluation instance; task outcomes are not used in either scoring or allocation.

C.1 Alternative Layer-Ranking Criteria

We compare failure-centered layer selection with three alternative criteria: reconstruction error, activation magnitude, and perplexity sensitivity. All criteria use the same calibration trajectories and a common precision-allocation scheme.

Reconstruction error(KV Tuner). We quantize one layer's KV cache at a time to 3 bits while retaining BF16 KV caches in all other layers.We compute the normalized mean-squared error between the target layer's quantized and BF16 attention outputs. Layers with larger reconstruction errors receive higher restoration priority.

$$S_\ell^{\mathrm{rec}} = \frac{\sum_{i,t} \|\widetilde{\mathbf{o}}_{\ell,i,t} - \mathbf{o}_{\ell,i,t}\|_2^2}{\sum_{i,t} \|\mathbf{o}_{\ell,i,t}\|_2^2 + \epsilon}. \tag{10}$$

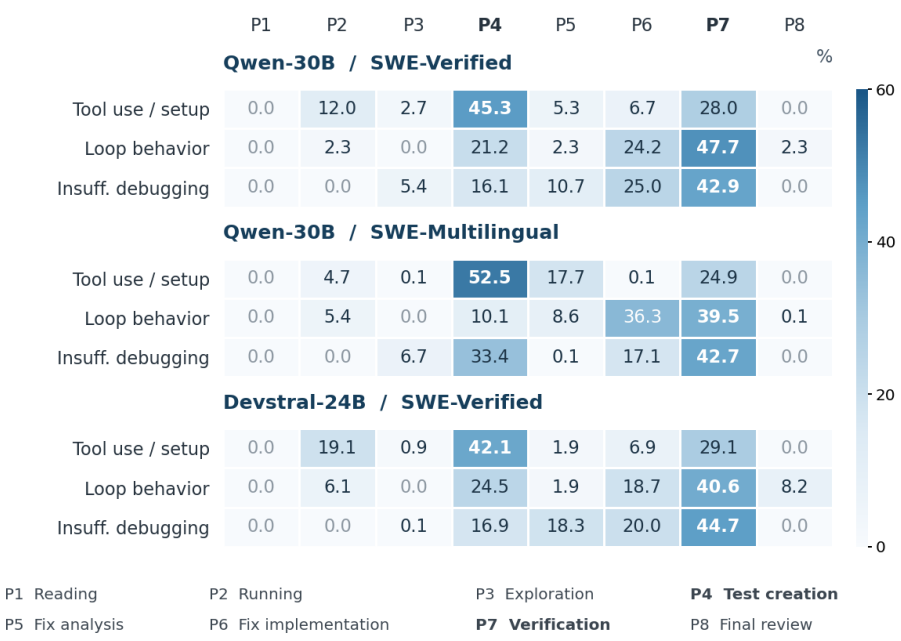


Figure 9: Phase-wise distribution of behavioral failures under quantization.

Activation magnitude(CoopQ). We collect each layer's key and value activations under BF16 and compute their root-mean-square magnitude over valid calibration tokens and activation dimensions. Layers with larger activation magnitudes receive higher restoration priority.

$$S_\ell^{\text{act}} = \sqrt{\frac{\sum_i \left(\|\mathbf{K}_{\ell,i}\|_F^2 + \|\mathbf{V}_{\ell,i}\|_F^2\right)}{\sum_i \left(\text{numel}(\mathbf{K}_{\ell,i}) + \text{numel}(\mathbf{V}_{\ell,i})\right)}}. \tag{11}$$

Perplexity sensitivity(AMQ). We quantize each layer's KV cache individually to 3 bits while retaining BF16 KV caches elsewhere. We measure the increase in perplexity relative to BF16. Layers that cause larger perplexity increases receive higher restoration priority.

$$S_\ell^{\text{ppl}} = \text{PPL}\left(\mathcal{M}_\ell^{\text{Q}}; \mathcal{D}_{\text{calib}}\right) - \text{PPL}(\mathcal{M}_{\text{BF16}}; \mathcal{D}_{\text{calib}})\,. \tag{12}$$

D Theoretical Analysis of KV Cache Quantization and Restoration

This section provides theoretical support for the attention diagnosis and precision restoration studied in the main text. These results motivate our diagnostic measurements and restoration strategy under the stated assumptions. We first prove Theorem 1, separating head-level output error into contributions from attention-routing changes and value quantization. We then derive a layer-wise perturbation bound that illustrates how selective BF16 restoration can reduce an upper bound on representation deviation. These results motivate our diagnostic measurements and restoration strategy under the stated assumptions.

D.1 Proof of Theorem 1

Proof of Theorem 1. We prove the result by decomposing the attention-output difference into two parts: an attention-weight perturbation term and a value-quantization term.

Algorithm 1 TQ-Mix: Sensitivity-Guided KV Cache Precision Restoration

Require: Calibration pool $\mathcal{D}$ disjoint from the test set; BF16 reference model $\mathcal{M}_{\text{full}}$; TQ-3bit model $\mathcal{M}_{\text{quant}}$; behavioral auditor $\mathcal{A}$
Ensure: TQ-Mix model $\mathcal{M}_{\text{mix}}$

$N_{\text{calib}} \leftarrow 50$, $w \leftarrow 24$
$\mathcal{D}_{\text{calib}} \leftarrow \textsc{RandomSample}(\mathcal{D}, N_{\text{calib}})$
$\mathcal{T}_{\text{quant}} \leftarrow \mathcal{M}_{\text{quant}}(\mathcal{D}_{\text{calib}})$
// Locate the three execution-control failure categories
$\mathcal{B} \leftarrow \mathcal{A}(\mathcal{T}_{\text{quant}})$
// Each annotation records the trajectory, category, and token span
$\mathcal{W}_{\text{onset}} \leftarrow \emptyset$, $\mathcal{W}_{\text{post}} \leftarrow \emptyset$
for each annotated failure segment $(\tau, t^\star, t^{\text{end}})$ in $\mathcal{B}$ **do**
  Add $(\tau, [t^\star, t^\star + w))$ to $\mathcal{W}_{\text{onset}}$
  Add $(\tau, [t^{\text{end}}, t^{\text{end}} + w))$ to $\mathcal{W}_{\text{post}}$
end for
// Replay recorded tokens; do not generate new continuations
for each layer $\ell = 1, \ldots, L$ **do**
  for each attention head $h = 1, \ldots, H_\ell$ **do**
    Replay both window types using $\mathcal{M}_{\text{full}}$
    Quantize only the target head's KV cache, keeping queries fixed and retaining BF16 elsewhere
    Measure attention KL and head-output cosine distance as in Section 4.1
    Compute mean attention KL $s_{\ell,h}^{\text{onset}}$ and $s_{\ell,h}^{\text{post}}$ over query positions and calibration examples
    $s_{\ell,h} \leftarrow \frac{1}{2}\left(s_{\ell,h}^{\text{onset}} + s_{\ell,h}^{\text{post}}\right)$
  end for
  $S_\ell \leftarrow \frac{1}{H_\ell} \sum_{h=1}^{H_\ell} s_{\ell,h}$
end for
// Allocate precision using attention-KL scores only
Compute the mean μ_S and standard deviation σ_S of $\{S_\ell\}_{\ell=1}^{L}$
$z_\ell \leftarrow (S_\ell - \mu_S)/\sigma_S$ for each layer ℓ
$\mathcal{R}_{\text{BF16}} \leftarrow \{\ell : z_\ell > 3\}$
$\mathcal{C} \leftarrow \{1, \ldots, L\} \setminus \mathcal{R}_{\text{BF16}}$
$\mu_{\mathcal{C}} \leftarrow \text{mean}_{\ell \in \mathcal{C}}\, S_\ell, \quad \sigma_{\mathcal{C}} \leftarrow \sqrt{\frac{1}{|\mathcal{C}|} \sum_{\ell \in \mathcal{C}} (S_\ell - \mu_{\mathcal{C}})^2}$
$\mathcal{R}_4 \leftarrow \{\ell \in \mathcal{C} : S_\ell > \mu_{\mathcal{C}} + 2\sigma_{\mathcal{C}}\}$
$\mathcal{M}_{\text{mix}} \leftarrow \mathcal{M}_{\text{quant}}$
for each layer $\ell \in \mathcal{R}_4$ **do**
  $\textsc{SetKVPrecision}(\mathcal{M}_{\text{mix}}, \ell, 4)$
end for
for each layer $\ell \in \mathcal{R}_{\text{BF16}}$ **do**
  $\textsc{SetKVPrecision}(\mathcal{M}_{\text{mix}}, \ell, \text{BF16})$
end for
return $\mathcal{M}_{\text{mix}}$

Recall that

$$o = \sum_j p_j v_j, \qquad \tilde{o} = \sum_j \tilde{p}_j \tilde{v}_j.$$

Adding and subtracting the mixed term $\sum_j \tilde{p}_j v_j$, we obtain

$$o - \tilde{o} = \sum_j p_j v_j - \sum_j \tilde{p}_j \tilde{v}_j \tag{13}$$

$$= \sum_j (p_j - \tilde{p}_j) v_j + \sum_j \tilde{p}_j (v_j - \tilde{v}_j). \tag{14}$$

Therefore, by the triangle inequality,

$$\|o - \tilde{o}\|_2 \leq \left\| \sum_j (p_j - \tilde{p}_j) v_j \right\|_2 + \left\| \sum_j \tilde{p}_j (v_j - \tilde{v}_j) \right\|_2 . \tag{15}$$

We first bound the attention-weight perturbation term. Using the triangle inequality and the assumption $\|v_j\|_2 \leq M$ for all j, we have

$$\left\| \sum_j (p_j - \tilde{p}_j) v_j \right\|_2 \leq \sum_j |p_j - \tilde{p}_j| \|v_j\|_2 \tag{16}$$

$$\leq M \sum_j |p_j - \tilde{p}_j| \tag{17}$$

$$= M \|p - \tilde{p}\|_1 . \tag{18}$$

By Pinsker's inequality,

$$\|p - \tilde{p}\|_1 \leq \sqrt{2 D_{\mathrm{KL}}(p \| \tilde{p})} . \tag{19}$$

Combining Eq. 18 and Eq. 19 gives

$$\left\| \sum_j (p_j - \tilde{p}_j) v_j \right\|_2 \leq M \sqrt{2 D_{\mathrm{KL}}(p \| \tilde{p})} . \tag{20}$$

We next bound the value-quantization term. Again by the triangle inequality,

$$\left\| \sum_j \tilde{p}_j (v_j - \tilde{v}_j) \right\|_2 \leq \sum_j \tilde{p}_j \|v_j - \tilde{v}_j\|_2 , \tag{21}$$

where we used that $\tilde{p}_j \geq 0$ for all j.

Substituting Eq. 20 and Eq. 21 into Eq. 15, we obtain

$$\|o - \tilde{o}\|_2 \leq M \sqrt{2 D_{\mathrm{KL}}(p \| \tilde{p})} + \sum_j \tilde{p}_j \|v_j - \tilde{v}_j\|_2 , \tag{22}$$

which proves Eq. 3.

If, in addition, $\|v_j - \tilde{v}_j\|_2 \leq \eta_V$ for all j, then

$$\sum_j \tilde{p}_j \|v_j - \tilde{v}_j\|_2 \leq \sum_j \tilde{p}_j \eta_V \tag{23}$$

$$= \eta_V \sum_j \tilde{p}_j \tag{24}$$

$$= \eta_V , \tag{25}$$

because $\tilde{p}$ is an attention distribution and hence $\sum_j \tilde{p}_j = 1$. Therefore,

$$\|o - \tilde{o}\|_2 \leq M \sqrt{2 D_{\mathrm{KL}}(p \| \tilde{p})} + \eta_V , \tag{26}$$

which proves the simplified bound. □

D.2 Perturbation Propagation Under Selective KV Restoration

We next examine how layer-local KV cache perturbations propagate through the model. Under Lipschitz and bounded-perturbation assumptions, we derive an upper bound on the final representation deviation and show how selective BF16 restoration removes the corresponding local perturbation terms from this bound. Here, restoration changes only the KV cache precision of the selected layers; model weights and non-KV computations remain unchanged.

Theorem 2 (Layer-local KV perturbations motivate selective restoration)**.** *Let F_ℓ be the BF16 Transformer block at layer ℓ, and let $\widetilde{F}_\ell$ be the corresponding block using quantized K/V caches. For notational simplicity, we write the block input as u, but u should be understood as the augmented decoding state, including the current hidden representation and the relevant cached K/V states under a fixed replay sequence.*

Let $\mathcal{R}$ be the set of layers whose K/V caches are restored to BF16. Define the mixed-precision block

$$G_\ell = \begin{cases} F_\ell, & \ell \in \mathcal{R}, \\ \widetilde{F}_\ell, & \ell \notin \mathcal{R}. \end{cases}$$

Let

$$h_\ell = F_\ell(h_{\ell-1}), \qquad h_\ell^{\mathcal{R}} = G_\ell(h_{\ell-1}^{\mathcal{R}})$$

be the full-BF16 and selectively restored hidden-state recursions, with the same input

$$h_0 = h_0^{\mathcal{R}}.$$

Assume that each BF16 block F_ℓ is ρ_ℓ-Lipschitz, i.e.,

$$\|F_\ell(x) - F_\ell(y)\|_2 \le \rho_\ell \|x - y\|_2,$$

and that the local KV-quantization perturbation satisfies

$$\sup_u \|F_\ell(u) - \widetilde{F}_\ell(u)\|_2 \le \delta_\ell.$$

Define the restoration-dependent perturbation term

$$\delta_\ell^{\mathcal{R}} = \begin{cases} 0, & \ell \in \mathcal{R}, \\ \delta_\ell, & \ell \notin \mathcal{R}. \end{cases}$$

Then the final representation deviation satisfies

$$\begin{aligned} \|h_L - h_L^{\mathcal{R}}\|_2 &\le \sum_{\ell=1}^{L} \left(\prod_{m=\ell+1}^{L} \rho_m \right) \delta_\ell^{\mathcal{R}} \\ &= \sum_{\ell \notin \mathcal{R}} \left(\prod_{m=\ell+1}^{L} \rho_m \right) \delta_\ell. \end{aligned}$$

Here the empty product is defined as 1.

Proof. Let

$$e_\ell = \|h_\ell - h_\ell^{\mathcal{R}}\|_2.$$

Since both recursions start from the same input, we have

$$e_0 = 0.$$

For layer ℓ, we add and subtract $F_\ell(h_{\ell-1}^{\mathcal{R}})$:

$$\begin{aligned} e_\ell &= \|F_\ell(h_{\ell-1}) - G_\ell(h_{\ell-1}^{\mathcal{R}})\|_2 && (27) \\ &\le \|F_\ell(h_{\ell-1}) - F_\ell(h_{\ell-1}^{\mathcal{R}})\|_2 && (28) \\ &\quad + \|F_\ell(h_{\ell-1}^{\mathcal{R}}) - G_\ell(h_{\ell-1}^{\mathcal{R}})\|_2. && (29) \end{aligned}$$

The first term is bounded by the Lipschitz assumption:

$$\|F_\ell(h_{\ell-1}) - F_\ell(h_{\ell-1}^{\mathcal{R}})\|_2 \le \rho_\ell e_{\ell-1}.$$

For the second term, there are two cases. If $\ell \in \mathcal{R}$, then $G_\ell = F_\ell$, and therefore

$$\|F_\ell(h_{\ell-1}^{\mathcal{R}}) - G_\ell(h_{\ell-1}^{\mathcal{R}})\|_2 = 0.$$

If $\ell \notin \mathcal{R}$, then $G_\ell = \widetilde{F}_\ell$, and by the local perturbation assumption,

$$\begin{aligned}\|F_\ell(h_{\ell-1}^{\mathcal{R}}) - G_\ell(h_{\ell-1}^{\mathcal{R}})\|_2 &= \|F_\ell(h_{\ell-1}^{\mathcal{R}}) - \widetilde{F}_\ell(h_{\ell-1}^{\mathcal{R}})\|_2 \\ &\le \delta_\ell.\end{aligned}$$

Thus, in both cases,

$$e_\ell \le \rho_\ell e_{\ell-1} + \delta_\ell^{\mathcal{R}}.$$

Unrolling this recursion gives

$$e_L \le \rho_L e_{L-1} + \delta_L^{\mathcal{R}}.$$

Applying the same inequality to e_{L-1},

$$e_L \le \rho_L \left(\rho_{L-1} e_{L-2} + \delta_{L-1}^{\mathcal{R}}\right) + \delta_L^{\mathcal{R}}.$$

Continuing recursively and using $e_0 = 0$, we obtain

$$e_L \le \sum_{\ell=1}^{L} \left(\prod_{m=\ell+1}^{L} \rho_m \right) \delta_\ell^{\mathcal{R}}.$$

By the definition of $\delta_\ell^{\mathcal{R}}$, all restored layers have zero perturbation contribution. Therefore,

$$e_L \le \sum_{\ell \notin \mathcal{R}} \left(\prod_{m=\ell+1}^{L} \rho_m \right) \delta_\ell.$$

Since $e_L = \|h_L - h_L^{\mathcal{R}}\|_2$, this proves the theorem. □

Interpretation. Theorem 2 shows that each non-restored layer contributes a layer-local KV-quantization perturbation term. The contribution of layer ℓ is

$$\left(\prod_{m=\ell+1}^{L} \rho_m \right) \delta_\ell,$$

where δ_ℓ measures the local perturbation caused by quantizing the K/V cache of layer ℓ, and the product of Lipschitz constants captures how this perturbation can propagate through subsequent layers.

Restoring a layer to BF16 removes its corresponding local perturbation term from the bound:

$$\ell \in \mathcal{R} \quad \Rightarrow \quad \delta_\ell^{\mathcal{R}} = 0.$$

Therefore, if a small number of layers have large local perturbations, selectively restoring those layers can reduce the dominant terms in the representation-deviation bound while keeping most other K/V caches quantized.

In practice, the true δ_ℓ and the propagation weights

$$\prod_{m=\ell+1}^{L} \rho_m$$

are difficult to estimate in large agentic coding models. We therefore use the empirical attention-routing score S_ℓ as a proxy for layer-local KV perturbation, while using head-output cosine distance as an independent check that the routing perturbation changes the information written to the residual stream.This gives a practical memory-efficient approximation to removing the largest layer-local KV-quantization perturbations.

E GENERALIZATION ACROSS DATASETS AND MODELS

This section examines whether the behavioral degradation and recovery patterns extend across datasets, model scales, and architectures. Unless stated otherwise, we use the same OpenHands–vLLM pipeline and behavioral auditing protocol as in the primary setting.

E.1 GENERALIZATION TO SWE-BENCH MULTILINGUAL

We further evaluate Qwen3-Coder-30B-A3B-Instruct on SWE-bench Multilingual to assess whether the behavioral degradation and recovery patterns observed on SWE-bench Verified extend to other programming languages, including Java, C, and C++. We retain the same OpenHands agent configuration, vLLM inference setup, and behavioral auditing protocol, using Docker environments for task execution. Table 8 reports failure rates across all 13 behavioral categories and the resolved rate.

Table 8: Behavioral and task-level results on SWE-bench Multilingual. Parentheses report the percentage-point change relative to TQ-3bit.

Behavior category	**BF16**	**TQ-3bit**	**TQ-Bd**	**TQ-Mix**
Improper Tool Use or Setup ↓	58%	83%	79% (−4 pp)	61% (−22 pp)
Loop Behavior ↓	28%	52%	51% (−1 pp)	34% (−18 pp)
Insufficient Debugging ↓	67%	84%	79% (−5 pp)	66% (−18 pp)
Incomplete Implementation ↓	57%	71%	65% (−6 pp)	63% (−8 pp)
Insufficient Analysis ↓	48%	63%	59% (−4 pp)	58% (−5 pp)
Other Agent Issue ↓	9%	21%	19% (−2 pp)	12% (−9 pp)
Did Not Follow Instruction ↓	22%	31%	31% (+0 pp)	29% (−2 pp)
Insufficient Testing ↓	90%	98%	96% (−2 pp)	96% (−2 pp)
File Management Errors ↓	95%	99%	99% (+0 pp)	97% (−2 pp)
Misunderstood Intention ↓	7%	13%	12% (−1 pp)	9% (−4 pp)
Insufficient Clarification ↓	0%	0%	0% (+0 pp)	0% (+0 pp)
Scope Creep ↓	0%	0%	0% (+0 pp)	0% (+0 pp)
Risky Actions or Permission ↓	0%	0%	0% (+0 pp)	0% (+0 pp)
Resolved Rate ↑	38.6%	33.3%	34.6% (+1.3 pp)	37.3% (+4.0 pp)

TQ-3bit produces the largest increases in improper tool use or setup, loop behavior, and insufficient debugging relative to BF16. TQ-Bd provides limited recovery, whereas TQ-Mix reduces these failure rates by 22, 18, and 18 percentage points, respectively, relative to TQ-3bit. At the task level, TQ-Mix raises the resolved rate from 33.3% to 37.3%, narrowing the gap to BF16 (38.6%) to 1.3 percentage points. These results extend the behavioral degradation and recovery patterns observed in the primary setting to additional programming languages and codebases.

E.2 GENERALIZATION ACROSS MODELS AND ARCHITECTURES

Behavioral recovery persists across Devstral model scales. We extend the evaluation to Devstral-Small-2-24B and Devstral-2-123B on SWE-bench Verified to assess whether the behavioral degradation and sensitivity-guided recovery patterns persist across model scales within the Devstral family. Table 9 provides the full breakdown of all 13 behavioral failure categories for the 24B model, while Table 10 compares both scales on the three execution-control failure categories highlighted in the main text and the resolved rate.

Table 9: Behavioral failure rates and resolved rate for Devstral-Small-2-24B on SWE-bench Verified. Parentheses report percentage-point changes relative to TQ-3bit. Lower failure rates and higher resolved rates are better.

Behavior category	**BF16**	**TQ-3bit**	**TQ-Bd**	**TQ-Mix**
Improper Tool Use or Setup ↓	41%	68%	63% (−5 pp)	45% (−23 pp)
Loop Behavior ↓	12%	38%	37% (−1 pp)	18% (−20 pp)
Insufficient Debugging ↓	51%	71%	65% (−6 pp)	52% (−19 pp)
Incomplete Implementation ↓	52%	66%	61% (−5 pp)	57% (−9 pp)
Insufficient Analysis ↓	43%	59%	55% (−4 pp)	52% (−7 pp)
Other Agent Issue ↓	4%	16%	14% (−2 pp)	7% (−9 pp)
Did Not Follow Instruction ↓	22%	31%	31% (+0 pp)	29% (−2 pp)
Insufficient Testing ↓	92%	99%	97% (−2 pp)	95% (−4 pp)
File Management Errors ↓	94%	98%	99% (+1 pp)	96% (−2 pp)
Misunderstood Intention ↓	5%	12%	10% (−2 pp)	8% (−4 pp)
Insufficient Clarification ↓	0%	0%	0% (+0 pp)	0% (+0 pp)
Scope Creep ↓	0%	0%	0% (+0 pp)	0% (+0 pp)
Risky Actions or Permission ↓	0%	0%	0% (+0 pp)	0% (+0 pp)
Resolved Rate ↑	66.2%	62.6%	63.4% (+0.8 pp)	65.8% (+3.2 pp)

Table 9 shows that TQ-3bit increases behavioral failure categories with nonzero occurrence rates, with the largest increases in improper tool use or setup, loop behavior, and insufficient debugging. TQ-Mix reduces these three failure rates by 23, 20, and 19 percentage points, respectively, relative to TQ-3bit, compared with reductions of 5, 1, and 6 points under TQ-Bd. It also lowers failure rates across the other seven nonzero categories, although recovery remains partial. At the task level, TQ-Mix raises the resolved rate from 62.6% to 65.8%, approaching BF16's 66.2%. These results extend the main findings to Devstral-Small-2-24B, showing substantial recovery of execution control alongside improved task resolution.

Table 10: Behavioral failure rates and resolved rates across two Devstral model scales on SWE-bench Verified. Parentheses report percentage-point changes relative to TQ-3bit. Lower failure rates and higher resolved rates are better.

Model	**Behavior**	**BF16**	**TQ-3bit**	**TQ-Bd**	**TQ-Mix**
24B	Improper Tool Use	41%	68%	63% (−5 pp)	45% (−23 pp)
	Loop Behavior	12%	38%	37% (−1 pp)	18% (−20 pp)
	Insufficient Debugging	51%	71%	65% (−6 pp)	52% (−19 pp)
	Resolved Rate	66.2%	62.6%	63.4% (+0.8 pp)	65.8% (+3.2 pp)
123B	Improper Tool Use	35%	60%	58% (−2 pp)	38% (−22 pp)
	Loop Behavior	8%	29%	31% (+2 pp)	13% (−16 pp)
	Insufficient Debugging	44%	59%	61% (+2 pp)	46% (−13 pp)
	Resolved Rate	71.0%	65.8%	67.4% (+1.6 pp)	70.4% (+4.6 pp)

Table 10 shows consistent behavioral degradation and recovery across both Devstral model scales. TQ-Mix reduces all three execution-control failures and brings task resolution close to BF16, whereas TQ-Bd provides limited and sometimes negative effects on behavioral recovery. These results suggest that increasing model scale alone does not eliminate quantization-induced behavioral failures, and that failure-centered precision restoration remains useful at both scales.

Sensitivity remains sparse across the evaluated architectures. We extend the localization analysis to five backbones spanning dense, mixture-of-experts, and hybrid-attention architectures, with model sizes ranging from 8B to 123B. Table 11 shows that the locations of sensitive layers vary across models, while the selected layers remain a minority of those analyzed. In particular, Devstral-2-123B has seven sensitive layers out of 88. For the hybrid Qwen3.6 model, localization is restricted to the

10 full-attention layers that maintain conventional token-level KV caches, of which two are selected. These observations extend the concentration pattern found in the primary model.

Table 11: Sensitive-layer localization across model architectures and scales. Layer counts refer to the full model. For the hybrid Qwen3.6 model, localization is performed only on its 10 full-attention layers.

Model	Architecture	Layers	Sensitive layers
LLaMA-3-8B-Instruct	Dense	32	9, 10, 11, 13
Devstral-Small-2-24B-Instruct	Dense	40	7, 13, 16, 28
Qwen3-Coder-30B-A3B-Instruct	MoE	48	3, 5, 16, 28, 40
Qwen3.6-35B-A3B	Hybrid MoE	$40\ (30+10)$	3, 23
Devstral-2-123B-Instruct	Dense	88	7, 15, 28, 33, 39, 45, 67

Table 12: Head- and layer-level sparsity across three full-attention models. Sensitive-layer and sensitive-head ratios are computed relative to all attention layers and heads, respectively. The final column reports the percentage of sensitive heads located in the selected sensitive layers.

Model	Sensitive-layer ratio	Sensitive-head ratio	Heads in sensitive layers
Qwen3-Coder-30B-A3B	$10.4\%\ (5/48)$	5.1%	89.6%
Devstral-Small-2-24B	$10.0\%\ (4/40)$	5.5%	86.0%
LLaMA-3-8B	$12.5\%\ (4/32)$	5.1%	88.5%

For the three full attention models with complete head-level statistics, only about 5% of heads are sensitive, and 86–90% of those heads lie inside the small set of sensitive layers (Table 12). Thus, head-level and layer-level observations are complementary: head-level analysis characterizes quantization sensitivity, while layer-level aggregation enables precision restoration within the serving implementation used in our experiments.

F Auxiliary Evaluation on Static Benchmarks

As a complementary evaluation beyond agentic coding, we compare KV cache configurations on four non-interactive coding benchmarks in the main text (Table 3),covering repository-level code completion, function-level program synthesis, and competitive programming. We use LLaMA-3-8B-Instruct with greedy decoding at temperature 0.To keep evaluation computationally manageable, we use subsets of selected benchmarks, as detailed below.

F.1 Coding Benchmarks

- **RepoBench-python-C.** RepoBench (Liu et al., 2024a) evaluates repository-level code completion using in-file and cross-file context. We use Python code-completion instances, denoted as `RepoBench-python-C`.
- **HumanEval.** HumanEval (Chen et al., 2021) contains Python programming problems, each specified by a function signature and docstring. Generated solutions are evaluated through execution against the benchmark tests.
- **MBPP.** MBPP (Austin et al., 2021) contains introductory Python programming tasks covering basic computation, string manipulation, and standard-library usage. Each task provides a natural-language specification and associated test cases.
- **LiveCodeBench-100.** LiveCodeBench (Jain et al., 2025) collects programming problems from competitive programming platforms and records their release dates. We evaluate code generation on the 100 most recent problems available at our evaluation cutoff, denoted as `LiveCodeBench-100`.

F.2 Coding Evaluation Metrics

We evaluate code completion using textual similarity metrics and program synthesis using execution-based correctness. All scores are reported on a 0–100 scale, with higher values indicating better performance.

- **Exact Match (EM).** For `RepoBench-python-C`, EM measures the percentage of predictions that exactly match the reference code after evaluation preprocessing:

$$\mathrm{EM} = \frac{100}{N} \sum_{i=1}^{N} \mathbb{I}(g(\hat{y}_i) = g(y_i)) \,, \tag{30}$$

  where N is the number of evaluated instances, $\hat{y}_i$ and y_i are the predicted and reference code, respectively, g denotes the preprocessing used by the evaluation script, and $\mathbb{I}$ is the indicator function.
- **Edit Similarity (ES).** For `RepoBench-python-C`, ES measures the average edit-based textual similarity between predictions and reference code using the evaluation script's similarity function. Unlike EM, it assigns partial credit to non-identical predictions. This metric measures reference similarity rather than functional correctness.
- **Pass@1.** For HumanEval, MBPP, and LiveCodeBench, we generate one solution per problem using greedy decoding and report the percentage of solutions that pass all required evaluation tests:

$$\mathrm{pass@1} = \frac{100}{N} \sum_{i=1}^{N} \mathbb{I}(\hat{y}_i \text{ passes all required tests}) \,. \tag{31}$$

  This execution-based metric accepts alternative implementations that satisfy the tests, without requiring textual agreement with a reference solution.

F.3 Long-Context Evaluation on LongBench-E

We evaluate on LongBench-E (Bai et al., 2024) to examine whether sensitivity-guided precision restoration also benefits non-agentic long-context tasks. LongBench-E provides a more balanced distribution of input lengths across the 0–4k, 4–8k, and 8k+ intervals. We follow its task-specific evaluation metrics and summarize performance across six task categories, reporting the unweighted mean of the category scores as the overall average.

Table 13: Auxiliary evaluation on LongBench-E. Bold values indicate the best score among the quantized configurations. The overall average is the unweighted mean across the six task categories. Higher scores are better.

Category	BF16	TQ-3bit	TQ-Bd	TQ-Mix
Single-Doc QA	48.78	47.91	**49.55**	48.19
Multi-Doc QA	53.89	52.59	52.84	**53.46**
Summarization	30.11	30.18	**30.32**	29.78
Few-shot	68.64	67.80	**68.53**	68.13
Synthetic	56.95	56.03	56.22	**57.22**
Code Completion	61.45	57.72	55.84	**60.13**
Overall Average	53.30	52.04	52.22	**52.82**

Table 13 shows that TQ-Mix achieves the highest overall average among the quantized configurations. Its largest improvement over TQ-3bit occurs in Code Completion, where it also outperforms TQ-Bd. This pattern is consistent with the coding focus of our calibration trajectories and provides complementary evidence that the resulting precision allocation can benefit non-interactive code completion. The benefits vary across tasks. TQ-Mix performs best among the quantized configurations on Multi-Doc QA, Synthetic tasks, and Code Completion, whereas TQ-Bd leads on Single-Doc QA, Summarization, and Few-shot tasks. TQ-Mix also scores slightly below TQ-3bit on Summarization. Overall, these results show improved average performance beyond agentic coding, with gains that depend on the task category.

G Statistical Analysis and Attention Visualizations

G.1 Confidence Intervals for Layer Sensitivity Scores

While the main text reports the average KL divergence score for each Transformer layer, relying solely on a point estimate (the sample mean) obscures the variance in sensitivity across different agentic trajectories. To provide a more robust and statistically rigorous validation of our findings, this section presents the 95% confidence intervals (CIs) for the layer-wise KL divergence scores. Interval estimation offers a more persuasive and reliable characterization of structural vulnerability by explicitly accounting for sample variance.

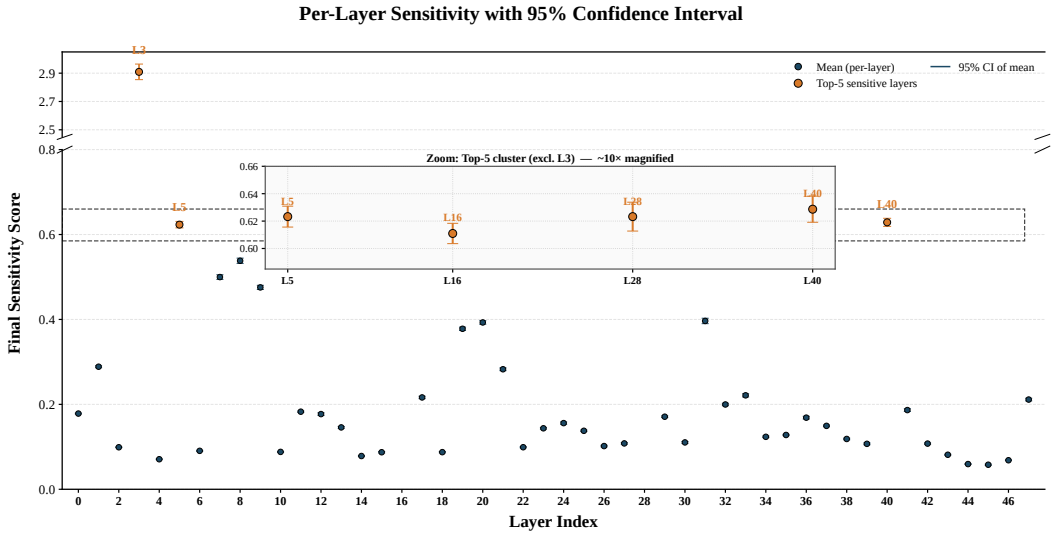


Figure 10: The 95% confidence intervals of the KL divergence score for each Transformer layer. The center inset magnifies the intervals by a factor of $10\times$ to visualize the minimal variance.

As illustrated in Figure 10, the computed confidence intervals for the vast majority of layers are exceptionally narrow, rendering them visually indistinguishable from single point estimates at a standard scale. To provide a clearer perspective on this variance, the center of the figure includes an inset that magnifies the confidence intervals by a factor of $10\times$. Crucially, even under this aggressive magnification, the intervals remain tightly constrained. This minimal variance demonstrates that our layer-wise sensitivity estimations are highly stable and robust across diverse agentic trajectories, confirming that the identified vulnerable layers are consistent structural bottlenecks.

G.2 Comparison of Attention Heads in Sensitive vs Non-Sensitive Layers

In this subsection, we visualize all attention heads from the most sensitive layer (Layer 3) alongside a non-sensitive control layer (Layer 15).

Specifically, we compare their states under the unquantized baseline (BF16) and after applying TQ-3bit. Comparing the two figures 11 and 12, we observe a pronounced distributional shift before and after quantization for the vast majority of attention heads in Layer 3, as evidenced by the substantial shaded area between the two curves. In contrast, for Layer 15, the two curves nearly coincide across most attention heads, leaving virtually no shaded region. This stark contrast clearly demonstrates the significance of our layer localization.

Furthermore, as illustrated in the Fig11, Layer 3 overwhelmingly allocates its attention weights to short-term tokens. Specifically, the cumulative attention weight assigned to the nearest 64 tokens is remarkably high across most heads. For a sequence length of at least several thousand tokens, a uniform attention distribution would yield a cumulative proportion of less than 10% (0.1) for this local

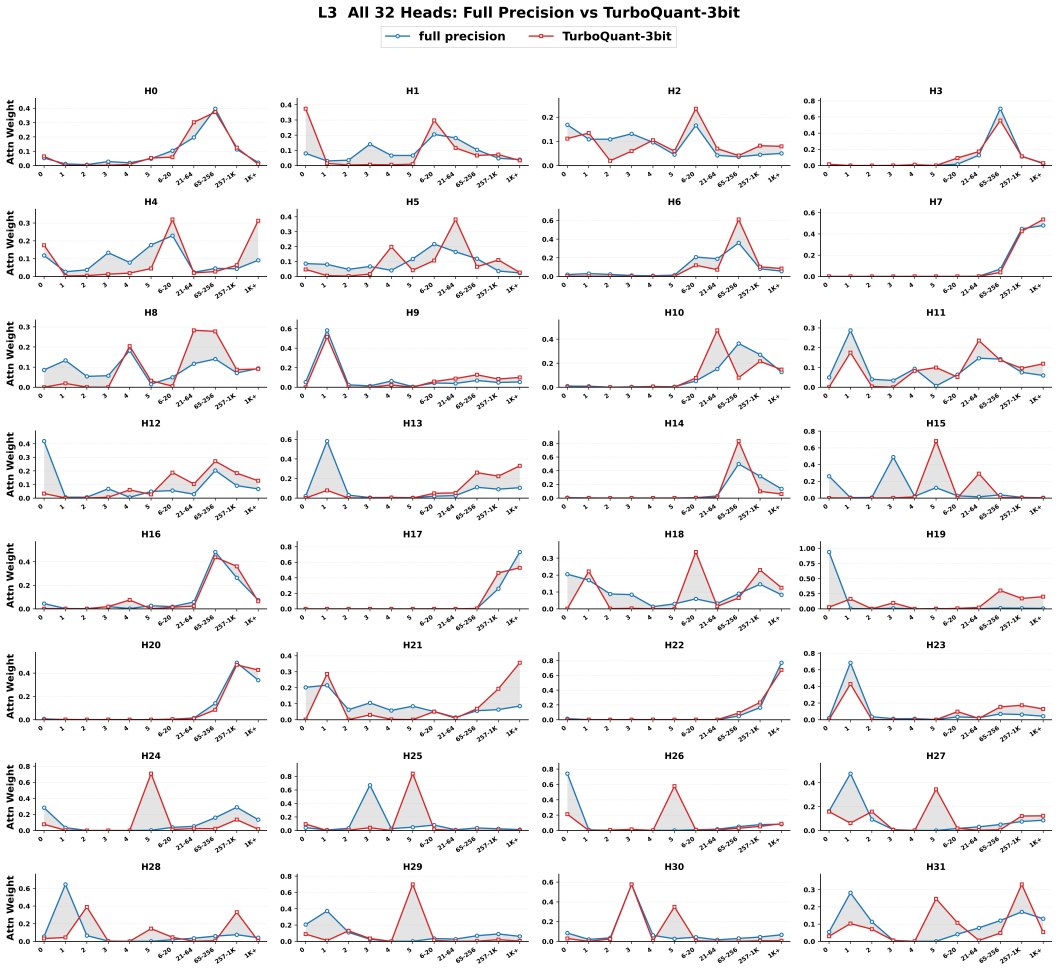


Figure 11: Distribution of Attention Scores Across Each Head in Layer 3.

window. However, the subplots reveal that the actual weight allocated to these 64 tokens far exceeds 0.1. This observation further corroborates our argument that quantization selectively damages local attention patterns.

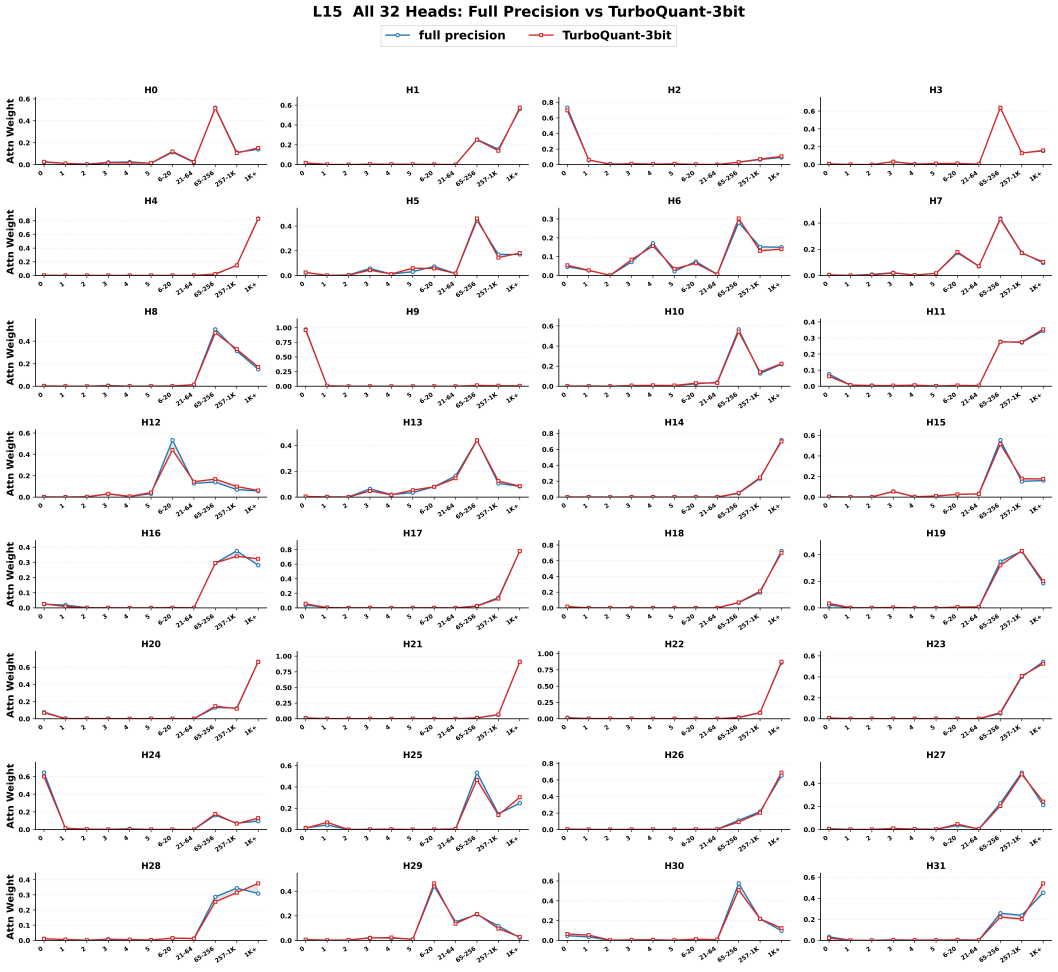


Figure 12: Distribution of Attention Scores Across Each Head in Layer 15.