QUOTA: LEARNING DOWNSTREAM-AWARE CONTEXT ALLOCATION FOR TOOL-USING LANGUAGE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-using language agents accumulate large volumes of tool outputs over long-horizon interactions. Existing compression methods primarily optimize immediate context length, but removing critical information can trigger task failures or redundant tool re-invocations, causing local token savings to increase end-to-end cost. We introduce QUOTA, a query-conditioned framework that learns how to allocate context budgets across individual tool outputs. At each compression event, QUOTA assigns a continuous retention quota to every output based on the query, execution state, and output characteristics. The policy is trained with REINFORCE using an objective that jointly considers task success, agent token consumption, and post-compression tool re-invocations, while keeping the agent and compressor frozen. To address delayed credit assignment, QUOTA learns an outcome model that compares each selected quota against a full-retention counterfactual, producing output-specific signals for policy optimization. Across held-out queries from a production data-analysis agent and three compressor backends, QUOTA reduces agent token consumption by 29%–46% relative to full retention, at success rates of 0.75–0.81 against 0.75 for full retention. It adds 15–18 percentage points of savings over a tuned tool-type-conditional policy. Interventional rollouts validate the learned outcome estimates, while cross-backbone and cross-compressor experiments demonstrate policy transfer. A separately trained policy reduces agent tokens by 18%–25% on five unseen LOCA-bench environments at success rates of 0.38–0.48 against 0.37 for full retention, showing that downstream-aware context allocation improves the efficiency of long-horizon language agents.

1 INTRODUCTION

Tool-using language agents answer questions through sequences of tool calls (Yao et al., 2023; Schick et al., 2023). Accumulated outputs can exhaust context windows and token budgets: a single SQL result may exceed ten thousand tokens. Compression reduces this immediate footprint, but removing a detail needed later can trigger costly recovery calls. We call the mismatch between local savings and downstream agent cost the **compression–consequence gap** (Figure 1).

Existing methods prune tokens (Jiang et al., 2023; 2024; Pan et al., 2024), summarize text (Xu et al., 2024), or drop steps (Zhang & Sun, 2026). Applying these operations also requires deciding how much context to allocate to each output. This matters because outputs differ in downstream utility: a schema lookup may support many subsequent steps, while a permission check may never be revisited. Repeated compression therefore motivates query- and state-conditioned allocation that accounts for the cost of recovering discarded information.

We propose **QUOTA**, a small retention policy trained with REINFORCE (Williams, 1992) while the agent and compressor remain frozen. Its reward combines task success, session tokens, and event-local excess calls. A learned outcome model supplies auxiliary per-output credit by comparing selected ratios with full retention, supplementing the shared trajectory signal.

Our contribution is the formulation and implementation of downstream-aware allocation at the tool-output level, using established policy-gradient and counterfactual-credit principles. Specifically:

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.

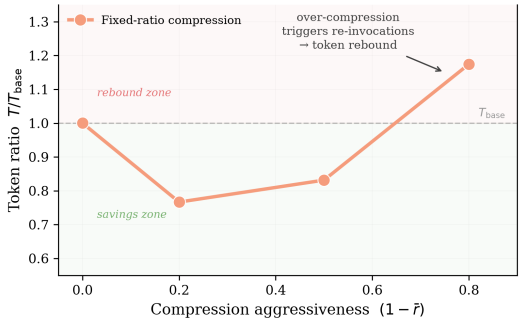

Figure 1: Schematic of the compression–consequence gap. Removing more context lowers the immediate footprint, but past some point the agent re-invokes tools to recover what was dropped, and total agent tokens rise again.

- We formulate repeated context compression as a sequential allocation problem whose objective accounts for task success, agent tokens, and subsequent excess tool calls.
- We develop a query- and state-conditioned controller that assigns a continuous quota to each output instance while keeping both the agent and compressor frozen.
- We combine measured trajectory feedback with learned single-output credit, and assess this design through policy ablations, real retention interventions, and transfer across backbones, compressors, and task environments.

2 Related Work

Learned context management Token pruning selects text by information scores or learned labels (Jiang et al., 2023; 2024; Pan et al., 2024; Li et al., 2023); LLM-DPC learns sequential token removal (Hu et al., 2026), and AGORA scores observation–action steps by counterfactual next-action change (Zhang & Sun, 2026). PACE adapts history granularity using next-action relevance (Wei et al., 2026). ACON optimizes compression guidelines (Kang et al., 2026), while AdaCoM learns an external manager for a frozen agent (Yi et al., 2026). QUOTA instead assigns continuous quotas to individual output instances, keeping both agent and compressor fixed and learning from session outcomes.

Routing and credit assignment RouteLLM allocates computation by selecting a model (Ong et al., 2025); QUOTA allocates context among existing outputs. Learning this allocation requires credit for retention decisions whose effects emerge later in the session. Agent RL typically assigns step- or turn-level credit (Wang et al., 2026; Li et al., 2026a;d). We apply REINFORCE and counterfactual-credit principles (Williams, 1992; Foerster et al., 2018; Wolpert & Tumer, 2001) at the output-instance level while keeping the agent and compressor frozen. Appendix K and Table A4 compare action spaces and feedback.

3 Method

Figure 2 summarizes QUOTA: at each compression event a policy assigns output-specific retention ratios, a fixed compressor applies them, and measured rollout outcomes plus learned counterfactual credit train the policy.

3.1 Problem Formulation

An agent $\mathcal{A}$ appends tool outputs to context c_t until it answers or reaches its iteration limit. Whenever $|c_t|$ exceeds budget τ, a compression event reduces tool outputs while preserving the system prompt and tool definitions. A session is an episode with T_c such events. At event t, state s_t encodes segment shares and per-output features; action a_t assigns ratios to the N active outputs. Removing information can affect later calls and success, coupling decisions across events (Appendix D).

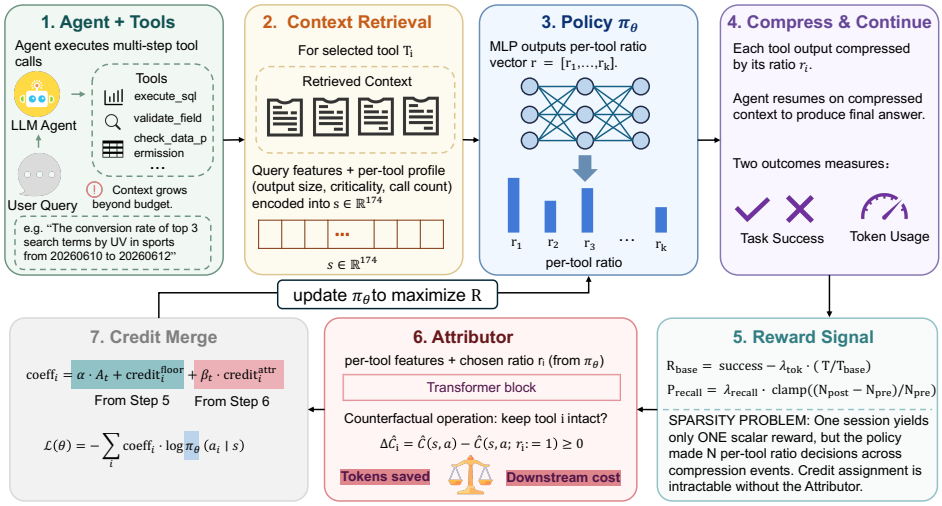

Figure 2: Overview of QUOTA. QUOTA uses a policy network to decide how much of each tool's information to keep during compression, trains that policy online from trajectory outcomes, and uses a post-hoc counterfactual model to estimate output-specific credit for policy training.

Output instances and repeated compression A tool type (e.g., `execute_sql`) names an operation; each invocation produces a distinct output instance i. At event t, an active slot represents one such instance in its currently retained form $x_{t,i}$. The compressor applies the requested ratio to this current content:

$$x^{+}_{t,i} = \mathcal{C}(x_{t,i}, r_{t,i}), \qquad \mathrm{tok}_{t,i} = |x_{t,i}|.$$

Subsequent compression of the same instance uses its retained content, not an original cached copy. Thus $r_{t,i} = 1$ preserves everything currently available for that instance; it does not restore information removed at an earlier event. Requested quotas and realized lengths can differ with the backend.

A two-event example appears in Appendix D.1.

3.2 Consequence-Aware Objective

Session tokens include downstream recovery costs; an event-local recall penalty adds a preference against excess calls.

Session-level base reward The first component combines task success with token efficiency and is measured only at episode end:

$$R_{\text{base},k} = \mathbb{1}[k{=}T_c]\Big(\text{success} - \lambda_{\text{tok}}\ \text{clamp}\left(\tfrac{T}{T_{\text{base}}}, 0.2, 2\right)\Big), \tag{1}$$

where success $\in \{0,1\}$ indicates whether the agent answered correctly, T is the agent-side session token count, including subsequent recovery interactions but excluding compressor-model calls and local scoring compute, and T_{base} is the token total under the same query with no compression, averaged over five reference runs.

Event-local recall penalty We assess excess calls after each compression event:

$$P_{\text{recall},t} = \lambda_{\text{recall}}\, h(N_{\text{post},t}, N_{\text{pre},t}),$$
$$h(m,n) = \begin{cases} \text{clamp}\left(\dfrac{m-n}{n}, 0, 2\right), & n > 0, \\ 0, & n = 0,\ m = 0, \\ 2, & n = 0,\ m > 0. \end{cases} \tag{2}$$

Here $N_{\text{post},t}$ counts calls in the post-event window. $N_{\text{pre},t}$ averages the corresponding counts over five keep-all timelines, matching compression-trigger windows by event order. Fractional positive reference means use the ratio branch; zero references use the explicit branches above. Missing measurements invalidate the recall signal and skip the policy update (Algorithm 2).

Combined objective Combining the two components defines the task-level objective:

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\big[R_{\text{base}} - P_{\text{recall}}\big], \quad P_{\text{recall}} = \textstyle\sum_t P_{\text{recall},t}. \tag{3}$$

Because the measured recall signal carries noise from natural call-count variation between rollouts, it is a proxy for compression-related recovery rather than a causal count of recovery calls. Appendix E describes an optional provenance-matching extension, unused in the reported experiments.

3.3 Per-Tool Compression Policy

Action parameterization The retention ratio a_i lives on a bounded interval. We use a Beta distribution whose support matches this interval by construction, avoiding truncation artifacts:

$$\tilde{a}_i \sim \text{Beta}(\tilde{m}_i\kappa_0,\ (1-\tilde{m}_i)\kappa_0), \tag{4}$$

where $\tilde{m}_i = \sigma(\text{net}_i)$ is the network-predicted mean and κ_0 is a fixed concentration. At evaluation we replace sampling with the deterministic mean $\tilde{m}_i$.

Network architecture A two-hidden-layer MLP maps $s \in \mathbb{R}^{174}$ to means for N output-instance slots. Each slot encodes query relevance, prefix-available reuse, redundancy, and recency; repeated calls to one tool occupy distinct slots. A presence mask restricts gradients to active outputs (features in Appendix F).

Training Equation 3 specifies the desired task trade-off. Training uses a shaped score-function surrogate with measured and model-based credit. For a query q, define the undiscounted base return and normalized advantage before updating its running statistics:

$$G_t^{\text{base}} = R_{\text{base},T_c}, \qquad A_t = \frac{G_t^{\text{base}} - b_q}{\sigma_q}.$$

Here b_q is the per-query EMA of prior base rewards and σ_q is their running scale. Recall is handled separately by the event-local floor below; it is not subtracted again inside A_t. For an episode with $T_c > 0$, the policy loss is

$$\mathcal{L}_{\text{sur}}(\theta) = -\frac{1}{T_c}\sum_{t=1}^{T_c}\sum_{i\in\text{active}_t} \text{sg}[\text{coeff}_{i,t}]\log\pi_\theta(a_{t,i}\mid s_t), \tag{5}$$

where sg holds the sampled credit fixed during the policy update. The predictor is fitted separately. Episodes with no compression event have no retention-policy score terms.

Appendix A derives the Beta score and action-independent baseline identity.

3.4 Counterfactual Credit Assignment

Shared trajectory returns do not explicitly identify output-specific consequences. We supplement them with a model-free penalty floor and learned per-output credit.

Channel I: uniform penalty floor The first channel distributes $P_{\text{recall},t}$ uniformly over the k active tools:

$$\begin{aligned} \text{credit}_i^{\text{floor}} &= -P_{\text{recall},t}\,\frac{\mathbb{1}[i\in\text{active}]}{k}, \\ \sum_i \text{credit}_i^{\text{floor}} &= -P_{\text{recall},t}. \end{aligned} \tag{6}$$

Even if the attribution channel degrades entirely, this floor preserves the measured aggregate penalty independently of the outcome model.

Channel II: counterfactual attribution via outcome model To estimate how retention affects downstream cost, we train an outcome model f_ϕ over the query and K active output instances. Here per-tool means per-output-instance: repeated calls to one tool occupy separate slots with separate predictions and credit. Self-attention captures dependencies between outputs, such as compressing one output leading to a call to another tool. The model is a 2-layer, 4-head Transformer with K output tokens and one query token. It predicts per-instance re-invocation count $\hat{n}_i$, log token total $\hat{T}_{\log}$, and auxiliary success probability $\hat{p}$. Only counts and tokens enter $\hat{C}$; observed success enters the session reward.

This model enables single-tool counterfactuals. For each output instance i, we compare the predicted cost under the chosen ratio versus $r_i{:=}1$, holding all else fixed. The alternative preserves its current pre-event content, including any compression inherited from earlier events. Defining $\hat{C}(s,a) = \lambda_N \sum_j \hat{n}_j + \lambda_T \cdot 2\tanh(\frac{1}{2}\frac{\exp \hat{T}_{\log}}{T_{\text{base}}})$, the per-tool credit becomes:

$$\text{credit}_i^{\text{attr}} = \underbrace{\lambda_{\text{save}}(1-r_i)\tfrac{\text{tok}_i}{T_{\text{base}}}}_{\text{local-saving proxy}} - \underbrace{\left[\hat{C}(s,a) - \hat{C}(s,a;\, r_i{:=}1)\right]_+}_{\text{predicted excess cost}}. \tag{7}$$

Here $[x]_+ = \max(0,x)$, matching Algorithm 3. The unrectified difference is used for the signed intervention comparison in Section 4.4. The credit balances immediate savings against predicted excess cost.

Channel fusion We fuse both channels into the per-tool gradient coefficient:

$$\text{coeff}_{i,t} = \underbrace{\lambda_A A_t + \text{credit}_i^{\text{floor}}}_{\text{measured}} + \underbrace{\omega_e\, \text{credit}_i^{\text{attr}}}_{\text{attributed}}, \tag{8}$$

where λ_A scales the session advantage and ω_e controls the contribution of learned credit. To collect training data before using this credit, we keep $\omega_e{=}0$ for 20 episodes, then increase it linearly to 1 over 10 episodes. The first outcome-model fit follows episode 20, with updates every 5 episodes thereafter (Appendix B).

Role of the learned credit The learned credit shapes the surrogate in Eq. 5. Because the token head predicts total agent tokens, its signal overlaps with the local-saving preference; Eq. 8 therefore does not partition J, and evaluation counts actual tokens only once. Moreover, fitting the model to observed rollouts can leave the full-retention boundary sparsely covered. We therefore retain measured returns and the uniform floor throughout training, using the learned difference as auxiliary guidance (Appendix C.1).

4 Experiments

4.1 Setup

Platform We evaluate on a production data-analysis agent deployed at a large e-commerce company. The agent answers business questions using tools for knowledge search, SQL execution, permission checks, table lookup, and date resolution. The agent backbone is Claude Sonnet 4.6 (Anthropic, 2026), with a 200K-token context window, held fixed throughout all experiments.

We evaluate three compressor backends ordered by model reliance. Uniform truncation needs no model. Self-Info, a clause-level variant of Selective Context (Li et al., 2023), ranks clauses by self-information scored by Qwen-2.5-0.5B (Team, 2024). Summarization uses Qwen-3.7-Max, a member of the Qwen3 family (Team, 2025), to condense text semantically. A fourth AGORA-style (Zhang & Sun, 2026) step-level compressor is held out to test cross-compressor transfer.

Queries We collect 120 queries from production logs and attach a human-verified answer to each query. The corpus contains 25 simple lookups, 55 multi-step schema-retrieval/SQL queries, and 40 multi-table permission-gated joins, split 80/40 into training and held-out evaluation by difficulty stratum (Appendix H). Correctness is determined against the verified answer, with a 5% tolerance for numeric queries. Each RL configuration trains for 100 episodes per seed with 5 seeds, and held-out evaluation uses frozen deterministic policies. Appendix H documents the split, and Appendix M describes the anonymized schema and backend requirements.

Baselines All strategies share the same compressor backend. Keep-all is the no-compression reference and token-normalization baseline. Uniform-0.5 retains half of each output; Recency favors newer outputs; Token-proportional compresses larger outputs more. Tool-type-conditional tunes one fixed ratio per canonical tool type on training queries, using greedy coordinate descent to maximize success minus 0.3 times token ratio. Appendix G.1 provides the formulas and tuning procedure.

Configuration We use $a_i = 0.05 + 0.95\tilde{a}_i$, Beta concentration $\kappa_0 = 8$, and no event discount. Appendix G gives all reward weights, optimizer settings, and comparison details.

Metrics We report agent token usage relative to keep-all over valid held-out query–seed records, without conditioning on success. Token summaries use the interquartile mean and stratified-bootstrap 95% confidence intervals; they exclude summarizer-model calls, Self-Info scorer compute, and monetary pricing (Appendix L). Success rates use the same records, whose effective count may differ from the nominal 40×5 runs. The recall statistic p_{recall} averages the unweighted rate in Eq. 2 first over events within each record, then across records. Tables 1 and 2 share these definitions and the same QUOTA runs, but use different comparison arms. Within each compressor, paired Wilcoxon signed-rank tests compare token ratios on common records; success differences are descriptive. Appendix I gives the estimators and tests.

4.2 Main Comparison: QUOTA vs. Fixed Strategies

Does learning per-tool retention ratios improve upon fixed compression strategies?

Table 1 shows that the untuned Uniform-0.5, Recency, and Token-proportional baselines increase agent token usage by 3%–24% under truncation and Self-Info, with $p_{\text{recall}} \geq 0.07$. This pattern is consistent with the compression–consequence gap: local compression can be accompanied by additional downstream calls and higher session-level token usage.

Table 1: Main comparison: QUOTA vs. fixed and heuristic strategies. Agent token ratio = $T/T_{\text{keep-all}}$. Cost/succ. = Tok. ratio / Success summarizes tokens spent per success; compressor costs are excluded (Appendix L). Keep-all is the reference timeline for p_{recall}, shown as a dash. Boldface highlights QUOTA.

Compressor	Strategy	Success↑	Tok. ratio↓	Save	p_{recall}↓	$\bar{N}_{\text{tools}}$	Cost/succ.↓
Truncation	Keep-all	0.75	1.000	0%	—	7.1	1.333
	Uniform-0.5	0.73	1.026	−3%	0.082	9.3	1.405
	Recency	0.77	1.086	−9%	0.071	9.0	1.410
	Token-prop.	0.69	1.112	−11%	0.091	9.5	1.612
	Tool-type-cond.	0.76	0.841	16%	0.045	8.5	1.107
	QUOTA	**0.77**	**0.694**	**31%**	0.021	7.8	**0.901**
Summarization	Keep-all	0.75	1.000	0%	—	7.1	1.333
	Uniform-0.5	0.80	0.832	17%	0.105	9.6	1.040
	Recency	0.80	1.194	−19%	0.088	9.1	1.493
	Token-prop.	0.77	1.231	−23%	0.098	9.4	1.599
	Tool-type-cond.	0.79	0.723	28%	0.052	8.8	0.915
	QUOTA	**0.81**	**0.541**	**46%**	0.015	7.2	**0.668**
Self-Info	Keep-all	0.75	1.000	0%	—	7.1	1.333
	Uniform-0.5	0.62	1.207	−21%	0.128	10.2	1.947
	Recency	0.71	1.242	−24%	0.112	10.4	1.749
	Token-prop.	0.66	1.187	−19%	0.121	10.0	1.798
	Tool-type-cond.	0.70	0.891	11%	0.062	9.1	1.273
	QUOTA	**0.75**	**0.706**	**29%**	0.025	8.1	**0.941**

QUOTA reduces agent tokens by 29%–46% relative to keep-all: 30.6% under truncation, 45.9% under summarization, and 29.4% under Self-Info. Corresponding success rates are 0.77, 0.81, and 0.75, with excess-call rates of 0.021, 0.015, and 0.025. Thus the savings accompany success point estimates at or above keep-all's 0.75, with observed differences of $+2$, $+6$, and 0 percentage points.

The savings also extend beyond tuned tool-type allocation: QUOTA uses 17.5%, 25.2%, and 20.8% fewer agent tokens, respectively. This supports adapting retention at the output-instance level beyond a fixed ratio for each tool type. Figure 3 shows the training curves.

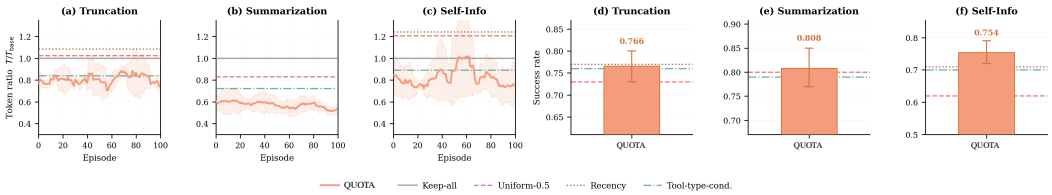

Figure 3: QUOTA vs. fixed-strategy baselines across three compressors. Panels (a)–(c): token-ratio training curves (5 seeds, 15-episode rolling window, ± 1 std). Panels (d)–(f): task success rate with 95% CI. The gray keep-all line in (a)–(c) denotes token ratio 1. Keep-all runs no compressor and has a single success rate of 0.75 on this evaluation set (Table 1).

4.3 Ablation: Effect of Attribution

Does the outcome model improve QUOTA's compression decisions?

We compare QUOTA against an ablated RL-only variant that uses REINFORCE with uniform credit assignment. Both arms share the same policy architecture, reward, and training schedule; the full arm activates the auxiliary attribution channel of Eqs. 7–8 while RL-only sets $\omega_e = 0$. The comparison therefore measures that channel's net effect, including both its local-saving and predicted-cost terms.

Table 2: Ablation of QUOTA against the RL-only baseline. Each arm uses 5 seeds. The token ratio uses the same definition as Table 1, $T/T_{\text{keep-all}}$ over all valid held-out query–seed records with no success conditioning, and success rates use those same records; $\Delta_{\text{tok}} = 100(r_{\text{QUOTA}}/r_{\text{RL-only}} - 1)\%$. The QUOTA rows summarize the same evaluation runs as Table 1 under the same estimator, so their token ratios and excess-call rates repeat that table's values and only the comparison arm differs (Appendix I).

Compressor	Arm	Success↑	Tok. ratio↓	95% CI	Δ_{tok}	p_{recall}↓	$\bar{T}_{\text{abs}}$ (K)
Truncation	RL-only	74.8%	0.728	[.703, .753]	—	0.022	72.8
	QUOTA	76.6%	0.694	[.685, .704]	−4.7%	0.021	69.4
Summarization	RL-only	76.2%	0.560	[.551, .569]	—	0.018	47.5
	QUOTA	**80.8%**	**0.541**	[.532, .550]	−3.4%	**0.015**	**45.9**
Self-Info	RL-only	71.5%	0.736	[.722, .750]	—	0.028	71.2
	QUOTA	**75.4%**	**0.706**	[.693, .719]	−4.1%	**0.025**	**68.3**

Adding auxiliary attribution reduces agent tokens by 4.7%, 3.4%, and 4.1% under truncation, summarization, and Self-Info, respectively (Table 2). On the same records, observed success rises by 1.8, 4.6, and 3.9 percentage points, while excess-call rates fall from 0.022 to 0.021, 0.018 to 0.015, and 0.028 to 0.025. Two-sided paired Wilcoxon tests on token ratios give $p = 0.019$, 0.003, and 0.006; all remain below 0.05 after Holm–Bonferroni correction across the three compressors. The pooled test gives $p = 0.008$ (Appendix I). Figure 4 shows the training dynamics.

The joint reduction in tokens and excess calls is consistent with the allocation mechanism in Appendix C. A shared trajectory return evaluates the complete set of retention decisions, while the auxiliary channel distinguishes outputs by their predicted downstream consequences. Its local-saving term favors compression, and its counterfactual-cost term offsets that incentive when preserving an output is predicted to avoid later cost. This gives the policy an output-specific signal for deciding where compression is worthwhile. The same pattern across truncation, summarization, and Self-Info supports the usefulness of this signal across different ways of realizing a quota. The following intervention study examines the complementary question: whether the predicted consequences agree with actual changes in downstream cost when a single output is preserved.

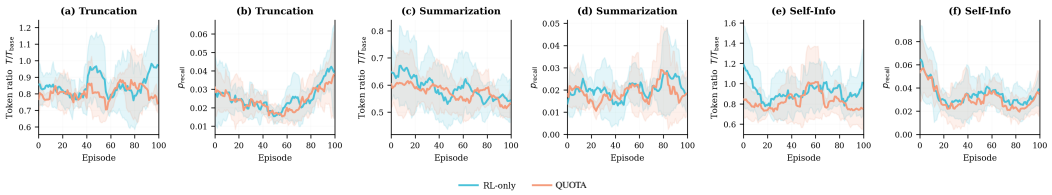

Figure 4: Attribution ablation, RL-only (blue) vs. QUOTA (orange). Columns are per-compressor; each shows token ratio and excess-call rate p_{recall}. 5 seeds, 15-episode rolling window; bands are ± 1 std.

4.4 Validation with Interventional Rollouts

The ablation measures the attribution channel's training benefit. We now test whether its predicted cost differences agree with actual single-output interventions. This complements prediction on observed outcomes, assessed separately in Appendix G.4.

From held-out trajectories we sample compression states s together with the frozen policy's action $a = (r_1, \ldots, r_K)$. For each active tool i we replay the agent from the saved snapshot twice: a factual rollout under a, and an intervention rollout that sets $r_i{:=}1$ while holding all other ratios fixed. Both use the same frozen deterministic policy after the event and share $M{=}3$ paired seeds to reduce sampling variability. The measured consequence and its model estimate are

$$\Delta C_i^{\text{roll}} = \tfrac{1}{M} \textstyle\sum_{m=1}^{M} \left[C^{(m)}(s,a) - C^{(m)}(s,a; r_i{:=}1) \right], \tag{9}$$

$$\Delta \hat{C}_i = \hat{C}(s,a) - \hat{C}(s,a; r_i{:=}1), \tag{10}$$

where a positive value means that preserving all currently retained content of output i at this event lowers expected downstream cost relative to the policy-selected ratio. Both branches start from the same pre-event snapshot; neither restores material removed before that snapshot.

Table 3: Validation of the outcome-model counterfactuals against single-tool intervention rollouts on held-out states. Cells are mean$_{\pm 95\%\ \text{CI}}$.

Metric	Trunc.	Summ.	Self-Info	Aggr.
Spearman $\rho \uparrow$	$0.64_{\pm.05}$	$\mathbf{0.73}_{\pm.04}$	$0.61_{\pm.06}$	$0.66_{\pm.03}$
NMAE$\downarrow$	$\mathbf{0.11}_{\pm.02}$	$0.12_{\pm.01}$	$0.17_{\pm.02}$	$0.13_{\pm.01}$
Top-3 overlap$\uparrow$	$0.80_{\pm.05}$	$\mathbf{0.86}_{\pm.04}$	$0.81_{\pm.05}$	$0.82_{\pm.03}$
Sign Acc.$\uparrow$	$0.83_{\pm.04}$	$\mathbf{0.88}_{\pm.03}$	$0.80_{\pm.04}$	$0.84_{\pm.02}$

Table 3 measures agreement with interventional outcomes: predicted vs. rollout differences yield Spearman $\rho = 0.66$, the attributor recovers 82% of the true top-3 highest-consequence output instances, and 84% of predicted credit directions match rollout. These metrics assess ranking and direction for the same full-retention intervention used by the credit model, complementing the optimization-level evidence of Table 2.

4.5 Behavior and Transfer

Learned quotas vary with output type and backend: under summarization, they range from 0.48 for `query_date` to 0.65 for `execute_sql` (Figure 5). We next test whether the policy remains useful when either system component changes. Frozen-policy transfer to Qwen-3.7-Max and GPT-5.5 saves 18%–40% agent tokens, with success 0.69–0.78 against keep-all rates of 0.71 and 0.74, respectively (Table 4). On an AGORA-inspired compressor, zero-shot transfer saves 31% at success 0.62; backend-specific training raises savings to 34% and success to 0.65, against keep-all's 0.64 (Table 5; Appendix N).

4.6 Generalization on LOCA-bench

We further test generalization on LOCA-bench (Zeng et al., 2026), a public benchmark whose 15 task environments, tool set, and task distribution are disjoint from our production setting, with deter-

Table 4: Frozen-policy backbone transfer from Claude Sonnet 4.6. Keep-all is the no-compression reference.

Comp.	Backbone	Succ.↑	Tok. r.↓	p_{rec}↓
Keep-all	Qwen-3.7-Max	0.71	1.00	—
	GPT-5.5	0.74	1.00	—
Trunc.	Qwen-3.7-Max	0.72	0.78	0.035
	GPT-5.5	0.75	0.74	0.026
Summ.	Qwen-3.7-Max	0.75	0.65	0.028
	GPT-5.5	0.78	0.60	0.020
Self-Info	Qwen-3.7-Max	0.69	0.82	0.039
	GPT-5.5	0.72	0.77	0.029

Table 5: Transfer to the AGORA-style compressor.

Strategy	Succ.↑	Tok.↓	Save
Keep-all	0.64	1.000	0%
Uniform-0.25	0.65	1.208	−21%
QUOTA (zero-shot)	0.62	0.690	31%
QUOTA (on AGORA)	**0.65**	**0.656**	**34%**

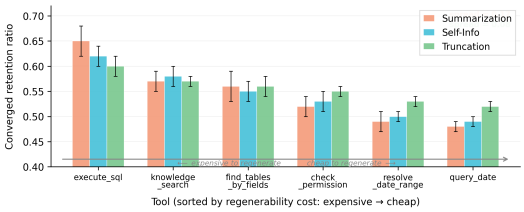

Figure 5: Converged retention ratios, averaged over the output instances of each tool type. Types are sorted by regenerability cost.

Table 6: LOCA-bench: 5 held-out tasks × 5 seeds × 3 windows; aggregate point estimates.

Comp.	Strategy	Succ.↑	Tok.↓
Keep-all		0.37	1.000
Trunc.	Recency	0.39	0.978
	Token-prop	0.40	0.961
	QUOTA	**0.44**	**0.751**
Summ.	Recency	0.41	0.847
	Token-prop	0.23	0.938
	QUOTA	**0.48**	**0.806**
Self-Info	Recency	0.35	0.891
	Token-prop	0.24	0.922
	QUOTA	**0.38**	**0.815**

ministic workspace-comparison evaluation. A fresh QUOTA policy is trained on 10 environments and evaluated on the 5 held out, aggregated over the three longest context windows (Table 6; Appendix G.5).

QUOTA saves 24.9%, 19.4%, and 18.5% agent tokens, with success 0.44, 0.48, and 0.38 versus keep-all's 0.37 (Table 6). These point estimates show savings on disjoint public environments; the one-percentage-point Self-Info difference is descriptive. Token-proportional, which allocates by size alone, instead falls to 0.23 and 0.24 success under summarization and Self-Info against keep-all's 0.37 while spending more tokens than QUOTA—the failure mode this objective targets.

Limitations Our evaluation combines 40 held-out production queries with disjoint LOCA-bench environments, providing evidence across two settings. Broader deployment studies and comparisons with tuned scalar allocations, restricted-input policies, and other learned context managers would further clarify the contribution of individual design choices. We report agent-token efficiency; its translation into deployment savings depends on backend-specific compression overhead (Appendix L). Statistical tests focus on token usage, with success rates reported descriptively. The event-local excess-call measure provides a practical recovery signal, while finer-grained provenance tracking offers a direction for distinguishing compression-induced calls from other trajectory variation.

5 Conclusion

QUOTA learns how much of each tool output to retain as a task progresses, while keeping the agent and compressor frozen. This allows context allocation to be optimized without retraining either deployed component. Across three compressor backends, downstream feedback reduces agent token use, and auxiliary counterfactual credit adds gains over uniform-credit RL. Intervention studies support the predicted consequences, while transfer experiments and a separately trained LOCA-bench policy broaden the evaluation. Together, these findings motivate context compression as resource allocation: retention should reflect how each output supports the agent's remaining work.

ETHICS STATEMENT

The production evaluation uses 120 business-analytics queries drawn from the operational logs of an internally deployed agent, together with answers verified by the analysts who own the corresponding reports. The queries concern aggregate business metrics and contain no personal data. Identifiers, table names, and column names are replaced with neutral placeholders in the released schema file, and no raw prompts, tool outputs, trajectories, or warehouse contents are released. The work was carried out with the operator's permission under its internal data-handling policy, and every tool call passed through the same permission checks that govern normal production use.

QUOTA removes material from an agent's working context, which creates one method-specific risk: if an output records an action that cannot be repeated safely, compressing it can induce a redundant re-invocation rather than a harmless re-read. A deployment that exposes tools with irreversible effects should therefore exempt the corresponding outputs from allocation; Appendix J specifies that extension.

REPRODUCIBILITY STATEMENT

The anonymized code and data accompanying this submission are provided in the supplementary `.zip` archive uploaded with the paper. The code supplement described in Appendix M provides the reference policy, feature construction, outcome model, reward and credit functions, training loop, compressor interfaces, evaluation and statistical utilities, offline tests, dependency specifications, and an anonymized query-schema file. The appendices document the state features, training algorithms, reference implementation defaults, and reported statistical procedures. The schema records the production-query composition and split structure; it does not include the private verified evaluation targets.

Exact replay of the production results requires the proprietary agent, queries, verified answers, and backend state. For LOCA-bench, Appendix G.5 reports the training/test environment counts, context-window settings, training rounds, and seed count. The benchmark and model endpoints must be obtained separately; the supplement provides integration and evaluation scripts rather than a self-contained replay of the reported runs.

AI USE STATEMENT

Large language models were used solely to polish the writing of this paper, meaning grammar, wording, and formatting of text the authors had already drafted. They were not used to generate research ideas, design the method, run or analyze experiments, or write code; the language models named in Section 4.1 are experimental subjects, not writing tools. The authors reviewed every part of the manuscript and take full responsibility for its content.

REFERENCES

Anthropic. Claude Sonnet 4.6 system card. Anthropic system card, 2026. URL https://www.anthropic.com/claude-sonnet-4-6-system-card.

Jose A. Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. RUDDER: Return decomposition for delayed rewards. In *Advances in Neural Information Processing Systems*, 2019.

Hongye Cao, Nuo Yan, Haoyuan Deng, Ziwei Wang, Tianpei Yang, Jing Huo, Yuyao Zhang, and Yang Gao. Tool-aware optimization with entropy guidance for efficient agentic reinforcement learning. arXiv preprint arXiv:2606.03762, 2026.

Jianming Chen, Yawen Wang, Junjie Wang, Xiaofei Xie, Jun Hu, Qing Wang, and Fanjiang Xu. Understanding individual agent importance in multi-agent system via counterfactual reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 15785–15794, 2025.

Liuji Chen, Dianxing Tang, Xing Shi, Dingshuo Chen, Qiang Liu, Shu Wu, and Liang Wang. Learning when not to act: Mitigating tool abuse in agentic reinforcement learning. arXiv preprint arXiv:2606.02132, 2026.

Xin Cheng, Shuo He, Lang Feng, HaiYang Xu, Ming Yan, Lei Feng, and Bo An. Beyond trajectory-level attribution: Graph-based credit assignment for agentic reinforcement learning. arXiv preprint arXiv:2605.26684, 2026.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In *Proceedings of EMNLP*, 2023.

Fangming Cui, Ruixiao Zhu, Cheng Fang, Sunan Li, and Jiahong Li. Rethinking agentic reinforcement learning in large language models. arXiv preprint arXiv:2604.27859, 2026.

Tianyu Ding, Jianhong Xin, and Juan Pablo De la Cruz Weinstein. Keep policy gradient in charge: Sibling-guided credit distillation for long-horizon tool-use agents. arXiv preprint arXiv:2606.12634, 2026.

Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. In *Proceedings of ICLR*, 2024.

Jinwu Hu, Wei Zhang, Yufeng Wang, Yu Hu, Bin Xiao, Mingkui Tan, and Qing Du. Dynamic prompt compression for efficient inference of large language models. *IEEE Transactions on Knowledge and Data Engineering*, 38(5):2731–2744, 2026. doi: 10.1109/TKDE.2026.3669558.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of EMNLP*, 2023.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression. In *Proceedings of ACL*, 2024.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A. Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. ACON: Optimizing context compression for long-horizon LLM agents. In *Proceedings of ICML*, 2026.

Junbo Li, Peng Zhou, Rui Meng, Meet P. Vadera, Lihong Li, and Yang Li. Turn-PPO: Turn-level advantage estimation with PPO for improved multi-turn RL in agentic LLMs. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 6227–6243, 2026a. doi: 10.18653/v1/2026.findings-eacl.328.

Mo Li, L. H. Xu, Qitai Tan, Long Ma, Ting Cao, and Yunxin Liu. Sculptor: Empowering LLMs with cognitive agency via active context management. In *Proceedings of ICLR*, 2026b.

Sijia Li, Yuchen Huang, Zifan Liu, Yanping Li, Jingjing Fu, Li Zhao, Jiang Bian, Ling Zhang, Jun Zhang, and Rui Wang. GEAR: Granularity-adaptive advantage reweighting for LLM agents via self-distillation. arXiv preprint arXiv:2605.11853, 2026c.

Yuanfan Li, Qi Zhou, Wenjing Duan, and Lu Chen. When denser credit is not enough: Evidence-calibrated policy optimization for long-horizon LLM agent training. arXiv preprint arXiv:2606.05885, 2026d.

Yucheng Li, Bo Dong, Frank Guerin, and Chenghua Lin. Compressing context to enhance inference efficiency of large language models. In *Proceedings of EMNLP*, 2023.

Miao Lu, Weiwei Sun, Weihua Du, Zhan Ling, Xuesong Yao, Kang Liu, and Jiecao Chen. Beyond the context window: Scaling agentic RL via end-to-end optimized context compression. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 21074–21125, 2026a. doi: 10.18653/v1/2026.acl-long.966. URL https://aclanthology.org/2026.acl-long.966/.

Zhicong Lu, Zichuan Lin, Wei Jia, Changyuan Tian, Deheng Ye, Peiguang Li, Li Jin, Nayu Liu, Guangluan Xu, and Wei Feng. HISR: Hindsight information modulated segmental process rewards for multi-turn agentic reinforcement learning. arXiv preprint arXiv:2603.18683, 2026b.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. Learning to compress prompts with gist tokens. In *Advances in Neural Information Processing Systems*, 2023.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs from preference data. In *Proceedings of ICLR*, 2025.

OpenAI. GPT-5.5 system card. OpenAI system card, 2026. URL https://openai.com/index/gpt-5-5-system-card/.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. arXiv preprint arXiv:2310.08560, 2023.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. LLMLingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 963–981, 2024. doi: 10.18653/v1/2024.findings-acl.57.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, 2023.

Jianghao Su, Xia Zeng, Luhui Liu, Chao Luo, Ye Chen, and Zhuoran Zhuang. Enhancing agentic RL with progressive reward shaping and value-based sampling policy optimization. arXiv preprint arXiv:2512.07478, 2025.

Qwen Team. Qwen2.5 technical report. arXiv preprint arXiv:2412.15115, 2024.

Qwen Team. Qwen3 technical report. arXiv preprint arXiv:2505.09388, 2025.

Nikhil Verma. Active context compression: Autonomous memory management in LLM agents. arXiv preprint arXiv:2601.07190, 2026.

Daoyu Wang, Qingchuan Li, Mingyue Cheng, Jie Ouyang, Shuo Yu, Chunli Liu, Shijin Wang, Qi Liu, and Enhong Chen. CAPO: Critic-guided action-aligned policy optimization for advancing LLM agent capabilities. arXiv preprint arXiv:2604.18401, 2026.

Lei Wei, Xiao Peng, Tt, Guannan Zhang, Chenhao Jiang, Hongyu Li, Lanbo Lin, Yuanwu Xu, Jiayao Liu, Kesu Wang, and Bin Wang. PACE: Predictive adaptive context extraction for long-horizon LLM agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27184–27199, 2026. doi: 10.18653/v1/2026.acl-long.1252. URL https://aclanthology.org/2026.acl-long.1252/.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256, 1992.

David H. Wolpert and Kagan Tumer. Optimal payoff functions for members of collectives. *Advances in Complex Systems*, 4(2–3):265–279, 2001.

Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Xinmiao Yu, Dingchu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, Minhao Cheng, Shuai Wang, Hong Cheng, and Jingren Zhou. ReSum: Unlocking long-horizon search intelligence via context summarization. arXiv preprint arXiv:2509.13313, 2025.

Zijun Xie, Binbin Zheng, Enlei Gong, Jihua Liu, Yuyang You, Lingfeng Liu, Jiayao Tang, Guanqun Zhao, Aoqi Hu, and Zeyu Chen. ECHO: Prune to act, trace to learn with selective turn memory in agentic RL. arXiv preprint arXiv:2606.31650, 2026.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. RECOMP: Improving retrieval-augmented LMs with compression and selective augmentation. In *Proceedings of ICLR*, 2024.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z. Pan, Hinrich Schütze, Volker Tresp, and Yunpu Ma. Memory-R1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12805–12825, 2026. doi: 10.18653/v1/2026.acl-long.583.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *Proceedings of ICLR*, 2023.

Lu Yi, Runlin Lei, Liuyi Yao, Yuexiang Xie, Yuyang Li, Wenhao Zhang, Zhewei Wei, Yaliang Li, and Jian-Yun Nie. Learning agent-compatible context management for long-horizon tasks. arXiv preprint arXiv:2605.30785, 2026.

Huining Yuan, Zelai Xu, Huaijie Wang, Xiangmin Yi, Jiaxuan Gao, Xiao-Ping Zhang, Yu Wang, Chao Yu, and Yi Wu. Verifiable process rewards for agentic reasoning. arXiv preprint arXiv:2605.10325, 2026.

Weihao Zeng, Yuzhen Huang, and Junxian He. LOCA-bench: Benchmarking language agents under controllable and extreme context growth. *arXiv preprint arXiv:2602.07962*, 2026.

Haoran Zhang and Zhaohua Sun. AGORA: Adapter-grounded observation-action retention for inference-free prompt compression in LLM agents. arXiv preprint arXiv:2605.26596, 2026.

Zhen Zhang, Kaiqiang Song, Xun Wang, Yebowen Hu, Weixiang Yan, Chenyang Zhao, Henry Peng Zou, Haoyun Deng, Sathish Reddy Indurthi, Shujian Liu, Simin Ma, Xiaoyang Wang, Xin Eric Wang, and Song Wang. CM^2: Reinforcement learning with checklist rewards for multi-turn and multi-step agentic tool use. arXiv preprint arXiv:2602.12268, 2026.

TECHNICAL APPENDIX

A SCORE FUNCTION OF THE BETA POLICY

The reference implementation uses $a_{\min} = 0.05$, $a_{\max} = 1.0$, and $\kappa_0 = 8$. Thus $a_i = 0.05 + 0.95\tilde{a}_i$, where $\tilde{a}_i \in (0, 1)$ has mean $\tilde{m}_i = \sigma(\text{net}_i)$, with $\alpha_i = 8\tilde{m}_i$ and $\beta_i = 8(1 - \tilde{m}_i)$. The log-density is

$$\log \pi(a_i \mid s) = (\alpha_i - 1) \log \tilde{a}_i + (\beta_i - 1) \log(1 - \tilde{a}_i) - \log B(\alpha_i, \beta_i) - \log 0.95.$$

Differentiating $\log B$ gives digamma terms $\partial_\alpha \log B = \psi(\alpha) - \psi(\alpha + \beta)$ and $\partial_\beta \log B = \psi(\beta) - \psi(\alpha + \beta)$. Since $\partial\alpha_i/\partial\tilde{m}_i = \kappa_0$ and $\partial\beta_i/\partial\tilde{m}_i = -\kappa_0$, the two $\psi(\alpha_i + \beta_i)$ terms cancel:

$$\frac{\partial \log \pi}{\partial \tilde{m}_i} = \kappa_0 \Big[\operatorname{logit}(\tilde{a}_i) - \big(\psi(\alpha_i) - \psi(\beta_i)\big) \Big] =: s_i. \tag{A1}$$

Using the Beta log-moment identity $\mathbb{E}[\log X] = \psi(\alpha) - \psi(\alpha + \beta)$ (differentiate the normalizer under the integral), $\mathbb{E}[\operatorname{logit} \tilde{a}_i] = \psi(\alpha_i) - \psi(\beta_i)$, so $\mathbb{E}[s_i] = 0$. Zero mean makes any constant or action-independent baseline unbiased, and gives $g_i = \mathbb{E}[\text{credit}_i \, s_i] = \operatorname{Cov}(\text{credit}_i, s_i)$, the entry point for Appendix C. At evaluation, the deterministic Beta mean is mapped back to the retention interval, so the applied ratio is $a_i = 0.05 + 0.95\tilde{m}_i$.

B FULL TRAINING ALGORITHM

Algorithm 1 QUOTA training loop (condensed)

Require: Queries $\mathcal{Q}$, compressor $\mathcal{C}$, agent $\mathcal{A}$, token budget τ

Phase 0: run $\mathcal{A}$ five times per query with $r_i{=}1$; store $T_{\text{base}}(q)$ and the reference timelines
for episode $e = 1, \ldots, E$ **do**
  $\omega_e \leftarrow 0$ if $e \le 20$, else $\min\{1, (e{-}20)/10\}$
  for each compression event t of the rollout **do**
    Build output-instance slots, state s_t, and the active mask
    Sample $a_{t,i}$ from the Beta policy (Eq. 4); apply $\mathcal{C}$
    Match $N_{\text{pre},t}$ against the reference timelines; set $P_{\text{recall},t}$ (Eq. 2)
  end for
  Observe success and session tokens T; form R_{base} (Eq. 1) and the advantage A against b_q
  for each recorded event t **do**
    $\text{credit}^{\text{floor}}$ (Eq. 6); if $\omega_e{>}0$ also $\text{credit}^{\text{attr}}$ (Eq. 7)
    $\text{coeff}_{t,i}$ (Eq. 8)
  end for
  Update θ on the loss of Eq. 5
  if $e = 20$, or $e > 20$ and $(e{-}20) \bmod 5 = 0$ **then**
    Refit the outcome model f_ϕ from the replay buffer
  end if
end for
return Frozen π_{θ^*}; evaluate with its deterministic mean

Algorithms 2 and 3 specify the event-level training path. Phase 0 keep-all rollouts establish T_{base} and reference timelines, while the outcome-model replay buffer contains eligible policy-rollout events. In the reported coevolve configuration, the predictor is initialized randomly and fitted from data accumulated during the $\omega = 0$ warmup. The first fit follows the 20th completed episode, with subsequent fits every five episodes. The full-retention query $r_i{:=}1$ in Eq. 7 is therefore a learned model counterfactual; Phase 0 supplies normalization and reference windows rather than paired intervention labels.

C GRADIENT-DIRECTION INTUITION FOR ATTRIBUTION

This section explains how the two credit terms guide retention decisions. Equation 7 combines two terms with opposing effects. The immediate-saving term $\lambda_{\text{save}}(1 - r_i^{\text{req}})\text{tok}_i/T_{\text{base}}$ uses the ratio

Algorithm 2 QUOTA: Rollout and Event Recording

Require: Queries $\mathcal{Q}$, compressor $\mathcal{C}$, agent $\mathcal{A}$, per-run trigger τ on backend-estimated full $|c_t|$

Initialize policy π_θ ($174 \to 128 \to 64 \to 16$ reference MLP), random outcome model f_ϕ, and replay buffer $\mathcal{B}$ (capacity 500)
Initialize per-query EMA baselines b_q and reward histories $\mathcal{H}_q$
Phase 0: Baseline Collection
for each $q \in \mathcal{Q}$ **do**
  Run $\mathcal{A}$ five times with r_i=1; set $T_{\text{base}}(q)$ to mean total tokens and retain reference timelines
end for
Phase 1: Online Training
for episode $e = 1, \ldots, E$ **do**
  Sample query $q \sim \text{Uniform}(\mathcal{Q})$
  $\omega_e \leftarrow 0$ for $e \le 20$; otherwise $\omega_e \leftarrow \min\{1, (e-20)/10\}$
  Initialize event records $\mathcal{D} \leftarrow \emptyset$
  Run $\mathcal{A}$ on q with the compression policy
  for each compression event $t = 1, \ldots, T_c$ **do**
    Build ordered output-instance slots, state s_t, and the active mask
    $\tilde{m}_i \leftarrow \sigma(\text{MLP}_\theta(s_t)_i)$ for each tool slot i
    Sample $\tilde{a}_{t,i} \sim \text{Beta}(8\tilde{m}_i, 8(1-\tilde{m}_i))$; request $r_{t,i}^{\text{req}} \leftarrow 0.05 + 0.95\tilde{a}_{t,i}$
    Apply $\mathcal{C}$ to each instance's currently retained content with ratios $\{r_{t,i}^{\text{req}}\}$
    Observe $N_{\text{post},t}$ and matched keep-all count $N_{\text{pre},t}$ in the same event window
    $h_t \leftarrow h(N_{\text{post},t}, N_{\text{pre},t})$ using Eq. 2; set $P_{\text{recall},t} \leftarrow \lambda_{\text{recall}} h_t$
    Append $(s_t, a_t{:=}r_t^{\text{req}}, h_t, P_{\text{recall},t}, \text{active}_t, \text{event targets})$ to $\mathcal{D}$
  end for
  if any event-local recall measurement is missing **then**
    Skip the update rather than treating missing recall as zero
  end if
  Observe success $\in \{0, 1\}$ and total session tokens T
  $P_{\text{recall}} \leftarrow \sum_{t=1}^{T_c} P_{\text{recall},t}$ (Eq. 3)
  $R_{\text{base},T_c} \leftarrow \text{success} - \lambda_{\text{tok}} \,\text{clamp}(T/T_{\text{base}}, 0.2, 2)$; $R_{\text{base},k<T_c} \leftarrow 0$ (Eq. 1)
  $\sigma_q \leftarrow 1$ if $|\mathcal{H}_q| < 3$, else $\max(\text{std}(\mathcal{H}_q[-20:]), 0.01)$
  $A \leftarrow (R_{\text{base},T_c} - b_q)/\sigma_q$; update b_q with decay 0.9 and append R_{base,T_c} to $\mathcal{H}_q$
  Pass $(e, \omega_e, \mathcal{D}, A, T_c, T_{\text{base}})$ to Algorithm 3
end for
return Frozen policy π_{θ^*}; use its deterministic mean for evaluation

sampled and requested by the policy; it grows as requested retention decreases and therefore rewards compression. The counterfactual downstream-cost term is $[\hat{C}(s,a) - \hat{C}(s,a; r_i^{\text{req}}{:=}1)]_+$, so only predicted harm relative to fully retaining tool i is subtracted. Predicted negative cost differences do not create an extra reward. The net credit favors compression for low-consequence outputs and protects outputs with predicted downstream cost.

The learned counterfactual serves as an empirically evaluated credit signal, without a formal causal guarantee. Its training contribution and predictive agreement are assessed by the attribution ablation in Table 2 and the single-tool interventional rollouts in Table 3.

C.1 Counterfactual Estimation and Training Use

Objective and shaping roles Two properties of Eq. 8 matter for reading the weights in Table A2. The outcome token head predicts total agent tokens, including the token consequences of current retention, so it is a level rather than a residual recovery-only target. And the positive-part cost difference charges predicted excess cost, while the measured floor prices observed excess calls, so the two channels price different quantities and carry separate training weights. Appendix D makes the resulting update and its relation to J explicit.

Role of the learned counterfactual The predictor learns from event records collected under the evolving stochastic retention policy. Beta exploration varies retention ratios across visited states; the full-retention query $r_i = 1$ evaluates the endpoint of this action interval. Accurate estimates require

Algorithm 3 QUOTA: Policy and Outcome-Model Updates

Require: Episode e, attribution weight ω_e, records $\mathcal{D}$, advantage A, event count T_c, baseline T_{base}

for each event record $t \in \mathcal{D}$ **do**
  $A_t \leftarrow \gamma_{\text{seq}}^{T_c - t} A = A$ with $\gamma_{\text{seq}} = 1$
  $\text{credit}_{t,i}^{\text{floor}} \leftarrow -P_{\text{recall},t}/|\text{active}_t|$ for $i \in \text{active}_t$ (Eq. 6)
  $\text{credit}_{t,i}^{\text{attr}} \leftarrow 0$
  if $\omega_e > 0$ **then**
    $\hat{C}(s_t, a_t) \leftarrow \lambda_N \sum_j \hat{n}_j + 2\lambda_T \tanh\left(\frac{\exp \hat{T}_{\log}}{2T_{\text{base}}}\right)$
    $\Delta\hat{C}_{t,i} \leftarrow \max\{0, \hat{C}(s_t, a_t) - \hat{C}(s_t, a_t; r_{t,i}^{\text{req}} := 1)\}$
    $\text{credit}_{t,i}^{\text{attr}} \leftarrow \lambda_{\text{save}}(1 - r_{t,i}^{\text{req}}) \frac{\text{tok}_{t,i}}{T_{\text{base}}} - \Delta\hat{C}_{t,i}$ (Eq. 7)
  end if
  $\text{coeff}_{t,i} \leftarrow \lambda_A A_t + \text{credit}_{t,i}^{\text{floor}} + \omega_e \, \text{credit}_{t,i}^{\text{attr}}$ (Eq. 8)
end for
Policy Update (shaped surrogate):
$\mathcal{L} \leftarrow -T_c^{-1} \sum_t \sum_{i \in \text{active}_t} \text{coeff}_{t,i} \log \pi_\theta(a_{t,i} \mid s_t)$
Adam update of θ (LR 3×10^{-4}; gradient norm clipped at 1)
Add eligible event records with propensity and outcomes to $\mathcal{B}$; mark episode e completed
if $e = 20$ or ($e > 20$ and $(e - 20) \bmod 5 = 0$) **then**
  Update f_ϕ for 10 steps from batches of 16 using the multi-task loss in Appendix G
  At $e = 20$, fail the coevolve run if eligible labels are insufficient or the first fit fails
end if
Every 10 episodes, retain the checkpoint with the best rolling-10 mean reward

informative state features and generalization from the observed state–action coverage, so unobserved trajectory factors and sparsely covered regions remain relevant to prediction quality. The five keep-all runs provide token normalization and recall windows, while predictor fitting uses the policy-event buffer (Algorithm 2). After warmup, the learned signal supplements measured episode returns and the uniform floor, which remain active throughout optimization. The held-out intervention study directly evaluates endpoint predictions, and Appendix D specifies the shaped-gradient formulation.

The schedule controls how this channel enters learning: 20 episodes of measured-feedback training, then activation of the attribution coefficient over 10 episodes, with predictor fitting beginning after episode 20 and refreshing every five episodes from a replay buffer of 500 event records. This provides initial data and periodic adaptation to the changing policy. Two complementary tests support the channel: the held-out intervention test evaluates the full-retention endpoint directly, and the RL-only ablation measures the net training benefit of adding it. They address different questions and are reported separately.

D Sequential Formulation and Worked Example

D.1 Repeated-Compression Example

Two-event example Consider an illustrative session with two output instances: a 1,000-token SQL result i and a 200-token date lookup j. At event 1, slots (i, j) receive ratios $(0.6, 1)$, leaving approximately $(600, 200)$ tokens. Between events, another SQL call returns a new output k of 800 tokens; i and k share a tool type but remain distinct instances. At event 2, active slots (i, j, k) receive $(0.5, 1, 0.75)$, leaving approximately $(300, 200, 600)$ tokens. Setting $r_{2,i} = 1$ instead would retain the current 600 tokens of i, not its original 1,000 tokens. If an earlier-removed field is needed, recovery requires an additional information source, such as a new tool invocation. These sizes assume the backend meets each quota and illustrate the sequential semantics rather than an additional measured trajectory.

D.2 Sequential Formulation

With events $t = 1, \ldots, T_c$, the main text defines the base reward only at the terminal event. With the reference setting $\gamma_{\text{seq}} = 1$, the terminal reward gives the same undiscounted return at each

compression event. The undiscounted base return and normalized advantage used by the shaped surrogate are

$$G_t^{\text{base}} = \sum_{k=t}^{T_c} R_{\text{base},k} = R_{\text{base},T_c},$$
$$A_t = \frac{G_t^{\text{base}} - b_q}{\sigma_q},$$
$$g_\theta^{\text{shaped}} = \mathbb{E}\Big[T_c^{-1} \sum_t \sum_{i \in \text{active}_t} \text{coeff}_{i,t} \nabla_\theta \log \pi_\theta(a_{t,i} \mid s_t)\Big].$$

Because λ_A rescales A_t and $\text{credit}_{i,t}^{\text{attr}}$ adds model-based shaping, g_θ^{shaped} is the expected update direction of Eqs. 5 and 8, not an unbiased estimator of $\nabla_\theta J$ for Eq. 3. Event averaging, query-dependent scaling, the event-local floor, and auxiliary credit define the surrogate. The floor conserves the scalar penalty; its gradient contribution additionally depends on the distinct score term of each slot.

The measured recall component is computed locally at each compression event. $P_{\text{recall},t}$ enters the credit at event t and is not replaced by a session-level count. Reference windows replay the same compression-trigger boundaries over five keep-all timelines, matching by event order; $N_{\text{pre},t}$ is their mean call count.

To implement this update, every event is logged separately and the active mask restricts the sum of log-probabilities to its active output slots. The running scale σ_q uses the most recent 20 rewards after at least three observations, with a floor of 0.01 (Algorithm 2; Table A2).

Combining local and downstream token signals. Let $u_i = \lambda_{\text{save}}(1 - r_i)\text{tok}_i/T_{\text{base}}$ and $d_i = \hat{C}(s,a) - \hat{C}(s,a;r_i := 1)$. The auxiliary credit is $u_i - [d_i]_+$. Since $\hat{C}$ contains a total-token predictor, d_i already reflects both retained-context usage and later recovery. If $d_i < 0$, rectification suppresses that predicted cost benefit rather than turning it into an additional positive bonus; the local preference u_i remains. If $d_i > 0$, it is compared with u_i to balance the compression preference against predicted excess cost. The sum $\sum_i (u_i - [d_i]_+)$ therefore shapes the update rather than partitioning $R_{\text{base}} - P_{\text{recall}}$; Table 2 evaluates the full auxiliary channel under the reported weights.

E Provenance-Based Attribution

This appendix specifies the provenance upgrade referenced in the main text. It replaces the proximal time-window proxy with content-based source matching: each removed span σ at event t is tagged with $\langle t, i(\sigma), \phi(\sigma)\rangle$, where ϕ is a content signature, and a later tool call u is matched to an earlier removed span:

$$\text{match}(u) = \underset{\sigma:\, \text{time}(\sigma) < \text{time}(u)}{\arg\max} \text{sim}(u, \phi(\sigma)) \quad \text{if } \max \text{sim} \geq \tau_{\text{prov}}, \text{ else } \varnothing.$$

The resulting count $n_{t,i}^{\text{prov}}$ sums matched costs and defines a provenance-based penalty and targeted floor:

$$\text{credit}_i^{\text{floor-prov}} = -P_{\text{recall},t}^{\text{prov}} \cdot \frac{n_{t,i}^{\text{prov}}}{\sum_j n_{t,j}^{\text{prov}}},$$
$$\sum_i \text{credit}_i^{\text{floor-prov}} = -P_{\text{recall},t}^{\text{prov}}.$$

When $\sum_j n_{t,j}^{\text{prov}} > 0$, the floor conserves the penalty by construction and assigns credit through matched provenance rather than temporal proximity alone. Match precision is an empirical property of the signature and threshold, so a deployment should calibrate hit and mismatch rates on labeled calls. The reported experiments and Algorithm 2 use the window-based signal of Eq. 2.

F Feature Definitions

QUOTA assigns retention ratios to output instances, with a configurable slot capacity S for the policy representation. Using the feature blocks below, the state is

$$s_t = [q_1, \ldots, q_7;\ o_{1,1}, \ldots, o_{S,10};\ c_1, \ldots, c_7] \in \mathbb{R}^{14+10S}.$$

The reference configuration uses $S = 16$, giving the 174-dimensional state reported in the main text. This is an implementation choice rather than an intrinsic limit of the allocation method. For a deployment with a different capacity, the MLP is configured and trained with $14 + 10S$ input dimensions and S output means; the retention objective and per-output credit construction remain the same.

At each compression event, the active instance count can vary within the configured capacity. Slots represent ordered tool-output instances: repeated calls to the same tool retain distinct identifiers, ratios, and gradients, while unused positions are zero-padded and masked. Thus, the representation accommodates varying numbers and combinations of tool outputs without assigning permanent slots to particular tool types. The reference implementation checks the configured capacity and rejects over-capacity events; supporting larger events requires a correspondingly configured policy, rather than silently merging or discarding outputs.

Table A1: Reference implementation feature specification. Runtime annotations use only values available in the trajectory prefix; unavailable annotations default to zero or the stated prior.

Block	Dim.	Meaning
Query	q_1	$0.3\log(1 + \lvert q\rvert/2)$, a token-length proxy.
	q_2	Numeric-constraint indicator.
	q_3	Multi-hop/comparison indicator.
	q_4–q_7	Mutually exclusive retrieval, SQL/aggregation, report, and other intents.
Output i	$o_{i,1}$	Presence flag.
	$o_{i,2}$	$0.1\log(1 + \text{tok}_i)$, output-size feature.
	$o_{i,3}$	Output-token share within the current event.
	$o_{i,4}$	$0.1\log(1 + \text{same-type tokens})$ within the event.
	$o_{i,5}$	$\log(1 + \text{same-type ordinal})$ for repeated calls.
	$o_{i,6}$	Query relevance; fallback static criticality prior high/medium/low $= 1.0/0.5/0.2$.
	$o_{i,7}$	Prefix-available downstream-reuse annotation, else zero.
	$o_{i,8}$	Prefix-available redundancy annotation, else zero.
	$o_{i,9}$	Normalized event-slot recency, from oldest 0 to newest 1.
	$o_{i,10}$	Prefix rework annotation when available, else zero.
Context	c_1–c_5	System, instruction/tool-definition, dialogue-history, tool-output, and scratch-pad fractions.
	c_6	Context pressure relative to compression threshold τ.
	c_7	Normalized compression-event index.

The outcome model reuses the ten dimensions for each output and appends the applied retention ratio as an eleventh dimension. The reference tensor provides $S = 16$ output positions, of which the K active instances described in the main text participate through the presence mask. A separate seven-dimensional query token is projected into the same 64-dimensional embedding space. It predicts $\hat{n}_i$, $\hat{T}_{\log}$, and the auxiliary success probability $\hat{p}$; only $\hat{n}_i$ and $\hat{T}_{\log}$ enter $\hat{C}$.

G Reported Baselines and Experimental Configuration

G.1 Baseline Strategy Formulas

We detail the baseline definitions and allocation rules used in Section 4.1.

Keep-all. $r_i = 1$ for all tools. Serves as the no-compression reference and token baseline (T_{base}).

Uniform-0.5. $r_i = 0.5$ for all tools, regardless of type, size, or position.

Recency. The ratio decays linearly from the most recent tool output (highest ratio) to the oldest:

$$r_i = 0.1 + 0.8 \cdot \frac{\text{pos}_i - 1}{N - 1},$$

where $\text{pos}_i \in \{1, \ldots, N\}$ is the position of tool i sorted by invocation time (most recent = N). This yields $r = 0.9$ for the most recent and $r = 0.1$ for the oldest. For the single-output case $N = 1$, the reference implementation uses the midpoint $r = 0.5$.

Token-proportional. For output-size estimate size_i,

$$r_i = 1 - 0.7 \frac{\text{size}_i}{\max_j \text{size}_j}.$$

The largest output receives 0.3; smaller outputs approach 1.

Tool-type-conditional. The released search uses greedy coordinate descent over $\{0.2, 0.3, \ldots, 0.9\}$, visiting canonical tool types in decreasing mean-output-size order. Each coordinate maximizes success rate minus 0.3 times token ratio on training queries while holding other coordinates fixed; the default script runs two passes.

Comparison design Table 1 compares the full policy with static allocations under a shared compressor. Table 2 isolates the auxiliary credit channel on the same held-out query–seed records, and Table 3 tests its predictions through real single-output interventions. Tables 4, 5, and 6 examine backbone transfer, compressor transfer, and training on disjoint public environments, respectively. The AGORA-inspired experiment changes the compressor backend; it is not a head-to-head evaluation against the original AGORA policy.

G.2 Reference Implementation Configuration

Table A2 gives the released reference configuration. Slot capacity, network dimensions, and optimization settings describe this instantiation; Appendix F explains capacity configuration. Reproducing a particular run additionally requires its command-line and environment overrides; the table provides defaults rather than a per-table run manifest.

Table A2: Reference implementation defaults.

Group	Setting	Value
Reward	$\lambda_{\text{tok}}, \lambda_{\text{recall}}, \lambda_{\text{save}}$	$0.3, 0.2, 0.3$
	$\lambda_N, \lambda_T, \lambda_A$	$0.1, 0.3, 0.5$
	γ_{seq}	1.0 (no event discount)
Policy	Architecture	174→128→64→16, ReLU
	Action interval; κ_0	$[0.05, 1.0]$; 8.0
	Optimizer; LR; grad clip	Adam; 3×10^{-4}; 1.0
Baseline	EMA decay	0.9
	Scale estimate	Last 20 rewards after 3 samples; floor 0.01
Outcome	Architecture	$d = 64$, 2 layers, 4 heads, FF 128, dropout 0.1
	Inputs	16 output tokens × 11 dims; one 7-dim query token
	Optimizer; LR	Adam; 10^{-3}
	Loss weights (w_n, w_T, w_s)	$(1.0, 1.0, 0.3)$
	Monotonicity weight	$\lambda_{\text{mono}} = 0.5$
	Batch; replay capacity	16; 500 event records
	Refresh work	10 gradient steps per refresh
	EMA anchor	Decay 0.995 after warmup
Schedule	Warmup; annealing; refresh	20 episodes; 10 episodes; at 20 completed episodes, then every 5
Environment	Compression threshold	Caller-supplied per run; backend-estimated full $\lvert c_t \rvert$
	Keep-all reference	5 runs per query
	Training length; seeds	100 episodes; $\{0, 1, 2, 3, 4\}$

The outcome loss is

$$\begin{aligned}\mathcal{L}_\phi = \; & \text{MSE}(\hat{n}, n) + \text{MSE}(\hat{T}_{\log}, \log T) \\ & + 0.3\,\text{BCE}(\hat{p}, \text{success}) + 0.5\Omega_{\text{mono}},\end{aligned}$$

with count and token predictions and targets standardized by running statistics. The auxiliary success head is trained by this loss but is omitted from $\hat{C}$. The released updater can additionally apply propensity weighting and an EMA anchor; these are implementation stabilizers rather than new terms in the paper objective.

G.3 Computing Infrastructure and Runtime

The reference environment used a single Intel Xeon Platinum 8369B server (32 CPU cores, 128 GB RAM) running a Linux distribution with kernel 5.10. The software stack was Python 3.12.4, PyTorch 2.3.1, NumPy 1.26.4, SciPy 1.13.0, Flask 3.0.3, and Transformers 4.42.0. The policy and outcome model together contain 102,306 trainable parameters (31,696 and 70,610, respectively), approximately 102K, and run on CPU; external agent and compressor-model calls dominate wall-clock cost.

The production agent backbone is Claude Sonnet 4.6 with a 200K-token context window. Truncation is local string slicing, Self-Info uses a local Qwen-2.5-0.5B clause scorer, and summarization uses Qwen-3.7-Max through an API; the AGORA-inspired backend combines step/clause segmentation, self-information scoring, and a numeric-content floor. A rollout takes roughly 30–180 seconds depending on query and backend. A 100-episode, five-seed run for one compressor takes about 25–42 hours. The broader experimental campaign was approximately 400 compute-hours and $2,800 in backbone API charges.

G.4 Outcome Model Held-Out Accuracy

To complement the intervention evaluation in Section 4.4, we report two factual prediction metrics on held-out outcomes: Spearman $\rho = 0.72$ for per-output re-invocation counts and Brier score 0.07 for success probability $\hat{p}$. These assess prediction on observed outcomes; the single-output interventions separately assess predicted counterfactual cost differences.

That head is auxiliary: it is excluded from the downstream-cost definition $\hat{C}(s,a) = \lambda_N \sum_j \hat{n}_j + 2\lambda_T \tanh\Big(\exp(\hat{T}_{\log})/(2T_{\text{base}})\Big)$ in the main text.

G.5 LOCA-bench Training Configuration

Section 4.6 and Table 6 report the LOCA-bench experiment. A fresh QUOTA policy is trained for five rounds of iterative RL on 10 of the benchmark's 15 task environments and evaluated on the other 5. Evaluation aggregates the 96K, 128K, and 256K context-window settings over 5 seeds.

H Difficulty Tiers and Evaluation Protocol

The full production-query corpus contains 120 queries in three tiers:

- 25 simple lookups;
- 55 multi-step queries combining schema retrieval with SQL; and
- 40 multi-table permission-gated joins.

The stratified 80/40 split leaves 7 simple, 19 multi-step, and 14 multi-table queries in the held-out set. Held-out evaluation runs frozen deterministic policies on these 40 queries. Table 1 reports 29%–46% savings aggregated across this held-out mixture; the reported estimand is the overall saving across difficulty tiers.

I Statistical Procedures and Metric Interpretation

Each RL configuration trains for 100 episodes per seed with five seeds. Held-out inference uses the effective sample size obtained by aligning method records on their observation keys and applying the metric-specific validity filter. This separates the training schedule from the evaluation sample

size. The analysis reports the paired tests and confidence intervals below, without a post-hoc power calculation.

I.1 Reported Recall Statistic

The uppercase $P_{\text{recall},t} = \lambda_{\text{recall}} h_t$ in Eq. 2 is a training penalty, where the event-local rate is

$$h_t^{(e)} = h\Big(N_{\text{post},t}^{(e)}, N_{\text{pre},t}^{(e)}\Big),$$

For the set $\mathcal{E}$ of valid query–seed evaluation records, the lowercase table statistic first averages events within each record and then averages the resulting record-level rates:

$$p_{\text{recall}} = \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \frac{1}{T_c^{(e)}} \sum_{t=1}^{T_c^{(e)}} h_t^{(e)}.$$

Here h is the complete piecewise definition in Eq. 2, including zero-reference cases; missing measurements are excluded from valid records. Thus p_{recall} is an event-local rate, not a probability or a training penalty: it does not include λ_{recall} and does not sum across events. Training curves apply the stated rolling-window mean to the corresponding record-level rates.

I.2 Confidence Interval Construction

Method-comparison token summaries use the interquartile mean (IQM) with 10,000 stratified bootstrap resamples. Observations are grouped by difficulty tier; within each resample, observations are sampled with replacement inside each tier while preserving the original tier sizes. The 2.5th and 97.5th percentiles of the bootstrap IQM distribution form the 95% interval. If no stratification variable is supplied, the implementation uses a single stratum rather than treating each observation as its own stratum.

Table 3 uses a different construction for counterfactual validation: each cell center is the metric's native point estimate, and percentile-bootstrap resampling treats each held-out state as a cluster so that paired interventions from the same state remain together. The reported $\pm$ value is the half-width of the resulting 95% interval. These intervals are not IQM intervals.

I.3 Hypothesis Testing

Method comparisons use a two-sided paired Wilcoxon signed-rank test. Records are aligned on the common explicit observation key—normally $(\text{query id}, \text{seed}, \text{episode})$ for training histories and $(\text{query id}, \text{seed})$ for one-shot held-out runs—before testing; unmatched records are not positionally paired.

The attribution ablation tests exactly one quantity: the per-record normalized token usage $z_{m,e}$ defined below, paired between QUOTA and RL-only over the matched record set $\mathcal{E}_{\text{paired}}$. Each compressor is tested separately, and the three tests are treated as a single Holm–Bonferroni family:

Table A3: Raw and Holm-adjusted p-values for the attribution ablation.

Compressor	**Raw** p	**Holm-adjusted** p
Truncation	0.019	0.019
Summarization	0.003	0.009
Self-Info	0.006	0.012

The pooled test over all matched records from the three compressors gives $p = 0.008$; it summarizes the same comparison and is outside the Holm family.

Success rates summarize the valid evaluation records, with their effective sample sizes supplying the denominators. Statistical testing targets paired token usage; success-rate differences in the main comparison and attribution ablation are descriptive estimates, without significance claims.

Token-ratio estimand in the attribution ablation. For each compressor, align QUOTA and RL-only evaluation records by $e = (\text{query id}, \text{seed})$, and let $\mathcal{E}_{\text{paired}}$ be the keys holding a valid record in both arms. For each arm m, Table 2 summarizes the normalized token usage $z_{m,e} = T_{m,e}/T_{\text{keep-all},e}$ over this set:

$$r_m = \text{IQM}\{z_{m,e} : e \in \mathcal{E}_{\text{paired}}\}.$$

No success conditioning is applied at any point: a record enters the token summary whether or not the task succeeded, and the paired Wilcoxon test aligns the two arms on these same keys. Success rates are descriptive proportions over the same records.

Table 2 therefore uses the same estimand as Table 1 and differs only in the comparison arm. Its QUOTA rows summarize the same held-out runs as Table 1, which is why the two tables print identical token ratios, excess-call rates, and success rates for QUOTA.

I.4 Reproducibility of Randomness

Production-query RL runs use seeds $\{0, 1, 2, 3, 4\}$. Each run seeds Python `random`, NumPy, and PyTorch before query sampling, policy initialization, Beta exploration, and replay-buffer sampling; output names carry the seed suffix. Frozen evaluation uses the deterministic policy mean. The LOCA launcher additionally propagates its selected seed through `PYTHONHASHSEED`. These controls reproduce the released program's stochastic choices; remote model responses and production-backend state remain additional sources of run-to-run variation.

J Tool Semantics and Protected Retention

The reported experiments use repeatable retrieval tools, for which fetching an output again gives p_{recall} a direct recovery interpretation. Deployments with writes, purchases, or other state-changing operations additionally require operation-specific handling: repeated execution can change external state, with consequences beyond token usage.

A protected-retention extension marks a set $\mathcal{P}_t$ of outputs or required audit fields and imposes $r_{t,i} = 1$ for $i \in \mathcal{P}_t$, while the policy allocates quotas to the remaining slots. Applying protection from the first eligible event preserves the required information throughout the session; later full retention preserves the content then available. This specifies a deployment extension for state-changing tools, whose evaluation is left to future work.

K Extended Related Work

Learned pruning and context management Token-level methods select which text survives compression using perplexity or learned token labels (Jiang et al., 2023; 2024; Pan et al., 2024); Selective Context scores information units (Li et al., 2023). LLM-DPC learns sequential token removal with a compression–quality reward (Hu et al., 2026), while AGORA scores observation–action steps using counterfactual next-action-change supervision (Zhang & Sun, 2026). A parallel line compresses context into learned representations rather than surviving text: AutoCompressors train summary vectors (Chevalier et al., 2023), gist tokens distil instructions into a short prefix (Mu et al., 2023), and in-context autoencoders encode long inputs into compact slots (Ge et al., 2024). These methods change the representation presented to the agent. QUOTA instead learns retention quotas for individual tool-output instances from downstream feedback, while a fixed compressor determines which content realizes each quota.

Context managers also adapt interaction history. PACE adjusts history granularity using predicted next-action relevance (Wei et al., 2026), while ACON optimizes compression guidelines from agent failures (Kang et al., 2026). Other managers provide explicit history operations: Focus consolidates knowledge and prunes history (Verma, 2026), Sculptor exposes fragmentation, summary, hide/restore, and search (Li et al., 2026b), and MemGPT manages tiered memory (Packer et al., 2023). Related work studies what to carry across turns through long-horizon summarization in ReSum (Wu et al., 2025), RL-based memory management in Memory-R1 (Yan et al., 2026), and selective turn pruning in ECHO (Xie et al., 2026).

Within this broader context-management setting, SUPO jointly optimizes summarization and tool-use behavior (Lu et al., 2026a), while AdaCoM learns an external context manager for a frozen agent (Yi et al., 2026). AdaCoM is especially close to QUOTA's deployment setting. QUOTA additionally keeps the compressor frozen and learns numerical quotas from task success, agent tokens, and excess calls, separating output-level allocation from text editing or summary generation.

Adaptive routing and allocation Learned routers such as RouteLLM select among models to balance response quality and inference cost (Ong et al., 2025). QUOTA shares this resource-allocation motivation, but its action distributes retention across outputs already in the session rather than selecting an execution model. Routing, context-management operations, and per-output quotas therefore act at different decision levels. Table A4 compares their action spaces and feedback; the distinction is a task-specific control design built on standard REINFORCE.

RL for language agents and credit assignment Prior RL methods for language agents assign credit at the step or turn level (Wang et al., 2026; Li et al., 2026a;d), exploit episode structure via graph decomposition or process rewards (Cheng et al., 2026; Lu et al., 2026b; Su et al., 2025; Zhang et al., 2026; Yuan et al., 2026; Cui et al., 2026), reweight or distil advantages across granularities and siblings (Li et al., 2026c; Ding et al., 2026), or penalize redundant tool calls (Cao et al., 2026; Chen et al., 2026). All optimize the agent's action policy. Our setting is the converse: we freeze the action policy and train only the context policy, using excess tool calls as a sparse credit signal.

To densify this signal, we adopt counterfactual credit assignment (Foerster et al., 2018; Wolpert & Tumer, 2001; Arjona-Medina et al., 2019; Chen et al., 2025), which isolates one decision's contribution by comparing realized and alternative outcomes. We instantiate this at the level of individual tool outputs: the outcome model compares a selected retention quota with fully retaining that output, holding the other quotas fixed.

Table A4: Conceptual comparison of decision spaces and training feedback across method families.

Approach	Decision or output	Feedback / learning target
LLM-DPC (Hu et al., 2026)	Sequential removal of prompt tokens	Compression rate, output quality, and key-information retention
AGORA (Zhang & Sun, 2026)	Retention of observation–action steps	Counterfactual next-action-change labels for a relevance scorer
PACE (Wei et al., 2026)	Historical-memory granularity	Predicted relevance for the next action
ACON (Kang et al., 2026)	Compression guidelines for observations and history	Agent-failure analysis in natural-language optimization
AdaCoM (Yi et al., 2026)	Flexible context modifications by an external LLM	End-to-end RL with a frozen task agent
SUPO (Lu et al., 2026a)	Summaries and tool-use behavior	Joint end-to-end policy optimization
RouteLLM (Ong et al., 2025)	Choice between stronger and weaker models	Preference-based quality–cost routing
QUOTA	Continuous quota per tool-output instance; fixed agent and compressor	Measured task success, agent tokens, excess calls, and learned single-output credit

L Deployment Cost Accounting and Break-Even Scenarios

Training adds an up-front cost. Whether agent token savings translate into recurring monetary savings depends on compressor costs and billing. For a deployment with training cost C_{train}, average keep-all tokens T_0, observed token ratio r, token price p, and daily volume V, a simple scenario calculation is

$$d_{\text{break-even}} = \frac{C_{\text{train}}}{V\,T_0(1-r)p}.$$

An illustrative input-token-only scenario uses the conservative full-campaign backbone-API expenditure $C_{\text{train}} = \$2{,}800$, $T_0 = 85{,}000$ tokens, the Summarization ratio $r = 0.541$, an input-token

price $p = \$3/10^6$ tokens, and $V = 1{,}000$ sessions/day. It gives

$$\begin{aligned} \Delta T_{\text{session}} &= 85{,}000(1 - 0.541) = 39{,}015 \text{ tokens}, \\ s_{\text{session}} &= 39{,}015 \times \$3/10^6 \approx \$0.117, \\ d_{\text{break-even}} &= \$2{,}800/(1{,}000 \times \$0.117) \\ &\approx 24 \text{ days} \approx 3.4 \text{ weeks}. \end{aligned}$$

Sensitivity to the billed input-token price can be assessed within this scenario. Holding the other quantities fixed, an effective billed input-token price of approximately \$1.7 per million gives a 42-day (six-week) break-even, while \$3 per million gives the 24-day point above; the spread reflects price sensitivity.

This scenario isolates the value of input-token reduction. Deployment accounting additionally considers summarizer calls, Self-Info scoring, output-token prices, policy and server costs, retries, and operational overhead. Latency, integration and maintenance, workload shifts, and the value of task success are further deployment-specific factors outside this calculation. The following formulation relates billed net savings to observed traffic.

Compressor-inclusive accounting. For a session, let C_A denote billed agent input and output tokens (including recovery interactions), C_C the summarizer/API and local scorer cost, and C_P policy-serving cost. With the keep-all counterpart $C_{A,0}$, the net inference saving is

$$\Delta C = C_{A,0} - (C_A + C_C + C_P).$$

Compressor-inclusive savings require $C_C + C_P < C_{A,0} - C_A$. If measured average $\Delta C > 0$, training amortizes after $C_{\text{train}}/(V\Delta C)$ days at volume V; otherwise this calculation has no finite payback. In the illustrative input-token-only scenario above, the gross budget for omitted overhead is approximately \$0.117 per session, before output-token pricing and other cost differences. Apply each model's input/output prices separately when computing these costs; the reported token tables quantify agent usage, while this equation specifies the corresponding deployment accounting.

M Released Resources and Reproduction Paths

The accompanying `code_supplement` contains the reference Beta policy, feature construction, outcome model, event-level training loop, policy server, compressor interfaces, reward and credit functions, statistical utilities, and API-adapter boundary. It also contains protocol scripts for fixed baselines and tool-type grid search, held-out evaluation, single-output interventions, cross-backbone transfer, AGORA-style evaluation, and LOCA-bench evaluation; offline unit tests; pinned Python dependencies; and an anonymized query-schema file.

The anonymized query-schema file preserves the nominal 120-query composition, difficulty labels, and 80/40 split, including the 7/19/14 held-out stratification. It documents task structure and split construction. Targets redacted for release are marked unscorable by the loaders, keeping unavailable labels separate from measured failures. Exact production evaluation uses the private verified targets.

The release separates reusable method components from environment-specific assets. The proprietary production agent, warehouse and knowledge-base contents, raw prompts and tool outputs, event logs and trajectories, trained checkpoints, and raw matched table inputs are outside the distributed package. LOCA-bench and commercial model endpoints are obtained separately. The `api_client.py` contract provides the integration interface; evaluation commands check that the required backend, checkpoint, trajectory, or benchmark artifacts are available before execution.

The release supports three complementary reproduction paths. First, the policy, feature builder, reward/credit functions, and offline tests expose the numerical method without production access. Second, a compatible agent backend can implement the `api_client.py` contract and run the baseline, training, held-out evaluation, and intervention protocols. Third, LOCA-bench provides the public task route: configure the benchmark and model endpoints, train on the designated 10 environments, and evaluate on the remaining 5 at 96K, 128K, and 256K context settings over five seeds. The production schema documents task composition and split construction; exact replay additionally requires the private queries, verified targets, and backend state.

N BEHAVIOR AND TRANSFER DETAILS

N.1 LEARNED RATIOS AND BACKBONE TRANSFER

Per-tool learned ratios Figure 5 shows converged per-tool ratios. Under summarization, `execute_sql` has the highest retention (0.65), while `query_date` has the lowest (0.48). This ordering is consistent with protecting data-bearing outputs that are expensive to recover. Under truncation, the reported range is narrower (0.52–0.60). Together, these allocations illustrate how the learned retention profile varies with output type and compressor backend.

Cross-backbone transfer To test whether the learned retention structure depends on the training backbone, we freeze the policy trained on Claude Sonnet 4.6 and evaluate it zero-shot on Qwen-3.7-Max (Team, 2025) and GPT-5.5 (OpenAI, 2026), cited as in Section 4.1. Table 4 reports results:

Across the two alternative backbones, the frozen policy achieves 18–40% agent token savings, with observed success rates of 0.69–0.78 against keep-all success of 0.71 on Qwen-3.7-Max and 0.74 on GPT-5.5. Relative to keep-all, success differs by -2 percentage points under Self-Info and $+4$ under summarization on both backbones. These results show positive token savings across the evaluated model families without policy retraining, at success rates that track the corresponding keep-all reference.

Deployment cost analysis. The reported token ratios quantify agent-side efficiency, with summarization yielding approximately 46% lower agent-token usage. Deployment expenditure additionally includes Qwen-3.7-Max summarization calls or local Self-Info scoring, as applicable. Appendix L combines these components in a net-cost formulation and illustrates how workload volume and token prices affect training amortization.

N.2 TRANSFER TO AN AGORA-INSPIRED COMPRESSOR

Does a policy learned on one compressor transfer to a structurally different one?

The three backends above share clause- or position-level granularity. The AGORA-style (Zhang & Sun, 2026) step-level compressor is structurally different: it chains clause segmentation, self-information scoring, an always-keep floor for numeric content, and greedy budget fill. We therefore evaluate QUOTA on this backend, both zero-shot and with backend-specific training.

Table 5 shows that Uniform-0.25 increases agent tokens by 21% relative to keep-all while gaining one percentage point of success over it. The policy trained on truncation and applied zero-shot achieves 31% savings at success 0.62, two percentage points below keep-all. Training directly on the AGORA-inspired backend yields 34% savings at success 0.65, one percentage point above keep-all and equal to Uniform-0.25. Target-specific training improves on the zero-shot policy in both success and token efficiency, closing the 3-percentage-point success gap to Uniform-0.25.

The AGORA-inspired implementation tests transfer to a compressor with a different allocation structure, complementing the cross-backbone evaluation.