KNOWLEDGE HAS NO FIXED HOME: DYNAMIC VERIFIED PLACEMENT ACROSS EXTERNAL AND PARAMETRIC MEMORY

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-evolving agents must decide whether acquired knowledge should be maintained externally or internalized in model parameters. Neither choice is universally preferable: their value depends on future use, updates, executor costs, and reliability. We formulate *dynamic knowledge placement* and show that preferred representations change with lifecycle conditions, earlier decisions can expire, and average executor success hides knowledge-specific risk. Our method, DYNAMIC VERIFIED PLACEMENT (DVP), filters executors by capability and risk, compares their expected lifecycle value, revisits earlier placements, and verifies new representations before use. On LightShop, external and parametric memory solve complementary tasks. Under fixed budgets, DVP completes 23.78 and 63.32 more correct tasks than always external on the primary and shifted workloads. It also completes 3.80 and 9.38 more tasks than a matched adaptive-amortization policy that uses the same forecasts, executor cards, and transaction protocol but optimizes only serving-time savings. Under the exact benchmark verifier, no incorrect internal output was released.

1 INTRODUCTION

AI agents are increasingly expected to improve through long-term interaction rather than remain static after deployment. As they execute tasks, use tools, receive feedback, and encounter failures, they accumulate experience that can inform future behavior (Shinn et al., 2023; Zhao et al., 2024; Xia et al., 2026). A central challenge in such self-evolving agents is therefore not only *what* to learn, but also *where* the resulting knowledge should reside. We formulate this problem as **dynamic knowledge placement**: deciding whether future uses of each piece of acquired knowledge should rely on external memory, model parameters, or both, and revisiting that decision as demand, knowledge, and system conditions change.

Neither location is uniformly preferable. Using external memory makes knowledge easy to inspect and update, but can repeatedly incur the cost of retrieval, context construction, model inference, or a remote service (Lewis et al., 2020). Parametric memory can avoid retrieval and additional context processing after knowledge has been successfully written into the model, but writing is costly and imperfect (Wang et al., 2024). Both paths are fallible: retrieval can miss relevant evidence, while parameter updates can fail silently or disturb other behavior. Consequently, the better choice depends on how the knowledge will be used, how long it remains useful, and the reliability and cost of each memory mechanism.

Existing lines of work generally study one part of this decision at a time. External-memory methods improve what an agent stores and retrieves (Lewis et al., 2020; Asai et al., 2024), while model-editing methods improve how knowledge is written into model parameters (Meng et al., 2022b; Mitchell et al., 2021). Hybrid systems retrieve experience, consolidate recurring knowledge, or coordinate external and parametric updates (Xia et al., 2026; Ji et al., 2026). These approaches provide important components, but do not explicitly compare the lifecycle value and exposure risk of available executors when deciding which representations to create, retain, or revise. This leaves a distinct question: **when should knowledge be maintained externally, when is it worth internalizing, and when should an earlier decision be revised?**

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.

Concretely, we establish three findings.

Knowledge placement is relational rather than intrinsic. Holding knowledge fixed, varying reuse, update rate, and external-service cost changes the relative lifecycle value of the available memory mechanisms and can reverse the preferred placement (Fig. 1). These results show that the preferred representation cannot generally be determined from the knowledge alone; it also depends on the workload and available memory mechanisms.

Placement decisions can become stale as workloads evolve. Compared with an otherwise identical policy that freezes its initial placement, re-evaluation becomes more valuable as post-decision demand grows, whereas changes without sufficient future use fail to repay their cost (Fig. 2). Placement should therefore be revisited as conditions change, but modified only when future savings exceed the cost.

Re-evaluating placement requires more than average executor success. Repeated physical writes show that prior outcomes identify knowledge that remains difficult for the same executor (Fig. 3a). Moreover, even at a fixed success probability, failure detectability can change whether direct use, verification, or fallback is preferred (Fig. 3b). Placement decisions therefore require knowledge-specific transaction history and evidence about each operation, not one executor-wide success rate.

Motivated by these findings, we propose DYNAMIC VERIFIED PLACEMENT (DVP). It compares admissible external and parametric representations by their expected lifecycle value, revisits placement as conditions change, and verifies new or updated memory before use. This lets DVP avoid unprofitable writes and costly external access, adapt to changing workloads, and keep uncertain memory results from being used directly.

We evaluate these benefits on LightShop (Li et al., 2026) and ZsRE (Levy et al., 2017) using implemented external and parametric memory. On LightShop, an oracle choosing between them per task raises the best fixed-mechanism success from 55.5% to 64.8%, confirming their complementarity. Under fixed budgets, DVP raises correct tasks over always external from 136.55 to 160.33 on the primary workload and from 202.92 to 266.24 under an independent shift. It also exceeds an information-matched one-step policy by 10.95 and 13.97 tasks, with no observed internal serving error passing the exact benchmark checks. The advantage over one-step placement persists when the exact benchmark checks are replaced by an independently selected and certified trace verifier. In a separate WISE evaluation, DVP starts with executor-level priors for previously unseen knowledge and updates its cards after online transactions. Every attempted transaction is included in the reported cost. On ZsRE, external-cost sensitivity shows that DVP moves from external service to selective internalization as the cost of external access increases.

2 RELATED WORK

External memory and adaptive retrieval. Retrieval-augmented generation grounds predictions in editable, inspectable stores (Lewis et al., 2020). Agent memory retains experience for later decisions (Shinn et al., 2023; Zhao et al., 2024; Packer et al., 2023), while adaptive retrieval learns when and how much to retrieve (Asai et al., 2024; Jeong et al., 2024; Tang et al., 2025). These methods manage what external memory stores or supplies at inference. DVP instead decides which representations to maintain across available memory mechanisms.

Parametric and native memory mechanisms. Model editing writes targeted behavior into existing parameters through learned gradient transformations or direct weight updates (Mitchell et al., 2021; Meng et al., 2022a;b); WISE isolates edits in a side memory for lifelong editing (Wang et al., 2024). Native-memory systems learn persistent states or generated guidance (Zhang et al., 2026a; Wang et al., 2026; Zhang et al., 2026b). These methods provide internal memory executors. DVP contributes the placement rule: it decides when an executor's capability, reliability, and lifecycle cost justify maintaining its representation, without designing the executor itself.

Hybrid memory management and dynamic placement. MemOS organizes and migrates plaintext, activation, and parametric memory (Li et al., 2025). UniMem retrieves sparse experiences and consolidates recurring patterns (Xia et al., 2026), while COVE coordinates harness- and parameter-based evolution through task- and stage-aware policies (Ji et al., 2026). DVP treats this coordination

as dynamic placement: it filters executors by capability and risk, compares their lifecycle value, revises earlier choices as conditions change, and verifies new representations before activation.

3 ANALYSIS

We ask what determines placement, when that decision expires, and what executor evidence is needed to revise it safely. The analyses combine measured executor outcomes and wall-clock costs with matched lifecycle replays.

3.1 PROBLEM SETUP

Knowledge units, representations, and executors. A *knowledge unit* k_j denotes information that may be reused and has a current value and a serving contract Γ_j. For a query and its observable outcome, the contract specifies whether that serving event is acceptable. Its declared scope may cover a single query type, a specified query set, or additional behavior-preservation requirements when corresponding evidence is available. An *executor* e, such as parameter writing, exact lookup, or retrieve-then-generate, makes the knowledge available through a representation $r_{j,e,t}$. The same knowledge may have an external representation, a parametric representation, or both, with different capabilities, failure modes, and costs. Placement determines which representations are maintained for future use; routing determines which available representation serves a particular query. We study independently maintained external records, adapters, and parameter side memories.

Fallible memory operations. An operation applies an executor to a knowledge unit but reveals only a receipt y_t, not its latent outcome o_t. Because a receipt may not reveal whether the resulting representation is usable, an acquisition can incur cost without being activated. Its value therefore depends separately on activation probability and serving behavior after activation.

Dynamic placement objective. The state s_t contains maintained representations, transaction history, workload forecasts, and resource constraints. An *executor card* summarizes capability, reliability, and lifecycle cost.

Let $z_{j,e,t} = 1$ indicate that an internal outcome produced by executor e for knowledge unit j satisfies its declared serving contract, and let $v_{j,e,t} = 1$ indicate that the read verifier releases that outcome. We define internal-output exposure risk as

$$\rho_{j,e,t} = \Pr(v_{j,e,t} = 1,\ z_{j,e,t} = 0 \mid s_t, j, e)\,. \tag{1}$$

Thus, $\rho_{j,e,t}$ measures the probability that an incorrect internal outcome passes verification and becomes user-visible. Recovery success and cost enter lifecycle value separately. The controller first removes actions whose executors cannot satisfy the required capability or whose exposure risk exceeds the limit ϵ_j. The certificate covers only outcomes and side effects checked by the serving contract. A pair-specific certificate can bound the conditional quantity above. A pooled empirical certificate instead bounds marginal exposure over its declared calibration mixture; it is not interpreted as a separate bound for every knowledge unit or state.

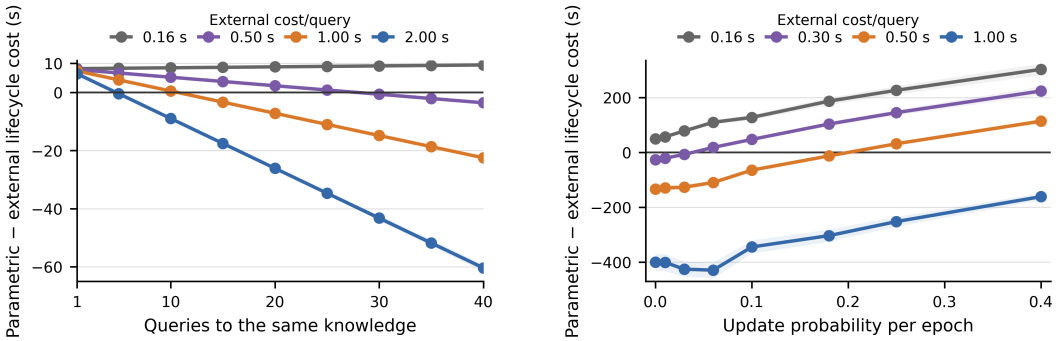
Among the remaining actions, the placement policy minimizes expected lifecycle cost and task loss:

$$\min_{\pi}\ \mathbb{E}_{\pi}\left[\sum_{t=1}^{T}\left(C_t + \lambda L_t^{\text{task}}\right)\right] \quad \text{s.t.} \quad \rho_{j,e,t} \le \epsilon_j. \tag{2}$$

Here, C_t includes acquisition, serving, verification, maintenance, and recovery costs; L_t^{task} measures task failure; and λ converts task failure into the common objective. A version change invalidates the previous representation. The policy chooses which admissible representations to create, retain, refresh, or remove.

3.2 PLACEMENT IS RELATIONAL

By *relational*, we mean that a knowledge unit has no intrinsically best representation. We hold knowledge and measured executor outcomes fixed while varying future queries, updates, and external-access cost. The comparisons isolate the wall-clock term C_t under matched behavior; the end-to-end experiments evaluate correct task completion under the full objective.



(a) Reuse and access cost.

(b) Update rate and access cost.

Figure 1: **Placement changes with workload and external-access cost.** Lifecycle cost is parametric minus external; negative values favor parametric memory.

Reuse and access cost jointly reverse placement. Figure 1a compares the lifecycle cost of parametric and external representations. Under the measured local context cost, external service remains cheaper through 128 queries per item. When an external query costs 0.5 seconds, parametric memory becomes cheaper after about 29 queries; at 2.0 seconds, the crossover occurs after about five. Acquisition is worthwhile only when the savings from future use exceed its cost.

Update rate can also reverse placement. We next hold query demand and executor outcomes fixed while varying the update rate. At an external-access cost of 0.3 seconds, the verified parametric representation has lower lifecycle cost through an update probability of 0.03, whereas the external representation is cheaper from 0.06 onward (Figure 1b). At 0.5 seconds, the reversal occurs between update probabilities 0.18 and 0.25. Updates shorten the useful lifetime of a representation, and the update rate at which placement reverses depends on the cost of the alternative executor.

These interventions reverse the preferred representation without assigning a different intrinsic type to the knowledge. Placement is therefore determined jointly by the knowledge lifecycle and the available executors. Because both can change, an earlier decision need not remain optimal.

3.3 Placement Decisions Can Expire as Demand Changes

We next test whether revising placement after a change in demand reduces lifecycle cost. We compare two policies that observe the same event stream and make identical decisions through a common split point. Thereafter, one may revise placement using the observed demand, while the other retains its earlier placement. Both use the same cost model, executor outcomes, and safety operations, so the comparison isolates the value of revision.

Rising demand can invalidate an earlier placement. As post-split query demand rises, revising placement reduces lifecycle cost relative to retaining the earlier decision (Figure 2). At the largest tested increase, the revised policy saves 1,295.67 seconds over the frozen policy. A representation that was initially not worth acquiring can therefore become worthwhile when its expected reuse increases.

Elapsed time alone does not justify changing placement. When underlying demand is unchanged, finite-history fluctuations can still change the estimated future value and trigger transactions that do not repay their cost. These revisions add 61.88 seconds relative to retaining the earlier placement. A policy should re-evaluate placement as demand evidence changes, but execute a change only when its expected future saving repays the transaction cost. Doing so requires predicting the consequences of fallible executor actions from the evidence available at decision time.

3.4 Re-evaluating Placement Requires More Than Average Success

The preceding results show why an earlier placement must sometimes be revised. Making that revision requires the controller to predict the consequences of each available executor. We therefore

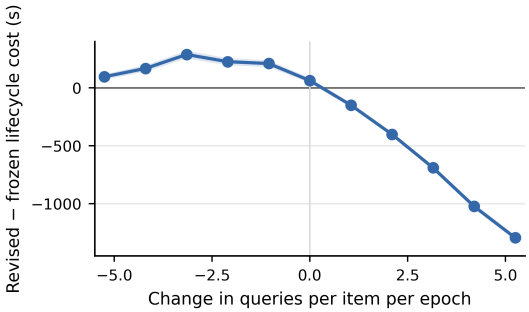
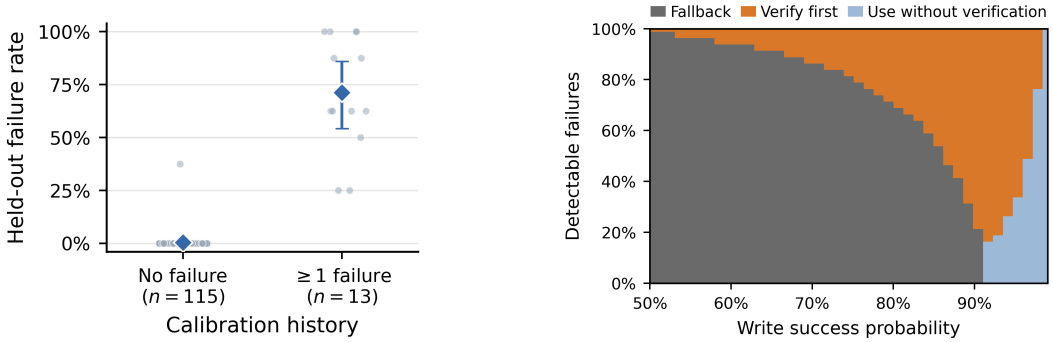


Figure 2: **Demand change determines when a placement decision expires.** Values are revised-minus-frozen lifecycle cost; negative favors revision.



(a) History predicts held-out failures.

(b) Detectability changes the action.

Figure 3: **Average success omits decision-relevant reliability.** (a) Prior outcomes predict held-out write failures: gray points represent individual knowledge units, and diamonds show group means with 95% confidence intervals. (b) The preferred action depends jointly on write success and failure detectability.

ask whether its average success rate is sufficient. The following results show that this average omits how reliability varies across knowledge and whether failures can be observed.

Past outcomes identify knowledge that remains difficult to write. We perform 16 physical writes with the WISE parameter-writing executor (Wang et al., 2024) for each of 128 knowledge units. The first eight writes summarize prior outcomes, and the next eight form a held-out evaluation. Units with no calibration failure have a held-out failure rate of 0.3%, whereas units with at least one calibration failure have a rate of 71.2% (Figure 3a). The executor-wide average therefore conceals large, persistent differences across executor–knowledge pairs. Transaction history identifies which knowledge remains risky to write.

Failure detectability changes how an executor should be used. Success probability alone still does not determine the best action. In a controlled one-attempt decision analysis, we vary write success and the fraction of unsuccessful writes that can be detected, while holding the declared costs fixed. Using the result without verification, verifying first, and falling back are preferred in different regions (Figure 3b). At the same success probability, changing only detectability can change the preferred action. Reliability must therefore describe both the likelihood of success and the evidence available before an uncertain result is exposed. Appendix A.3.8 gives the costs and losses used to construct the decision map.

Together, these findings impose three requirements. First, placement must compare capability and complete lifecycle value across available executors. Second, it must be re-evaluated from observed demand and version history while charging the cost of each change. Third, reliability estimates must use the relevant transaction history, and activation and recovery decisions must depend on the

evidence available for each operation. These requirements motivate the dynamic verified placement method introduced next.

4 Dynamic Verified Placement

Following the three requirements identified in our analysis, DVP screens executors for capability and risk, compares admissible placements by expected lifecycle value, and verifies a candidate before activation (Figure 4).

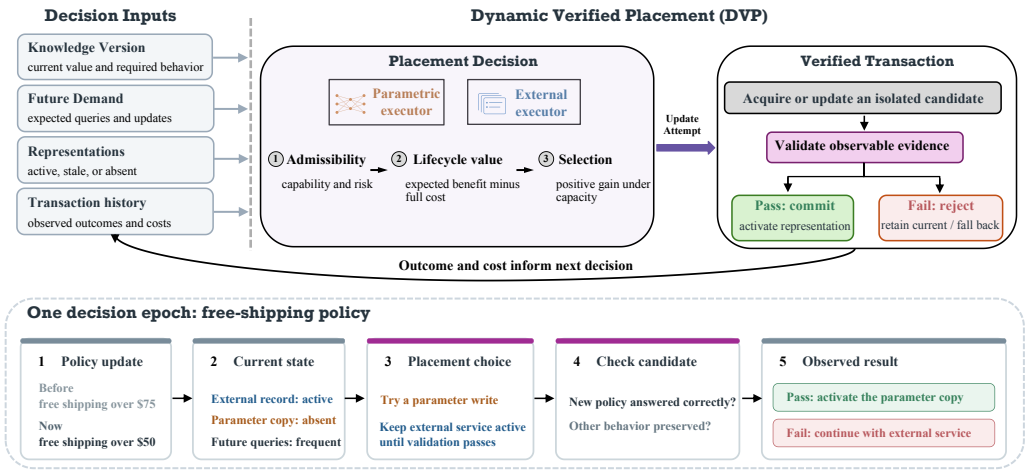


Figure 4: **Overview of DVP.** Executor candidates are screened by capability and risk, compared by lifecycle value, and verified before activation.

Executor cards. For knowledge unit j and executor e, DVP maintains

$$\mathcal{C}_{j,e,t} = \big(\Gamma_{j,e}, \hat{a}_{j,e,t}, \hat{p}_{j,e,t}, \bar{\rho}_{j,e,t}, c^{\text{txn}}_{j,e}, c^{\text{serve}}_{j,e}, c^{\text{keep}}_{j,e,t}, b_{j,e}\big). \tag{3}$$

The card records capability Γ, activation probability $\hat{a}$, the success probability of the full serving path $\hat{p}$, exposure-risk bound $\bar{\rho}$, transaction, serving and maintenance costs, and representation footprint b. Each risk bound also records its certification scope: a knowledge-specific conditional scope for pair-level evidence, or a declared population mixture for pooled evidence. The latter is used only as a marginal bound for that mixture and does not imply a conditional bound for every unit. Let $A_{j,e,t} = 1$ denote that an acquisition passes its transaction check and is committed. For a subsequent query, let $Z = 1$ denote a correct initial executor outcome, $V = 1$ verifier acceptance, and $F = 1$ successful fallback after rejection. Then $\hat{a}_{j,e,t} = \Pr(A_{j,e,t} = 1 \mid H_t, j, e)$ and

$$\hat{p}_{j,e,t} = \Pr(V = 1, Z = 1 \mid A = 1, H_t, j, e) + \Pr(V = 0, F = 1 \mid A = 1, H_t, j, e). \tag{4}$$

Thus, $\hat{p}$ measures the complete serving path, including recovery, while $\bar{\rho}_{j,e,t}$ bounds $\Pr(V = 1, Z = 0 \mid A = 1, H_t, j, e)$, the probability that an incorrect outcome from the internal executor is released. Recovery success contributes to $\hat{p}$ and task loss rather than this internal-output exposure bound. Transaction cost includes acquisition, verification, and expected recovery after rejection; serving cost includes read verification and expected fallback. Each deployment protocol declares which card components are frozen from disjoint calibration data and which are updated from transaction history; cold-start rules are given in Appendix A.1.

Admissibility precedes optimization. Let Γ^{req}_j be the required capability, ϵ_j the risk limit, and e_0 the declared external recovery executor. The admissible internal candidates are

$$\mathcal{A}_{j,t} = \{e \neq e_0 : \Gamma_{j,e} \succeq \Gamma^{\text{req}}_j,\ \bar{\rho}_{j,e,t} \leq \epsilon_j\}. \tag{5}$$

The currently maintained representation remains a candidate while valid; if no internal candidate has positive value, the unit is served by e_0. A certificate is applied only to the scope for which it

was calibrated; pooled certificates support population-level admission but not an unobserved pair-specific guarantee.

Lifecycle-aware placement. At each decision epoch, DVP estimates value over a finite H-epoch window and replans after new query, version, or transaction observations. Let the per-epoch update probability of knowledge unit j have prior $u_j \sim \mathrm{Beta}(\alpha, \beta)$. After observing $n_{j,t}^{\mathrm{upd}}$ updates in $n_{j,t}^{\mathrm{obs}}$ epochs, the posterior mean is

$$\hat{u}_{j,t} = \frac{\alpha + n_{j,t}^{\mathrm{upd}}}{\alpha + \beta + n_{j,t}^{\mathrm{obs}}}. \tag{6}$$

Let $\hat{q}_{j,t}$ be the recent query rate, $\hat{L}_{j,t}$ the expected remaining lifetime of the current version within the planning window, and $\hat{N}_{j,t}$ its predicted future queries. We compute

$$\hat{L}_{j,t} = \frac{1 - (1 - \hat{u}_{j,t})^H}{\hat{u}_{j,t}}, \qquad \hat{N}_{j,t} = \hat{q}_{j,t}\hat{L}_{j,t}, \tag{7}$$

with $\hat{L}_{j,t} = H$ when $\hat{u}_{j,t} = 0$.

For representation r, define its conditional active value relative to external service as

$$V_{j,r,t} = \hat{N}_{j,t}\left[c_{j,e_0}^{\mathrm{serve}} - c_{j,r}^{\mathrm{serve}} + \lambda\big(\hat{p}_{j,r,t} - \hat{p}_{j,e_0,t}\big)\right] - \Delta c_{j,r\leftarrow e_0,t}^{\mathrm{keep}}, \qquad V_{j,e_0,t} = 0. \tag{8}$$

If candidate e would replace the current representation r, its activation-weighted transition gain is

$$G_{j,e\leftarrow r,t} = \hat{a}_{j,e,t}\big(V_{j,e,t} - V_{j,r,t}\big) - \delta_{j,e,t}c_{j,e}^{\mathrm{txn}}. \tag{9}$$

Successful activation replaces r; rejection retains it, while the attempted transaction still incurs $c_{j,e}^{\mathrm{txn}}$. External service is the special case $r = e_0$, which recovers the standalone acquisition score. The indicator $\delta_{j,e,t}$ equals one when the representation must be acquired or refreshed. For an already maintained valid representation, $\hat{a}_{j,e,t} = 1$ and $\delta_{j,e,t} = 0$. The maintenance term includes any retained backup, and λ converts expected task success into the common objective.

Let $\mathcal{T}_t$ contain feasible transitions $\tau = (j, e \leftarrow r)$, where r identifies either an occupied capacity slot or a distinct empty slot, and let $x_{\tau,t} \in \{0, 1\}$. DVP selects positive-gain transitions by

$$\begin{aligned} \max_{\mathbf{x}_t} \quad & \sum_{\tau\in\mathcal{T}_t} x_{\tau,t}G_{\tau,t} \\ \text{s.t.} \quad & \sum_{j} \sum_{r\in M_{j,t+1}(\mathbf{x}_t)} b_{j,r} \le B_t, \\ & \sum_{\tau:\, j(\tau)=j} x_{\tau,t} \le 1, \qquad \sum_{\tau:\, r(\tau)=r} x_{\tau,t} \le 1. \end{aligned} \tag{10}$$

Here, $M_{j,t+1}(\mathbf{x}_t)$ is the maintained set after the selected transactions. It contains the representations retained or successfully committed under the selected placement, including any backup whose footprint and maintenance cost are charged by the objective. The final constraint prevents two candidates from claiming the same incumbent or empty slot. Retaining a valid resident defines the zero-gain baseline. Its original write cost is sunk and is excluded from the comparison. Under capacity pressure, a candidate is compared with the resident occupying its assigned slot; the resident's value is lost only after successful activation. This per-epoch optimization is a receding-horizon surrogate for the sequential objective in Eq. 2. The evaluated capacity structures admit exact discrete solutions, detailed in Appendix A.1.

Verification before activation and serving. An acquisition is first created as an isolated candidate. Its transaction receipt is checked before the candidate can replace the active representation. Acceptance commits the candidate atomically; rejection retains the current representation. Because successful activation does not certify every future use, the serving path applies the declared read verifier to each evaluated outcome. An accepted outcome is released, whereas a rejected outcome invokes an admissible recovery executor. We charge the cost of every attempted acquisition, check, and recovery operation. Verification covers only the observable properties in the capability contract.

Algorithm 1 gives the complete decision procedure.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Benchmarks. LightShop (Li et al., 2026) provides tool-use tasks with versioned rules, changing demand, and checked outcomes. We test transfer on held-out ZsRE knowledge (Levy et al., 2017).

Executors. Both LightShop executors use the same Qwen3-8B agent (Yang et al., 2025) and tools: external execution supplies the current rule, while internal execution loads a reversible rank-8 LoRA (Hu et al., 2021). ZsRE pairs BGE-M3 retrieval (Chen et al., 2024) with verified WISE writes (Wang et al., 2024).

Baselines. We compare with fixed, static, one-step, and adaptive-amortization placement, plus UniMem-style recurrence (Xia et al., 2026) and COVE-style coordination (Ji et al., 2026). The matched controls and tuning are defined in Appendix A.1.

Metrics and protocol. Our main metric is correct tasks completed under a common wall-clock budget. All queries and memory operations consume measured time. Executor cards use disjoint development tasks; query and update forecasts use online history. Appendix A.3 evaluates cold start and online cards.

5.2 MAIN RESULTS

External and internal memory provide complementary capabilities. External and internal execution solve 49.2% and 55.5% of held-out tasks, respectively. External execution alone solves 12 tasks, while internal execution alone solves 20; together, the two executors cover 64.8% of tasks.

Lifecycle conditions determine how this complementarity should be used. Table 1 favors external execution when writes cannot be amortized, internal execution under stable reuse, and selective placement under heterogeneous lifetimes and updates.

Table 1: **Mean correct tasks per stream across lifecycle regimes.**

Policy	Short horizon	Long horizon low update rate	Long horizon high update rate	Mixed stream
Always external	**18.71**	180.64	95.53	153.03
Always internal	10.93	**361.26**	128.02	31.50
DVP	**18.71**	350.53	**156.65**	**187.93**

DVP completes more tasks under the same budget. Using online history, DVP completes 160.33 and 266.24 correct tasks on the primary and shifted workloads (Table 2). It exceeds adaptive amortization by 3.80 and 9.38 tasks, and the one-step policy by 10.95 and 13.97 tasks. All four paired intervals exclude zero (Appendix A.3.1) and the gain persists across budgets (Figure 5).

Table 2: **Mean correct tasks per stream under a common lifecycle budget.**

Policy	Primary @ 3,500 s	Shifted @ 5,500 s
Always external	136.55	202.92
Static placement	139.30	216.07
One-step placement	149.38	252.27
Adaptive amortization	156.52	256.87
UniMem-style recurrence	125.11	218.24
COVE-style coordination	152.70	208.83
DVP	**160.33**	**266.24**
Internal, zero write cost	145.39	233.52
Hindsight oracle	171.36	286.22

The placement rule transfers to natural-language knowledge. On held-out ZsRE, DVP moves from external-only to mixed placement as external-access cost rises, outperforming static and external-only placement (Appendix A.3).

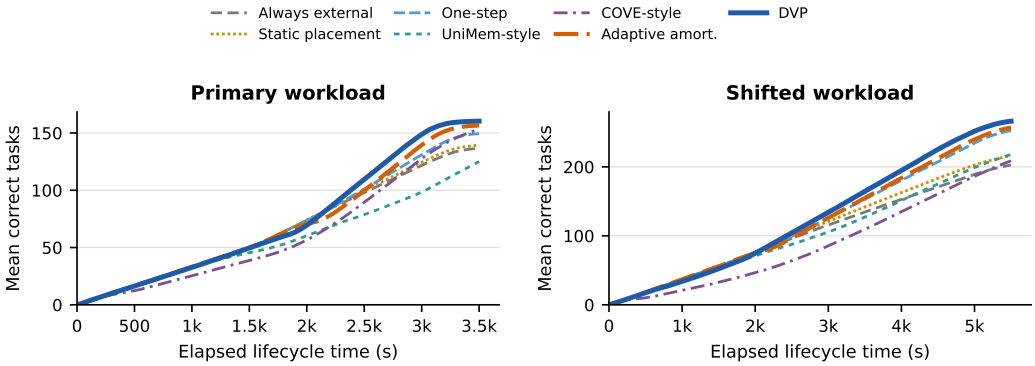


Figure 5: **Correct task completion as the lifecycle budget increases.**

5.3 Why Does DVP Work?

Each component contributes. Re-evaluation, multi-epoch planning, and knowledge-specific reliability each increase correct tasks relative to their respective ablations (Table 3).

Table 3: **DVP ablations.** Change in correct tasks from full DVP and mean writes per stream.

	Primary		Shifted	
Variant	Δ correct	Mean writes	Δ correct	Mean writes
No re-evaluation	−21.04	0.06	−44.32	0.76
One-step placement	−10.95	0.23	−13.97	1.05
Global-average reliability	−93.53	5.17	−91.18	4.91

Verification reduces exposed memory failures. Without read verification, the shifted workload exposes an average of 4.33 internal errors per stream. Exact verification eliminates these exposed errors, at the cost of 1.81 fewer correctly completed tasks per stream. With the independently certified verifier, DVP still completes 7.88 and 11.30 more correct tasks per stream than the one-step policy on the primary and shifted workloads, respectively (Appendix A.3.9).

5.4 Runtime Overhead and End-to-End Cost

Table 4: **DVP runtime overhead.** Mean seconds per stream; overhead includes controller computation and verification.

Workload	Controller	Verification	Executor	End-to-end	Overhead / budget (%)
Primary	0.0008	0.442	3,122.99	3,123.44	0.0126
Shifted	0.0013	0.769	5,184.08	5,184.85	0.0140

Controller computation takes less than 0.04 ms per decision. Controller computation and verification together consume at most 0.014% of the evaluation budget. On the primary workload, DVP spends an average of 237 seconds on writes, 728 seconds on internal service, and 2,158 seconds on external service per stream. Always external spends 3,340 seconds on external service.

6 Conclusion

DVP uses workload forecasts and knowledge-specific reliability estimates to choose between external and parametric memory, revisiting those choices as conditions change. In the LightShop evaluations, it completes more correct tasks than the compared placement policies under the same budgets. On ZsRE, it shifts toward selective internalization as external-access cost increases.

AI Use Statement

We used generative AI tools to draft and revise portions of the manuscript, improve its clarity and organization, and support the retrieval and discovery of related literature. We did not use genera-

tive AI to generate synthetic datasets, develop the conceptual or mathematical framework, formulate or prove mathematical claims, propose hypotheses, design or execute experiments, implement the methods, process datasets, or interpret experimental results. Tasks involving translation, interviews, surveys, recordings, and qualitative or thematic analysis were not applicable to this work. All AI-assisted text was reviewed and substantially revised by the authors; suggested references were verified against their original publications, and factual and numerical statements were checked against the underlying experimental results. We take full responsibility for the final content of this work, including all text, claims, citations, and artifacts produced with the aid of generative AI.

REFERENCES

Akari Asai, Zeqiu Wu, Yizhong Wang, Avi Sil, and Hannaneh Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *International conference on learning representations*, volume 2024, pp. 9112–9141, 2024.

Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In *Findings of the association for computational linguistics: ACL 2024*, pp. 2318–2335, 2024.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C Park. Adaptive-rag: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of the 2024 conference of the north american chapter of the association for computational linguistics: Human language technologies (volume 1: Long papers)*, pp. 7036–7050, 2024.

Tianyun Ji, Zhenya Huang, Jiayu Liu, Zirui Liu, Yu Su, and Hongbin Pei. Learning what to remember and what to internalize in llm self-evolution via adaptive memory-parameter coordination. 2026.

Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. Zero-shot relation extraction via reading comprehension. In *Proceedings of the 21st conference on computational natural language learning (CoNLL 2017)*, pp. 333–342, 2017.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33: 9459–9474, 2020.

Yuanyang Li, Xue Yang, Longyue Wang, Weihua Luo, and Hongyang Chen. ComplexMCP: Evaluation of LLM agents in dynamic, interdependent, and large-scale tool sandbox. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=3u1M9wokD7.

Zhiyu Li, Shichao Song, Hanyu Wang, Simin Niu, Ding Chen, Jiawei Yang, Chenyang Xi, Huayi Lai, Jihao Zhao, Yezhaohui Wang, et al. Memos: An operating system for memory-augmented generation (mag) in large language models. *arXiv preprint arXiv:2505.22101*, 2025.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. *Advances in neural information processing systems*, 35:17359–17372, 2022a.

Kevin Meng, Arnab Sen Sharma, Alex Andonian, Yonatan Belinkov, and David Bau. Mass-editing memory in a transformer. *arXiv preprint arXiv:2210.07229*, 2022b.

Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D Manning. Fast model editing at scale. *arXiv preprint arXiv:2110.11309*, 2021.

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.

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. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.

Xiaqiang Tang, Qiang Gao, Jian Li, Nan Du, Qi Li, and Sihong Xie. Mba-rag: a bandit approach for adaptive retrieval-augmented generation through question complexity. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 3248–3254, 2025.

Peng Wang, Zexi Li, Ningyu Zhang, Ziwen Xu, Yunzhi Yao, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Wise: Rethinking the knowledge memory for lifelong model editing of large language models. *Advances in Neural Information Processing Systems*, 37:53764–53797, 2024.

Xiaoqiang Wang, Chao Wang, Hadi Nekoei, Christopher Pal, Alexandre Lacoste, Spandana Gella, Bang Liu, and Perouz Taslakian. Mem-π: Adaptive memory through learning when and what to generate, 2026. URL https://arxiv.org/abs/2605.21463.

Siyu Xia, Chenheng Zhang, Yanting Wu, Haoxuan Li, Jiajun Chai, Xiaohan Wang, Guojun Yin, Wei Lin, Zhouchen Lin, Haifeng Zhang, et al. Unimem: Complementary episodic-to-parametric memory for boundary-agnostic task streams. *arXiv preprint arXiv:2607.26017*, 2026.

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

Guibin Zhang, Muxin Fu, and Shuicheng Yan. Memgen: Weaving generative latent memory for self-evolving agents. In *International Conference on Learning Representations*, volume 2026, pp. 22555–22588, 2026a.

Zeyu Zhang, Ziliang Guo, Yihang Sun, Xichong Zhang, Xixuan Hao, Zehao Lin, Yang Zhang, Xiaoyan Zhao, Tong Shen, Bo Tang, et al. Metis: Memory foundation model. *arXiv preprint arXiv:2607.26760*, 2026b.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

A APPENDIX

A.1 EXPERIMENTAL DETAILS

Evaluation inputs. The evaluation supplies DVP with reusable knowledge units and version-change events. The controller uses these inputs to choose which representations to maintain and when to refresh them.

Implementation and evaluation protocol. Table 5 summarizes the settings used for the reported results. LightShop executor cards are estimated from 128 physical development tasks (16 states for each of four rules and two versions). The primary and shifted evaluations use two disjoint sets of 128 physical tasks. Controller hyperparameters are selected on 128 independently generated development streams before opening the 256 primary and 512 shifted evaluation streams. ZsRE uses 64 items for executor-card calibration and 64 disjoint items for evaluation. All policies within a workload replay the same event tapes and physical executor outcomes. A realized activation outcome is revealed only after a policy selects and pays for the corresponding transaction; before that decision, the policy observes only its executor-card estimate.

Table 5: **Experimental settings.** Costs are measured wall-clock seconds; failed and rejected operations retain their cost.

Component	LightShop	ZsRE
Model	Qwen3-8B; deterministic decoding	Qwen3-8B with WISE; BGE-M3 retrieval
Accelerator	Eight PPU-ZW810 devices (95.6 GiB each); one rule–version block per device; FP32 eager attention	BF16 WISE on PPU-ZW810; FP32 BGE-M3 on CPU, with a frozen GPU retrieval-time card
Evaluation units	128 tasks per split: 4 rules $\times$ 2 versions $\times$ 16 states	64 calibration and 64 held-out items
Streams	128 development; 256 primary; 512 shifted	256 streams, each with 40 epochs and 32 queries per epoch
Budget / capacity	3,500 s / 1 (primary); 5,500 s / 1 (shifted)	900 s / 8
Parameter write	Rank-8 LoRA, $\alpha = 16$, AdamW, 80 steps, learning rate 10^{-3}, batch size 1, at most 8,192 training tokens	Physical Qwen3-8B WISE write
Cost measurement	Separate end-to-end wall-clock time for LoRA writing, internal serving, and runbook-conditioned serving	Separate WISE write/read and BGE-M3 retrieval time; the 4K mode adds a measured Qwen3-8B reader cost

LightShop stream generation and stopping. Table 6 specifies the frozen event process. At each epoch, each rule independently changes version with its declared update probability; queries are then sampled from the hot-rule distribution. The hot rule changes once at the stated epoch. Each policy receives the same resulting event tape and paired physical executor outcomes.

Table 6: **Frozen LightShop stream protocol.** Query probabilities list the hot rule followed by each of the other three rules. The quantity-limit rule uses the stable update rate; the other rules use the volatile rate.

Setting	Primary	Shifted
Evaluation streams	256	512
Epochs / queries per epoch	32 / 8	40 / 10
Hot-rule change epoch	16	12
Hot / other query probability	0.70 / 0.10	0.55 / 0.15
Stable / volatile update probability	0.01 / 0.35	0.05 / 0.20
Warm-up / history window (epochs)	4 / 4	4 / 4
Budget / representation capacity	3,500 s / 1	5,500 s / 1

Events are processed chronologically until the next service cannot fit within the remaining budget. An internal query is attempted only when the remaining budget can cover its read, verification, and possible external recovery; otherwise the policy uses the external path if that service fits. Every write, read, verification, rejection, and recovery retains its realized cost.

For LightShop, a write creates an isolated rule–version adapter. Each internal read is checked by the deterministic benchmark-state judge, whose measured mean runtime is 5.237 ms over 256 outcomes and five timing repetitions. A failed check is not served: the same query is executed with the

external runbook, and the write, internal read, verification, and fallback are all charged. Version changes invalidate the corresponding adapter. On ZsRE, the write receipt and an independent target-signature read check determine whether a WISE result is used; rejected reads use BGE-M3 retrieval. The retrieval-only card charges 0.0256 s per query. The 4K-reader card adds 1.478 s as an external-cost sensitivity, giving 1.5036 s per query. Calibration gives 10.051 s per WISE write and 1.059 s per internal read.

Lifecycle-regime settings. Table 7 gives the four settings used in Table 1. Each setting uses 256 streams, the same measured executor cards, capacity one, a four-epoch history and warm-up, and a task-success value of 100 seconds.

Table 7: **Lifecycle-regime protocol.** The mixed stream changes its hot rule at epoch 10; its first three rules update with probability 0.35 and the quantity-limit rule with probability 0.01.

Regime	Budget (s)	Epochs	Queries/epoch	Planning epochs	Demand and update process
Short	300	128	4	2	one rule; update 0.01
Long, low update	3,500	256	4	32	one rule; update 0.01
Long, high update	3,500	256	4	32	one rule; update 0.35
Mixed	3,500	32	10	8	four rules; hot rule changes

Information available to each policy. Table 8 states the decision information used by the main LightShop policies. All policies share evaluation streams, physical executor outcomes, and budgets. The one-step ablation matches DVP's online information and transaction protocol and changes only its planning horizon. Adaptive amortization additionally matches the planning horizon and capacity, but removes task-success value from its placement score.

Table 8: **Decision signals and revision rule used by each policy.**

Policy	Query history	Version events	Reliability	Lifecycle costs	Read check	Revision rule
Always external	No	Yes	–	No	–	none
Static placement	Yes	Yes	card	Yes	No	frozen
One-step placement	Yes	Yes	pair-specific	Yes	Yes	one interval
Adaptive amortization	Yes	Yes	pair-specific	Yes	Yes	break-even
UniMem-style	Yes	Yes	–	No	No	recurrence
COVE-style	Yes	Yes	task feedback	No	Yes	stage triggers
DVP	Yes	Yes	pair-specific	Yes	Yes	lifecycle value

Policy definitions and tuning. *Always external* executes every query with the external executor. *Always internal* writes the current version when it is first queried, invalidates it after an update, and thereafter serves that version internally. *Static placement* chooses one rule after warm-up using the initial history and frozen executor cards, keeps that membership fixed, and refreshes the selected rule when its version changes. *Hindsight* selects the successful executor for each realized query using both counterfactual outcomes and provides an upper-bound reference.

Adaptive amortization uses the same online query and update forecasts, pair-specific cards, verification and fallback protocol, 16-epoch planning horizon, and capacity constraint as DVP. It sets task-success value to zero and admits an internal representation only when its expected serving-time saving over the estimated valid lifetime exceeds the measured write cost. Thus, its sole difference from DVP is whether executor effects on task success enter lifecycle value.

The UniMem-style comparison implements recurrence-led consolidation on the common executor interface. A rule–version is served externally until its observed recurrence count reaches 50, then written internally if the capacity-one store is free. Updates invalidate stale versions; admission depends only on recurrence, and membership persists between version changes. The evaluation uses a threshold of 50, frozen before evaluation; sensitivity results additionally report thresholds of 25 and 75. The COVE-style comparison begins with every rule eligible for both paths and uses only outcomes from the path actually executed. It triggers an internal write after enough successful trajectories, cold-start failure, or a performance plateau, and returns a rule to external-only service after enough observed revisions. Its thresholds are selected by correct tasks under the fixed budget on the same 128 development streams used for DVP: 8 successful trajectories, an 8-observation plateau window, a 0.25 cold-start success threshold, and 2 observed revisions. DVP selects a $\mathrm{Beta}(1, 9)$

prior and a 16-epoch planning cap on these development streams. At evaluation time, every online policy conditions solely on observations from its executed path.

Card estimation and cold start. In LightShop, the complete-serving card is estimated from paired internal and external outcomes on disjoint physical development tasks. If Z denotes a correct internal outcome, V verifier acceptance, and F successful external recovery, then

$$\hat{p}_{j,e,t} = \Pr(V = 1, Z = 1) + \Pr(V = 0, F = 1). \tag{11}$$

The serving-cost card likewise charges the internal read, verification, and expected external recovery after rejection. The joint recovery term is estimated directly from paired outcomes rather than by assuming independent executor failures. In particular,

$$c^{\mathrm{serve}}_{j,e} = \mathbb{E}[C^{\mathrm{read}} + C^{\mathrm{verify}} + \mathbf{1}\{V = 0\}C^{\mathrm{fallback}} \mid A = 1, H_t, j, e]. \tag{12}$$

In the main LightShop evaluation, $\hat{a}$, $\hat{p}$, and $\bar{\rho}$ are frozen before evaluation; only query and version-update forecasts are updated online. The internal representations are prevalidated reversible adapters, so loading an available adapter has $\hat{a}_{j,e,t} = 1$; its serving uncertainty remains in $\hat{p}_{j,e,t}$ and $\bar{\rho}_{j,e,t}$. All evaluated rule–version pairs have development cards, so the main LightShop comparison does not extrapolate to an unseen pair. In ZsRE, the 64 calibration items define an aggregate WISE card for the 64 held-out items: $\hat{a}$ is the fraction of writes that pass both the write receipt and independent read check, conditional serving success is one under the exact target-signature contract, and failed activation retains external service. More generally, a pair with no pair-specific history inherits the capability-matched executor-level card. If that card cannot certify the required risk limit, the pair remains inadmissible until calibration evidence is available.

Forecasts and placement solver. For each LightShop rule, $\hat{q}_{j,t}$ is its query count over the preceding four epochs divided by the number of observed epochs in that window. The update estimator uses every observed epoch and the selected $\mathrm{Beta}(1, 9)$ prior, $\hat{u}_{j,t} = (1 + n^{\mathrm{upd}}_{j,t})/(10 + n^{\mathrm{obs}}_{j,t})$. The posterior mean is inserted into the truncated-lifetime expression in the main text to obtain $\hat{N}_{j,t}$. Because the LightShop capacity is one, the implemented solver compares the external path, the valid resident without a new write cost, and every admissible candidate with its acquisition cost. It chooses the largest positive transition gain, or retains the current path if all gains are nonpositive. ZsRE uses eight explicit equal-size capacity slots. Empty slots have incumbent value zero; occupied slots carry the value of their current resident. The aggregate ZsRE card gives all candidates the same activation probability and transaction cost. The solver orders candidate active values from high to low and incumbent-slot values from low to high, pairs them one-to-one, and executes only positive-gain pairs. By the rearrangement inequality, this is the maximum-weight assignment for this common-card case. A failed transaction leaves its paired incumbent in place, and a successful transaction replaces only that incumbent.

Implementation tests cover the objective's limiting cases: zero capacity recovers external-only service; a valid resident is evaluated without a new write cost; a stale resident requires reacquisition; and zero forecast demand cannot justify a positive-cost write. A separate exhaustive audit enumerates all candidate subsets and slot permutations in small instances and matches the implemented ZsRE assignment exactly.

A.2 Complete DVP Decision Procedure

Algorithm 1 summarizes one decision epoch. It first updates the decision state from observations, then selects among admissible representations, and finally applies each selected change as a verified memory transaction.

Activation and transaction outcomes. For a new or refreshed representation, let $A_{j,e,t} \in \{0, 1\}$ indicate whether acquisition produces a candidate that passes transaction verification and is committed. Before executing the transaction, the controller predicts

$$\hat{a}_{j,e,t} = \Pr(A_{j,e,t} = 1 \mid H_t, j, e)\,, \tag{13}$$

where H_t contains only observations available before the decision. The realized value of $A_{j,e,t}$ is observed only after the acquisition attempt.

Conditional on $A_{j,e,t} = 1$, $\hat{p}_{j,e,t}$ predicts success of the complete serving path: either the initial result is correct and accepted, or it is rejected and recovery succeeds. The bound $\bar{\rho}_{j,e,t}$ limits the probability that an incorrect result passes the read verifier and is exposed. Thus, $\hat{a}$, $\hat{p}$, and $\bar{\rho}$ describe activation, end-to-end serving success, and exposure risk, respectively.

Let $C^{\text{txn}}_{j,e,t}$ denote the realized acquisition cost, including acquisition, transaction verification, and the outcome-specific commit or rejection path. The executor card stores its conditional expectation,

$$c^{\text{txn}}_{j,e} = \mathbb{E}\big[C^{\text{txn}}_{j,e,t} \mid H_t, j, e\big]. \tag{14}$$

This cost includes rejected attempts and is charged whenever $\delta_{j,e,t} = 1$, regardless of whether the candidate is committed.

Let r be the current incumbent and let $V^{\text{active}}_{j,e,t}$ and $V^{\text{active}}_{j,r,t}$ denote their future values conditional on being active. The incremental value conditional on successful replacement is

$$\Delta V^{\text{active}}_{j,e\leftarrow r,t} = V^{\text{active}}_{j,e,t} - V^{\text{active}}_{j,r,t}. \tag{15}$$

If acquisition is rejected, r remains active and the future incremental value is zero. Hence

$$\mathbb{E}\big[A_{j,e,t}\Delta V^{\text{active}}_{j,e\leftarrow r,t} - \delta_{j,e,t}C^{\text{txn}}_{j,e,t} \mid H_t, j, e\big] = \hat{a}_{j,e,t}\Delta V^{\text{active}}_{j,e\leftarrow r,t} - \delta_{j,e,t}c^{\text{txn}}_{j,e}, \tag{16}$$

which yields the transition gain in the main text. With external service as the incumbent, $V^{\text{active}}_{j,e_0,t} = 0$. Retaining a valid incumbent is the zero-gain baseline; a stale or removed representation requires a new transaction.

Algorithm 1 One DVP decision epoch

Require: Maintained representations M_t, transaction history H_t, executor cards $\{\mathcal{C}_{j,e,t}\}$, capability requirements $\{\Gamma^{\text{req}}_j\}$, risk limits $\{\epsilon_j\}$, capacity B_t, planning horizon H
Ensure: Updated representations M_{t+1}, history H_{t+1}, and cards

for each knowledge unit j **do**
  Invalidate representations whose version is stale
  Estimate query rate $\hat{q}_{j,t}$ and update probability $\hat{u}_{j,t}$ from observed history
  Compute expected lifetime $\hat{L}_{j,t}$ and future queries $\hat{N}_{j,t} = \hat{q}_{j,t}\hat{L}_{j,t}$
  Update card components designated online by the deployment protocol; retain frozen components otherwise
  Form $\mathcal{A}_{j,t}$ by applying capability and risk constraints
  for each $e \in \mathcal{A}_{j,t}$ **do**
    Compute transition gain $G_{j,e\leftarrow r,t}$ relative to the incumbent assigned to the candidate's capacity slot
  end for
end for
Select positive-gain transitions $\mathbf{x}_t$ under capacity B_t
Initialize M_{t+1} with the current valid representations
for each placement (j, e) selected by $\mathbf{x}_t$ **do**
  if a current representation for (j, e) is already maintained **then**
    Retain that representation in M_{t+1}
  else
    Acquire an isolated candidate and incur its transaction cost
    Apply transaction verification and observe $A_{j,e,t}$
    if $A_{j,e,t} = 1$ **then**
      Commit it atomically, then release the representation it replaces
      Update M_{t+1} with the committed representation
    else
      Discard it and retain the current serving representation in M_{t+1}
    end if
  end if
end for
Apply capacity-constrained eviction to the successfully realized set
Record transaction and maintenance outcomes in H_{t+1}
Serving phase: execute the active representation for each query
Apply its read check; on rejection, invoke an admissible recovery executor
Record serving, verification, and recovery outcomes and update the cards

A.3 Additional Results

A.3.1 Matched One-Step Uncertainty

The one-step control differs from full DVP only in planning horizon. Across matched streams, full DVP completes 10.95 more correct tasks on the primary workload (paired-bootstrap 95% interval: $[10.33, 11.58]$) and 13.97 more after the shift ($[12.97, 14.95]$).

A.3.2 Matched Adaptive Amortization

Table 9 compares DVP with the matched break-even rule defined above. Both policies use identical event tapes, physical outcomes, online forecasts, cards, verification, fallback, capacity, and budgets. Both expose zero undetected internal errors.

Table 9: **Matched adaptive-amortization control.** Correct tasks and writes are means per stream; intervals are paired 95% bootstrap intervals for DVP minus adaptive amortization.

Workload	Adaptive correct	DVP correct	Adaptive writes	DVP writes	Gain [95% interval]
Primary	156.52	160.33	1.09	1.17	3.80 [3.15, 4.55]
Shifted	256.87	266.24	1.87	2.53	9.38 [8.44, 10.31]

A.3.3 Unseen-Knowledge Cold Start

We evaluate 64 unseen WISE knowledge units from 32 source clusters disjoint from executor-card estimation. Each unit starts from the executor-level prior; every attempted write, including rejection and retry, is charged. Estimating that prior requires 1,024 physical calibration writes totaling 19,548.21 seconds. Table 10 amortizes this one-time campaign over a deployment pool. At the primary 128-query horizon, calibration and deployment writes break even after 464 units; over 1,024 units, the net saving is 23.05 seconds per unit.

Table 10: **Full-cost cold start.** Savings are relative to always external and include deployment writes plus the complete executor-card calibration campaign amortized over 1,024 units.

Future queries per unit	Write attempts per unit	Gross saving (s/unit)	Net saving (s/unit)	Break-even pool size
64	1.28	8.57	−10.52	2,282
128	1.28	42.14	23.05	464
256	1.28	109.29	90.20	179

We next remove the separate calibration campaign and process all 128 physical units sequentially from a Beta$(1, 1)$ executor prior. The card is updated only after an attempted deployment write, and every failed attempt and retry is charged. Table 11 averages 256 matched deployment orders. Online learning has positive net value from 64 future queries onward. At 128 future queries per unit, online card learning saves 5,524.24 seconds per deployment stream relative to always external, but costs 418.99 seconds more than restarting from the conservative prior for each unit. This reversal is consistent with the persistent knowledge-specific difficulty in Figure 3a: online evidence is useful, but a global average cannot replace pair-specific history.

Table 11: **Online card learning without up-front calibration.** Values are cumulative seconds per 128-unit deployment stream.

Future queries per unit	Saving vs. always external	Saving vs. frozen prior
32	−41.63	25.13
64	957.75	651.84
128	5,524.24	−418.99
256	14,130.43	−462.77

Dynamic deployment of unseen knowledge. We place 128 previously unseen WISE units in 256 matched streams. Each stream has 32 epochs and 64 queries per epoch, changes its hot set after epoch 16, uses capacity eight, and stops at a 2,800-second budget. External and internal serving cost 1.5036 and 1.059 seconds per query, and the controller estimates 10.051 seconds per transaction.

Each unit has 16 measured WISE attempts; a policy observes an outcome only after selecting and paying for that attempt, with at most eight attempts per unit. All units begin with a Beta$(1, 1)$ executor-level prior. The posterior combines executor-wide outcomes with pair-specific outcomes of the current unit and is updated only after deployment writes. The read check is the exact target-signature check.

Table 12: **Dynamic placement of unseen WISE knowledge.** Values are means over 256 matched streams. Intervals are paired 95% bootstrap intervals for DVP minus each baseline.

Policy	Correct tasks	Writes	Internal queries	DVP gain [95% interval]
Always external	1,862.00	0.00	0.00	66.42 [60.52, 72.15]
Frozen placement	1,894.68	10.57	542.90	33.75 [28.53, 39.14]
One-step placement	1,862.00	0.00	0.00	66.42 [60.29, 72.29]
DVP	**1,928.42**	18.92	1,018.18	–

DVP activates 15.38 candidates, rejects 3.54 writes, and changes membership after the demand shift. All attempted transactions, including rejections and retries, are charged.

A.3.4 ZsRE Transfer

Table 13 compares retrieval-only service with a higher-cost condition that adds the measured cost of a 4K-context reader. With retrieval alone, all online value-based policies remain external. At 1.5036 seconds per external query, DVP serves an average of 320.31 queries internally and 308.40 externally per stream and completes more correct tasks than the other evaluated online policies. Retrieval and WISE outcomes are identical across the two sensitivity modes; only external cost changes. Across paired streams in the 4K-reader condition, DVP exceeds static placement by 5.92 correct tasks (95% interval $[4.24, 7.67]$) and always external by 70.73 ($[69.75, 71.68]$).

Table 13: **ZsRE external-cost sensitivity.** Values are means per stream under the common 900-second budget. "Internal" is the number of queries served by WISE after verification.

	Retrieval only (0.0256 s/query)			Higher-cost sensitivity (1.5036 s/query)		
Policy	Correct	Writes	Internal	Correct	Writes	Internal
Always external	1074.78	0.00	0.00	475.40	0.00	0.00
Always internal	330.96	53.65	329.36	300.64	51.98	299.19
Static placement	1074.78	0.00	0.00	540.21	7.26	293.11
One-step placement	1074.78	0.00	0.00	475.40	0.00	0.00
DVP	**1074.78**	0.00	0.00	**546.13**	8.00	320.31
Hindsight oracle	1074.78	0.00	0.00	572.29	9.95	415.99

A.3.5 Lifecycle Cost Decomposition

Table 14 uses the online posterior version reported in the main results. Executor time includes every attempted write and every internal or external query, including recovery. Verification and controller time are shown separately. Storage is enforced through the capacity constraint and has no additional wall-clock charge.

Table 14: **Complete wall-clock cost decomposition.** Mean seconds per stream at the main evaluation budgets.

Workload	Policy	Write	Internal	External	Verify	Control	Total
Primary	Always external	0.00	0.00	3340.04	0.000	0.0000	3340.04
Primary	DVP	237.17	728.31	2157.52	0.442	0.0008	3123.44
Shifted	Always external	0.00	0.00	5398.02	0.000	0.0000	5398.02
Shifted	DVP	513.82	1378.99	3291.28	0.769	0.0013	5184.85

The controller timing uses the exact online posterior decision kernel from the main experiment. It takes 0.0287 ms per decision on the primary workload and 0.0368 ms after the shift. Controller computation plus verification accounts for 0.0126% and 0.0140% of the respective lifecycle budgets.

A.3.6 Paired Main-Policy Comparison

The primary and shifted evaluations use matched event streams for DVP and COVE-style coordination. With 5,000 paired bootstrap resamples, 95% intervals for their difference in correct tasks are 5.88–9.55 on the primary workload and 53.71–61.11 after the shift; both intervals exclude zero.

A.3.7 DEMAND-FORECAST ROBUSTNESS

The online forecast multiplies the recent query rate by the expected lifetime of the current version under a knowledge-specific Beta posterior over update probability. We use the selected prior $\alpha = 1, \beta = 9$ and substitute the posterior mean update probability into the truncated lifetime expression for a simple online estimate. LightShop uses queries from the preceding four epochs, while the posterior uses all version events observed so far. Table 15 scales only the resulting future-query count while keeping event streams and executor outcomes fixed. DVP remains above always external from $0.5\times$ to $3\times$ forecast scaling, supporting the posterior-mean plug-in estimate. Setting the forecast to zero suppresses every write and exactly recovers always external.

Table 15: **Correct tasks under forecast scaling.** Oracle uses realized future queries and provides a ceiling reference.

	$0.25\times$	$0.5\times$	$1\times$	$2\times$	$3\times$	Oracle	External
Primary	152.42	159.04	160.33	161.38	161.66	162.11	136.55
Shifted	257.51	264.86	266.24	266.62	266.67	268.75	202.92

A.3.8 RISK-CERTIFICATE SEMANTICS

The risk certificate is tied to both a verifier and its declared scope. For a deterministic check, risk zero means that an accepted output satisfies the properties directly checked by that procedure; it does not certify properties outside the check. For an empirically calibrated verifier, we use a one-sided Clopper–Pearson upper confidence bound on false acceptance. With neither an exact check nor calibration trials, the certificate is one and the candidate is inadmissible. At 95% confidence, zero false acceptances require 59, 149, and 299 trials to fall below risk limits of 5%, 2%, and 1%, respectively.

In LightShop, the read verifier is the deterministic benchmark-state judge. Its scope is the task-visible final state used to score a query. Each internal and external physical outcome is measured from the same frozen initial state; the stream replay charges an internal attempt and, after rejection, substitutes the paired external outcome. The reported zero exposure risk therefore covers benchmark-visible task failure, not rollback of an already executed, irreversible real-world action. In ZsRE, the transaction receipt checks the canonical completion and the read verifier checks an independent paraphrase completion against a stored target signature; rejected candidates use the declared external service. This certificate covers the normalized target answer only. Parameter restoration is checked separately, while unrelated language-model behavior is not included in the zero-risk claim.

Detectability decision map. Figure 3b enumerates a one-attempt decision problem over write-success probabilities from 0.50 to 0.99 and detectable fractions of failure from 0 to 1. Acquisition and verification cost 0.1 loss units each, immediate fallback costs 1.0, and an accepted target failure costs 10.0. The validator accepts a correct outcome with probability 0.95, a detectable failure with probability 0.05, and a silent failure with probability 0.95. The plotted action minimizes exact expected loss among direct use, verification, and immediate fallback at each grid point.

A.3.9 IMPERFECT-VERIFIER CALIBRATION

We additionally apply controlled verifier errors to 256 measured WISE write outcomes, using disjoint halves to calibrate and evaluate a one-sided 95% false-acceptance bound. At a 5% risk limit and a 10% false-reject rate, increasing the verifier's false-accept probability from 0 to 0.4 reduces the probability of admitting the internal executor from 1.00 to 0.27. At a false-accept probability of 1.0, the underlying false-acceptance risk is 5.18%, above the declared limit, and admission falls to 0.34%. Thus, the risk gate increasingly withholds uncertain internal service as verification quality deteriorates.

Independent trace verifier. We also evaluate a verifier separate from the benchmark judge. Its inputs are deployment-observable metadata and traces: latency, token counts, termination, tool identities, tool order, and tool-response status. Target states, judge outputs, success labels, and counterfactual outcomes are excluded from the input. Success labels provide offline supervision only. The protocol uses four physically disjoint groups: 256 tasks fit a shallow decision tree, 128 select a fixed

Table 16: **Risk gating with an imperfect verifier.** The risk limit is 5% and the verifier false-reject probability is 10%. Rates average 5,000 random half-splits; calibration and evaluation are disjoint within each split.

False-accept probability	Executor admitted	Queries served internally	Undetected errors
0%	100.0%	85.7%	0.00%
5%	96.2%	82.7%	0.23%
10%	88.3%	76.2%	0.42%
20%	65.3%	56.5%	0.63%
40%	27.0%	23.6%	0.60%
80%	1.8%	1.6%	0.10%
100%	0.3%	0.3%	0.03%

release threshold, 256 independently certify that threshold, and 256 form the untouched final test. The 10% operating point accepts 108 certification traces with five errors, giving a one-sided 95% upper bound of 9.49%. This is a marginal certificate for accepted traces drawn from the declared certification mixture; it is not a separate conditional guarantee for every rule, version, knowledge unit, or workload state. On final test, it serves 104 of 256 queries internally. Seven accepted outputs are incorrect, giving a 6.73% conditional error rate among accepted outputs and a 2.73% joint exposure rate over all queries. Complete-path accuracy increases from 49.22% for external service to 57.42%, while mean serving time falls from 32.37 to 26.97 seconds. The stricter 5% operating point is not certified and therefore abstains.

Using this verifier throughout the complete dynamic replay, all policies use the same event tapes, budgets, physical outcome pools, forecasts, and executor cards. Table 18 reports the complete replay. The aggregate certificate is frozen before replay; because a dynamic workload can reweight rule–version groups, the replay reports realized exposure rather than extending the 10% certificate to each workload mixture.

Table 17: **Independently selected and certified trace verifier.** Certification uses a one-sided 95% false-acceptance bound; final metrics use a separate 256-task physical test set.

Risk limit	Certified upper bound	Internal served	Accepted error rate	All-query exposure	Final accuracy	Serving time (s)
5%	not certified	0.00%	0.00%	0.00%	49.22%	32.37
10%	9.49%	40.63%	6.73%	2.73%	57.42%	26.97

Table 18: **Dynamic replay with the certified trace verifier.** Means use 256 primary and 512 shifted matched streams. Brackets give paired 95% bootstrap intervals for DVP minus each baseline.

Workload	Policy	Correct	Writes	Exposed errors	DVP gain [95% interval]
Primary	Always external	134.40	0.00	0.00	13.39 [11.90, 14.93]
	Frozen placement	137.74	0.30	0.18	10.05 [8.58, 11.60]
	One-step placement	139.91	0.22	0.50	7.88 [7.10, 8.66]
	DVP	**147.79**	1.28	0.66	–
Shifted	Always external	191.87	0.00	0.00	38.19 [36.03, 40.36]
	Frozen placement	221.57	1.96	2.57	8.49 [6.84, 10.16]
	One-step placement	218.76	0.95	2.73	11.30 [10.03, 12.55]
	DVP	**230.06**	2.49	3.16	–

A.3.10 Metric Decomposition

This decomposition separates the number of tasks attempted within the budget from accuracy among those attempts.

A.3.11 Objective Weight and Capacity

Table 20 varies λ, measured in seconds per correct task. DVP remains above always external across the tested range. Between $\lambda = 75$ and 200, correct tasks range from 160.07 to 161.09 on the primary workload and from 266.12 to 266.52 after the shift.

Capacity zero recovers always external. Capacity one gives 160.33 and 266.24 correct tasks on the primary and shifted workloads. Increasing capacity from one to four leaves these values unchanged

Table 19: **Attempted tasks and accuracy among attempts** at the main lifecycle budgets.

	Primary		Shifted	
Policy	Attempted	Accuracy	Attempted	Accuracy
Always external	249.82	54.7%	381.76	53.2%
Static placement	250.89	55.5%	385.23	56.1%
One-step placement	255.35	58.5%	396.33	63.6%
UniMem-style	226.88	55.1%	368.07	59.3%
COVE-style	246.62	61.9%	338.62	61.7%
DVP	255.69	62.7%	395.41	67.3%

Table 20: **Correct tasks as the success value λ changes.**

λ	0	25	50	75	100	150	200	400
Primary	156.52	158.89	159.66	160.07	160.33	160.65	161.09	143.00
Shifted	256.87	264.65	265.68	266.12	266.24	266.43	266.52	264.82

because, in these streams, no second representation has positive lifecycle gain at the same decision epoch. In a matched capacity-stress replay that changes only external-service time, a second representation becomes profitable on the shifted workload: at $3\times$ external cost, capacity two makes 0.54 additional writes and completes 5.68 additional tasks per stream relative to capacity one; at $4\times$, the gains are 0.37 writes and 3.68 tasks. Thus, the solver uses additional capacity when multiple representations have positive lifecycle value rather than filling it unconditionally.

A.3.12 How Much Reliability History Is Needed?

The main analysis uses eight calibration writes per knowledge unit. Earlier history is already informative: one observed failure flags 7.0% of units that contain 75.3% of failures in the fixed held-out half; using two attempts flags 9.4% of units and captures 89.6% of held-out failures. Across 500 alternative 8/8 splits, the gap between flagged and unflagged held-out failure rates has a 5th–95th percentile range of 65.2–75.0 percentage points, and the flagged units capture at least 96.2% of held-out failures in 95% of splits. The 500 splits resample the fixed 16 physical attempts for each item. Opaque associations retain the ZsRE prompt and subject with a matched arbitrary target and fail on 9.6% of writes. Natural relations retain the original ZsRE target and fail on 5.7%; item-level history further separates reliability within each structure.

A.4 Scope of Evidence

The LightShop policy comparison replays matched event tapes over measured physical executor outcomes and wall-clock costs. Its internal representations are independently maintained LoRA adapters; the ZsRE writer uses WISE side memory. The exact LightShop certificate covers the benchmark-visible final state, while the ZsRE certificate covers normalized target answers and checks parameter restoration separately. The independent trace verifier reports the risk–coverage tradeoff obtained from deployment-visible signals. Together, these evaluations cover controlled matched replay, independently maintained representations, and behavior named by each serving contract.

A.5 Limitations

Our experiments cover two benchmarks and representative external and parametric memory executors. Extending DVP to broader agent environments, additional memory mechanisms, and live services with time-varying costs and reliability is an important direction for future work.