GOVERNMEM: GOVERNED EVOLUTION OF PERSISTENT AGENT MEMORY THROUGH FAILURE REPAIR

Anonymous authors
Paper under double-blind review

ABSTRACT

Persistent memory helps language agents retain knowledge and experience, but it also allows errors to affect later decisions. Repairing these errors provides useful feedback. The challenge is to turn this feedback into reliable guidance for future repairs. We propose `GovernMem`, a framework that learns from memory repairs without updating model weights. It connects repair execution and learning through two loops. The fast loop diagnoses a memory fault, selects a repair skill, and verifies the changes before applying them. The slow loop collects feedback on whether the repair worked, what it cost, and whether the fault returned. To use this experience for skill selection, we further propose `GovernRouter`. It scores eligible skills with a frozen language model, adjusts the scores using past repair outcomes when enough evidence is available, and selects the highest-scoring skill. `GovernMem` also manages skill versions and keeps feedback linked to the version that produced it. With skill content fixed, reusing experience collected by another model on the same cases and in the same order improves repair utility by 6.39 points on Meta-Llama-3.1-8B-Instruct and 6.63 points on Qwen3-14B over starting without prior experience. Moreover, `GovernRouter` ranks first among seven routing configurations under the evaluated settings. These results suggest that agents can improve memory repair by learning when to use existing skills, without changing the skills themselves.

1 INTRODUCTION

Persistent memory helps language agents retain knowledge and experience across interactions (Li et al., 2025; Xu et al., 2025; Li et al., 2026). However, it also allows errors to persist (Garg et al., 2026; Tan et al., 2026). In other words, an outdated fact or a poisoned memory entry can affect many later decisions. Agents therefore need to repair their memories. They should also learn from these repairs, so that experience with one failure helps them handle the next. The challenge is to ensure that the experience they reuse is reliable.

Previous work has studied both memory repair and experience reuse. Methods such as MemAudit (Tan et al., 2026) and MemTrace (Deng et al., 2026) detect harmful memories and trace the sources of errors. Other methods store reflections, reusable programs, or memory-operation skills to improve future behavior (Shinn et al., 2023; Wang et al., 2023; Zhang et al., 2026a). However, how to turn repair outcomes into reliable guidance for future repairs remains an open problem. Fixing a memory error does not necessarily mean that the same repair should be used again.

Learning from repairs is difficult for two reasons. First, an immediate fix may not remain effective. A repair can remove an error but also change useful information, incur high costs, or allow the error to return later. The agent should therefore wait for enough feedback before judging the repair. Second, repair skills can change over time. A result obtained with one version may not support a later version. To learn reliably, the agent needs to keep each outcome linked to the repair that produced it, including its conditions, costs, and skill version. We therefore ask: **How can agents use verified repair outcomes to improve future repair decisions?**

In this paper, we introduce `GovernMem`, a framework that learns from memory repairs without changing model weights. It connects two loops: a *fast* loop repairs the current memory fault, while a *slow* loop uses repair outcomes to improve future skill selection and manage repair skills. An explicit state machine controls when repairs are applied and when their outcomes can be used for learning.

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.

The *fast* loop first diagnoses a memory fault and identifies the skills that can repair it. Selecting among these skills requires both an assessment of the current fault and experience from past repairs. To combine the two, we further propose `GovernRouter`. It starts with scores from a frozen language model and adjusts them using each skill's past performance, both overall and on similar failures, with further adjustments for the current context. These adjustments take effect only when enough evidence supports them. The selected skill then runs on a copy of memory, and its changes are applied only after verification.

The *slow* loop evaluates the repair once the required feedback is available. It records whether the repair worked, what it cost, and whether the fault returned. These outcomes update the experience used by `GovernRouter` in later decisions. The loop also manages skill versions, using verified results to guide promotion, quarantine, and retirement. Each outcome remains linked to the exact version that produced it. In this way, past repairs improve both which skill the agent selects and which skills remain available.

We evaluate `GovernMem` on three types of memory failures: process faults, state drift, and poisoned memory. The results show that agents make better repair choices when they reuse past repair experience. Compared with starting without prior experience, using repair experience collected by another model on the same cases and in the same order improves repair utility by 6.39 points on Meta-Llama-3.1-8B-Instruct and 6.63 points on Qwen3-14B. These gains are measured on shared cases with model weights and skill content held fixed. `GovernRouter` also achieves the highest repair utility among the seven routing configurations tested. Ablations show that most of the improvement comes from learning how well each skill performs overall, with additional gains from local correction and support gating.

Our contributions are threefold:

- We propose `GovernMem`, a framework that connects memory repair with experience reuse through two loops. It verifies memory changes and uses repair outcomes to improve future decisions without updating model weights.
- We introduce `GovernRouter`, which selects repair skills by combining frozen-model scores with past repair experience. It learns from each skill's performance and requires enough supporting evidence before adjusting its scores.
- We integrate skill versioning and lifecycle management into the repair–learning loop. Each outcome remains linked to the skill version that produced it, providing evidence for skill promotion, quarantine, and retirement.

2 Related Work

Early work on agent memory focuses on storing and retrieving external information. RAG retrieves knowledge to augment model context (Lewis et al., 2020), while Generative Agents, MemoryBank, and MemGPT organize interaction histories into retrievable and updatable memories (Park et al., 2023; Zhong et al., 2024; Packer et al., 2023). Subsequent work extends memory systems toward long-term state management: A-MEM dynamically updates memory associations (Xu et al., 2025), MemoryOS uses hierarchical storage (Kang et al., 2025), and AgeMem incorporates memory operations into agent decisions (Yu et al., 2026). LoCoMo and LongMemEval evaluate cross-session retrieval, temporal reasoning, knowledge updates, and abstention (Maharana et al., 2024; Wu et al., 2025). These studies advance dynamic memory management but primarily focus on information organizing and reusing.

Another line of work addresses error accumulation and security risks in persistent memory. AgentPoison and PoisonedRAG show how poisoned content influences agent behavior through retrieval (Chen et al., 2024; Zou et al., 2025). HaluMem evaluates hallucinations and missing information in memory processing (Chen et al., 2025), MemFail constructs diagnostic tasks for memory operations (Garg et al., 2026), and MemTrace traces errors through execution graphs (Deng et al., 2026). MemAudit combines counterfactual analysis and structural detection to identify and remove poisoned memories (Tan et al., 2026). These studies support fault diagnosis and repair, but ensuring reliable memory updates remains a challenge.

Research on experience reuse increasingly extends to skill evolution. Reflexion stores reflections (Shinn et al., 2023), ExpeL extracts experience rules (Zhao et al., 2024), Voyager accumulates code

skills (Wang et al., 2024), and Agent Workflow Memory extracts reusable workflows (Wang et al., 2025). More recent methods develop evolving memory and skill systems: Evo-Memory evaluates test-time memory updates (Wei et al., 2025), MemSkill learns memory-operation skills (Zhang et al., 2026a), and Mem^{2}Evolve uses experience to create tools and expert agents (Cheng et al., 2026). Double Ratchet studies feedback reliability during co-evolution (Zhang et al., 2026b), MSCE develops callable skills from execution evidence (Tang et al., 2026), and ERSkill and CoEvo-Mem study evolving retrieval skills and the co-evolution of routing policies and memory banks (Chen et al., 2026; Ye et al., 2026). These methods support adaptation without weight updates, but learning reliably from repair outcomes remains underexplored.

GovernMem addresses these challenges by using verified repair outcomes as reusable experience for future skill selection and lifecycle management. Linking each outcome to the skill version makes this experience attributable across successive skill updates.

3 METHODOLOGY

`GovernMem` operates through two coupled loops: a *fast* loop for diagnosing memory errors, selecting repair skills, and verifying repairs, and a *slow* loop for learning from repair outcomes and managing skill versions. A skill is a versioned repair program with conditions on memory changes. `GovernRouter` selects eligible skills using model scores and past outcomes. Repairs run on a memory copy and commit only after verification. Delayed feedback remains linked to the executed version and informs subsequent routing, promotion, quarantine, or retirement, as shown in Figure 1.

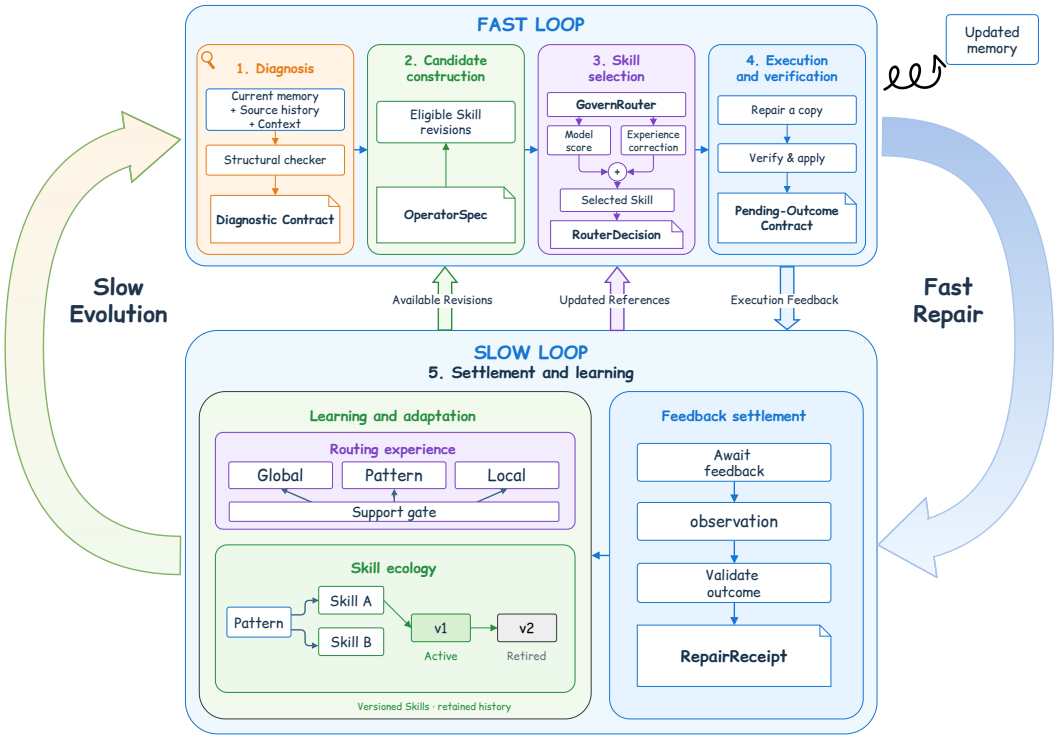


Figure 1: The *fast* and *slow* loops of `GovernMem`. The *fast* loop diagnoses, selects, and verifies a memory repair. The *slow* loop settles its outcome and updates routing experience and skill ecology. Both updates guide subsequent repairs.

3.1 BACKGROUND AND TASK DEFINITION

Memory repair changes persistent state, making execution traceability essential for interpreting its outcomes. Past outcomes can guide future repair choices, but delayed feedback and changes to skill versions complicate their reuse.

This leads to a sequential decision problem with delayed feedback and a changing set of repair actions. At event t, let M_t denote the current memory, $H_{\leq t}$ the available event history, A_t the set of eligible skill revisions, and E_t the verified repair outcomes available before the decision. The agent selects

$$a_t = \pi(M_t, H_{\leq t}, A_t, E_t), \qquad a_t \in A_t \cup \{\text{no repair}\}. \tag{1}$$

The objective is to improve repair utility by learning from E_t while keeping model weights fixed. Each outcome in E_t remains associated with the skill revision that produced it.

The following subsections address three parts of this problem. Section 3.2 constructs A_t and governs repair execution and feedback collection. Section 3.3 uses verified outcomes to improve the selection policy π. Section 3.4 manages skill revisions and their availability for subsequent decisions.

3.2 Executing Verified Memory Repairs

The state machine controls memory changes and learning through five stages, The state machine governs memory changes and learning across five stages, each with an explicit contract specifying its preconditions, permitted operations, and postconditions. The first four stages form the *fast* loop; settlement connects them to the *slow* loop. Appendix B.2 gives the contract schemas.

1. Diagnosis. The *fast* loop checks memory against immutable source history. Structural inconsistencies indicate process faults, verifiable replacement relations indicate state drift, and active untrusted events outside quarantine indicate poison. A `Diagnostic Contract` records the fault and supporting evidence. Diagnosis uses only information available at the current event and abstains on ambiguous cases.

2. Candidate construction. The action contract, `OperatorSpec`, defines skill preconditions, permitted reads and writes, invariants, mutation bounds, and rollback operations. Registered versions that satisfy these conditions and service-status requirements form the legal action set.

3. Skill selection. `GovernRouter` selects the highest-scoring eligible skill using Section 3.3. The selection contract, `RouterDecision`, records the exact version, model scores, and routing state for later feedback attribution.

4. Execution and verification. The selected skill runs on an isolated memory copy. Verification checks fault removal, structural invariants, safety, and mutation scope. Commitment also requires live memory to match the starting version. Failed checks or version conflicts discard the proposed changes. A `Pending-Outcome Contract` records the attempt, state identities, and commit or rollback status.

5. Settlement and learning. After the observation window ends and feedback arrives, settlement links outcomes and costs to the repair through `RepairReceipt`. Source, timing, decision, and version checks determine eligibility for routing and lifecycle updates. Each outcome is used once. Outcomes still awaiting observation at evaluation end are right-censored and do not update learned state.

3.3 Learning from Repair Outcomes

`GovernRouter` corrects frozen-model utility predictions using errors observed in past repairs. Corrections capture overall skill performance (Global), recurring fault structure (Pattern), and current routing features (Local), as shown in Figure 2.

Action scoring. Let A_t contain eligible skill versions and $b_t(a)$ denote the frozen-model utility prediction for version a at event t. The router selects by

$$\begin{aligned} q_t(a) &= b_t(a) + \Delta_t(a) + z_t(a), \\ a_t &= \arg\max_{a \in A_t} q_t(a). \end{aligned} \tag{2}$$

where $\Delta_t(a)$ is the learned correction and $z_t(a)$ is the exploration term.

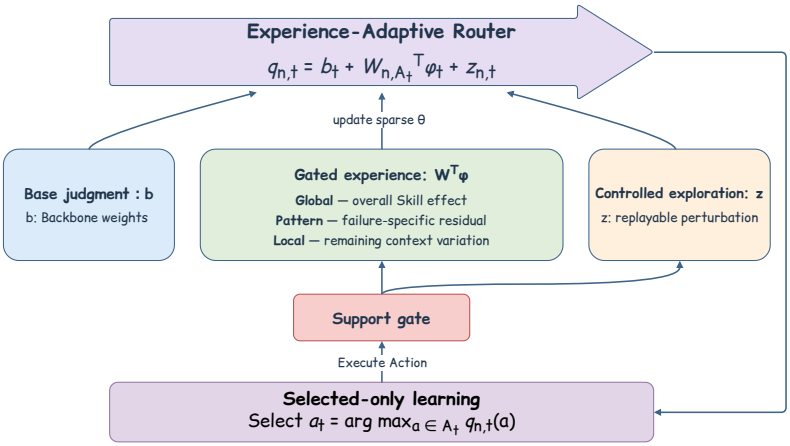


Figure 2: `GovernRouter` learns hierarchical corrections to backbone utility predictions. Supported corrections and exploration determine the selected skill. Mature feedback updates Global, Pattern, and Local experience for the executed revision. The term $W^{\top}\phi$ in the diagram is expanded as Δ_t in Equation 3.

Hierarchical correction. Global estimates overall prediction error for a skill version. Pattern learns the remaining error for recurring faults, and Local further accounts for current routing features.

Each coordinate k stores precision Λ_k and natural parameter η_k, initialized to 1 and 0. The mean $\mu_k = \eta_k/\Lambda_k$ gives the correction estimate, and effective support $s_k = \Lambda_k - 1$ determines its eligibility:

$$\Delta_t(a) = \sum_{k \in I_t(a)} \mathbf{1}\left[s_k \geq \tau_{\text{level}(k)}\right] w_{t,k}\mu_k. \tag{3}$$

Here, $I_t(a)$ contains active coordinates for version a, and $\tau_{\text{level}(k)}$ is the level-specific support threshold. Weights $w_{t,k}$ are 1 for Global, $\rho_{t,p}$ for Pattern, and $\rho_{t,p}x_{t,j}/\|x_t\|_1$ for Local. Responsibility $\rho_{t,p}$ measures association with pattern p, and x_t contains decision-time features. Zero-norm features activate no Local coordinates.

Support and exploration. Corrections affect selection only after meeting their support thresholds. Exploration requires sufficient Global support for at least two eligible skills. Weighted zero-mean Gaussian perturbations, with variance decreasing as precision grows, sum to $z_t(a)$. Without active corrections or exploration, the router retains the frozen-model choice. Appendix C specifies thresholds and reproducible sampling.

Feedback updates. Mature receipts provide utility based on structural outcomes and costs. Let u_i be the effective utility of repair i: failure, noncommit, rollback, invalidity, or later degradation receives -1; successful repairs incur penalties for mutation scope, collateral changes, execution cost, delay, and recurrence. With stored pre-execution prediction $\hat{u}_i$, the residual target is

$$r_i = \text{clip}(u_i - \hat{u}_i, -1, 1). \tag{4}$$

Each level learns the error remaining after preceding estimates:

$$y_{i,k} = \begin{cases} r_i, & k \text{ is Global,} \\ r_i - \mu^{-}_{g(i)}, & k \text{ is Pattern,} \\ r_i - \mu^{-}_{g(i)} - \mu^{-}_{p(i,k)}, & k \text{ is Local.} \end{cases} \tag{5}$$

Here, $g(i)$ identifies the Global coordinate of the selected version, $p(i,k)$ identifies the Pattern coordinate for Local coordinate k, and $-$ denotes pre-update means. Active coordinates of the executed version update as

$$\Lambda_k \leftarrow \Lambda_k + w^2_{i,k}, \qquad \eta_k \leftarrow \eta_k + w_{i,k}y_{i,k}, \tag{6}$$

where $w_{i,k}$ is the weight recorded at selection. These updates revise correction estimates and support for subsequent decisions.

3.4 Evolving Repair Skills

Learning from repairs changes both which skill the router prefers and which versions the agent can use. The skill library stores the repair programs and their versions. We use the term skill ecology to describe how these skills cover recurring faults, compete with one another, and move through their lifecycles.

Failure evidence and promotion. Failure memory records individual faults and their sources. Patterns describe recurring fault structures, and skills provide repairs for them. Several skills may address the same pattern, and one skill may address several patterns. This allows alternative repairs to be compared on similar failures.

Successful repairs provide evidence for the version that was executed. A probationary skill becomes eligible for promotion after succeeding on at least three distinct later failures from at least two audit families. The failure that produced the skill does not count toward this requirement. Promotion also requires the governance checks to pass and no regression on the locked anchor.

Version management. Changing the repair logic, parameter constraints, or applicability of a skill creates a new version linked to its predecessors. The version enters service after passing stability verification and meeting the service requirements. Its feedback and routing experience are stored under its own identifier, starting from an empty prior unless experience is imported. Safety, provenance, or integrity anomalies can trigger quarantine from the next event. Retirement and supersession also remove versions from service while preserving their programs and supporting evidence.

Coverage and competition. A niche summarizes repair needs and skill use over a time window. It records fault occurrence, the fraction of unresolved faults, skill usage, and performance differences. The governance layer uses these records to manage skill availability. An append-only ledger links routing updates and version changes to the evidence behind them. Appendices B.5 and B.6 give the full lifecycle rules.

4 Experiments

We evaluate whether verified repair experience improves subsequent repair decisions with model weights and skill content fixed. The main experiments measure the gains from experience reuse and compare routing configurations; ablations identify the contributions of hierarchical experience, support gating, and exploration.

4.1 Experimental Setup

Datasets and models. GovernMem-RepairStream introduces process faults, state drift, and poison into public interaction content from HaluMem (Chen et al., 2025), MemFail (Garg et al., 2026), and MemTraceBench (Deng et al., 2026). Twenty episodes per source yield 58, 29, and 49 repair cases, respectively, totaling 136 cases including clean controls. Each source is evaluated in stationary and abrupt orders. Qwen2.5-14B-Instruct (Qwen et al., 2024) supplies source experience; Meta-Llama-3.1-8B-Instruct (Grattafiori et al., 2024) and Qwen3-14B (Yang et al., 2025) are the target models. Source and target evaluations use the same cases, with target weights and skill content fixed. Separate family-disjoint skill-transfer tests appear in Appendix E.4.

Evaluation metrics. Repair utility in $[-1, 1]$ combines structural repair outcomes with predefined cost penalties; one utility point equals 0.01. Locality measures logical modification scope. Regret_7 and Regret_6 measure the gap to the best realized outcome among the seven routing and six ablation configurations, respectively. The negative override rate measures how often changing the backbone choice lowers utility; it averages stream rates weighted by event count, excluding streams without overrides. Feedback uses the structural recurrence proxy defined in Appendix D.2.

Baselines. We compare `GovernRouter` with six controls: Random legal samples eligible actions; Fixed-order legal uses a fixed action order; Global Thompson and Niche Thompson use signed-utility posterior updates per skill and per niche–skill pair, respectively; Contextual bandit uses diagonal

optimistic scoring; and `GovernRouter-P` uses hierarchical posterior sampling. All configurations share legal actions, budgets, cached backbone predictions, event order, and feedback rules. Under Matched initialization, `GovernRouter` and `GovernRouter-P` import source state, whereas Thompson and contextual controls start empty. The comparison therefore evaluates these routing configurations with their stated initializations. Appendix C specifies each control.

Experimental settings. Paired experience-reuse runs compare empty routing state (Reset) with source experience matched to the data source and event order (Matched), holding cases, seeds, predictions, and budgets fixed. Both conditions update from mature selected-action feedback; immature outcomes are right-censored. Reuse comprises 144 runs, while routing and ablation each comprise 24 runs. Global, Pattern, and Local support thresholds are 2, 4, and 8. Exploration uses scale 0.08 and requires at least two legal skills with Global support of at least 4. Three frozen Patterns represent the diagnosed fault classes. Local uses normalized event index, giving positive indices unit weight and implementing an additional Pattern-conditioned residual stage.

Aggregation and uncertainty. Table 1 reports stream-mean utility and event-mean locality and regret; Table 2 reports event-mean utility and regret. Paired effects first average within failure families and then weight families equally, so they need not equal table differences. Pointwise 95% confidence intervals use family-blocked paired bootstrap, keeping order, seed, and intervention variants together. Appendix D.1 provides supplementary protocols and results.

4.2 Effectiveness of Experience Reuse

Retained experience improves repair utility. Matched improves family-balanced utility over Reset by 6.39 points on Llama and 6.63 points on Qwen3, with 95% confidence intervals of $[5.04, 7.79]$ and $[5.06, 8.24]$. Each model contributes 451 paired mature events across 49 failure families (Appendix Table 8). With model weights, skill content, and evaluated cases fixed, this comparison isolates the benefit of initializing future repair decisions with retained source experience. The gains show that the same skill library can support better repair choices when its execution history informs selection, providing a way to improve decisions without retraining the target model or rewriting the skills. Because Reset also learns online, the gain highlights the value of carrying prior repair experience into subsequent decisions. This reuse allows a different backbone to benefit from accumulated repair outcomes when handling a previously encountered workload.

Gains extend across failure families. Figure 3 shows positive mean gains in 46 of 49 families on Llama and 40 of 49 on Qwen3. Experience reuse thus benefits a broad range of failure families in the evaluated streams. Positive family means on both backbones show that the aggregate gain accompanies improvements across many repair conditions. Supplementary comparisons of pooled and prefix initialization appear in Appendix Table 8.

Routing improves utility with smaller modifications. `GovernRouter` achieves the highest utility and lowest Regret_7 on both models among the evaluated configurations (Table 1). Family-balanced gains over Fixed-order legal are 5.84 points on Llama and 6.10 on Qwen3; gains over `GovernRouter-P` are 10.76 and 8.60 points. All 12 pointwise confidence intervals against the six controls are positive (Appendix Figure 20). Relative to Fixed-order legal, modification scope falls from 3.412 to 2.767 on Llama and 2.957 on Qwen3, reductions of 18.89% and 13.34%. These results show that the utility gains accompany more localized repairs; locality penalties are included in the utility measure. Compared with the fixed-order selector, the router therefore improves the measured balance between repair outcomes and modification scope.

4.3 Contributions of the Routing Mechanisms

We compare six configurations on 301 paired mature events per model (Table 2). Starting from Frozen backbone, we add Global, Pattern, and Local corrections with support gating and no exploration. `GovernRouter` adds supported exploration. Full w/o support gate retains all three levels and exploration while removing their support thresholds.

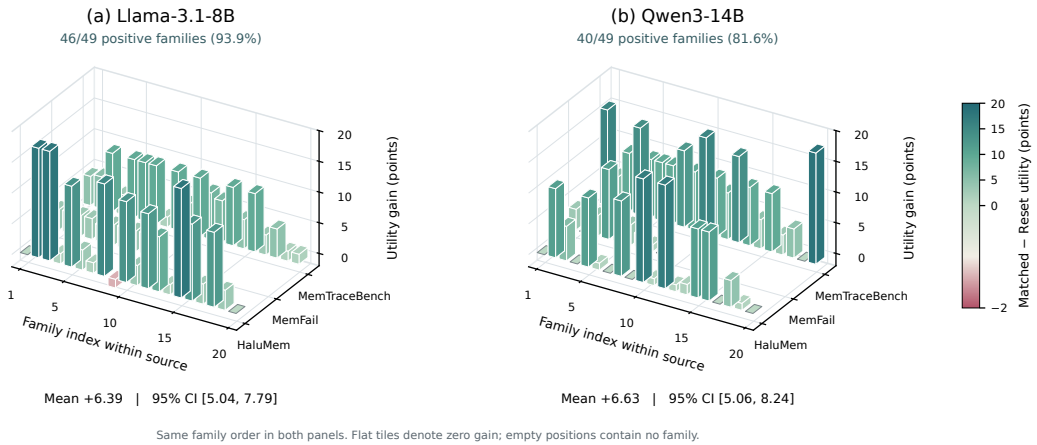


Figure 3: Experience reuse across failure families. Bars show mean paired Matched-minus-Reset utility across orders, seeds, and intervention variants, grouped by data source. Both panels use the same family ordering; flat tiles indicate zero gain and empty positions contain no family. Summaries give equally weighted means over 49 families and pointwise 95% family-blocked bootstrap intervals, based on 451 paired mature events per model. One utility point equals 0.01.

Table 1: Routing performance over 12 streams and 301 mature events per model. Utility measures structural repair outcomes with cost penalties; locality measures modification scope. Regret_7 is the gap to the best realized outcome among the seven configurations. Utility averages streams; the other metrics average events. `GovernRouter` and `GovernRouter-P` import matched source experience; other adaptive controls start empty. Bold marks the best value per column.

	Llama-3.1-8B			Qwen3-14B		
Method	Utility ↑	Locality ↓	Regret_7 ↓	Utility ↑	Locality ↓	Regret_7 ↓
Random legal	0.5436	2.153	0.2024	0.5436	2.153	0.2020
Fixed-order legal	0.6695	3.412	0.0953	0.6695	3.412	0.0949
Global Thompson	0.6056	2.482	0.1511	0.6056	2.482	0.1506
Niche Thompson	0.5662	2.412	0.1603	0.5662	2.412	0.1599
Contextual bandit	0.5962	**2.113**	0.1800	0.5962	**2.113**	0.1795
`GovernRouter-P`	0.6352	2.299	0.1369	0.6441	2.821	0.0991
`GovernRouter`	**0.7222**	2.767	**0.0363**	**0.7220**	2.957	**0.0315**

Table 2: Ablations on 301 paired mature events per model. Utility and Regret_6 are event means; regret references the best realized outcome among these six configurations. Neg. is the percentage of overrides with lower utility than the frozen backbone, aggregated as event-weighted stream rates. It is undefined for Frozen backbone. Bold marks the best value per column.

	Llama-3.1-8B			Qwen3-14B		
Method	Utility ↑	Regret_6 ↓	Neg. (%) ↓	Utility ↑	Regret_6 ↓	Neg. (%) ↓
Frozen backbone	0.6511	0.0911	—	0.6396	0.0864	—
Global	0.6891	0.0531	13.52	0.7166	0.0094	6.11
Global+Pattern	0.6926	0.0496	8.61	0.7167	0.0093	8.52
Global+Pattern+Local	0.7033	0.0389	7.46	**0.7210**	**0.0050**	6.44
Full w/o support gate	0.6837	0.0585	14.53	0.7150	0.0109	5.88
`GovernRouter`	**0.7141**	**0.0281**	**3.13**	0.7208	0.0052	**4.82**

Global experience supplies the main gain. `GovernRouter` improves family-balanced utility over Frozen backbone by 6.15 points on Llama and 7.74 points on Qwen3. Global supplies the largest sequential gain, at 4.12 and 7.35 points. Pattern adds 0.27 and −0.02 points, while Local adds

0.92 and 0.43 points over Global+Pattern, with positive confidence intervals on both models. These contrasts identify Global experience as the main contributor and show additional gains from the Local residual stage (Appendix Figure 21). The results support a coarse-to-fine correction scheme: skill-level history supplies most of the benefit, while subsequent residual stages refine the choices supported by that history. This suggests that skill-level reliability provides a useful foundation for selection, with Local corrections adding refinement.

Support gating reduces harmful overrides. Enabling support gating raises family-balanced utility by 2.33 points on Llama and 0.71 points on Qwen3, with positive confidence intervals. Negative override rates fall from 14.53% to 3.13% and from 5.88% to 4.82%, respectively. Figure 4 shows how these gains arise at the event level. On Llama, 32 below-backbone and 32 above-backbone outcomes move to equality, so the net utility effect depends on the magnitude as well as the number of changes. Utility remains unchanged on 234 of 301 Llama events and 273 of 301 Qwen3 events. Gating therefore improves aggregate utility through a subset of decisions while lowering harmful override rates. Its practical value is to control when historical corrections replace the backbone preference: requiring sufficient support improves the measured quality of those interventions. The simultaneous utility gains and lower negative override rates show that support gating improves both the overall outcome and the reliability of experience-based overrides.

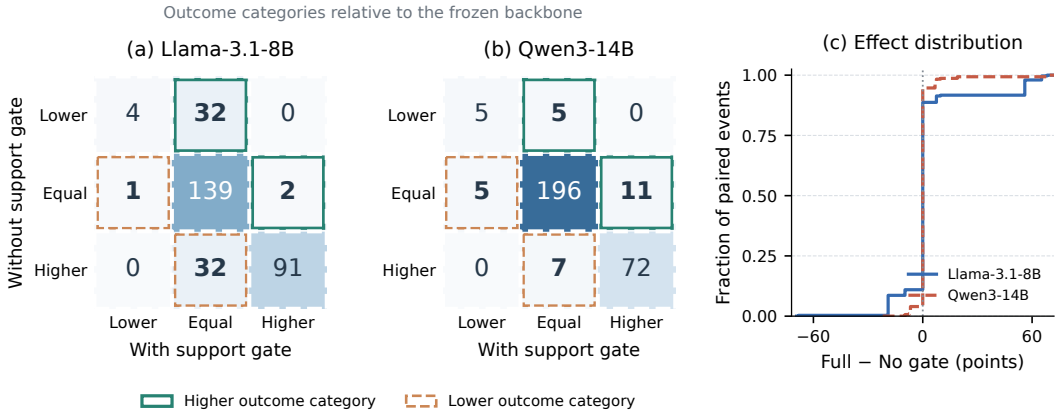


Figure 4: Event-level effects of support gating on 301 paired mature events per model. (a,b) Rows and columns categorize utility relative to the frozen backbone without and with gating, respectively. Cells report event counts; green and orange outlines mark shifts to higher and lower categories. (c) Empirical cumulative distributions of utility with gating minus utility without gating, including unchanged events. All panels use structural replay outcomes; one utility point equals 0.01.

Supported exploration improves routing. Adding exploration to Global+Pattern+Local raises Llama event-mean utility from 0.7033 to 0.7141 and lowers negative overrides from 7.46% to 3.13%. Qwen3 utility remains nearly unchanged at 0.7208 versus 0.7210, while negative overrides fall from 6.44% to 4.82%. Exploration improves Llama utility and reduces harmful override rates on both models. The lower negative override rates indicate that supported exploration can improve the quality of departures from backbone choices. Appendix E.5 extends the analysis to temporal adaptation and skill lifecycles.

5 Conclusion

We present GovernMem, a framework that connects persistent-memory repair with experience reuse without updating model weights. Governed repair execution, hierarchical residual routing, and versioned skill management enable repair outcomes to guide future decisions. Experiments on two backbone models show that matched experience reuse improves repair utility by 6.39 and 6.63 points, averaged equally across failure families. Ablations identify global repair experience as the main source of improvement, with additional gains from local correction and support gating. Future work will evaluate independent downstream tasks and longer-term skill development.

AI USE STATEMENT

We used GPT-5.6 GPT-6 claude-fable-5 to assist with writing, literature discovery, and drafting portions of the manuscript. For language editing, it helped translate Chinese draft passages into English, improve sentence structure and readability, reduce repetitive wording, and maintain consistent terminology. It also assisted with section transitions, LaTeX formatting, and the wording of figure captions and table descriptions. We reviewed the revisions to preserve the intended technical meaning.

For retrieval and discovery, GPT-6 helped formulate search queries and identify potentially relevant work on agent memory. These suggestions served as starting points for consulting the original sources. The authors assessed the relevance of the suggested references and checked the descriptions and citation information used in the manuscript.

For drafting, GPT-6 helped turn author-provided notes, method descriptions, and experimental findings into preliminary paragraphs. This included suggesting section organization, connecting related points, and helping express observations and possible explanations in the experimental discussion. The authors revised these drafts, checked interpretations against the available evidence, and made the final decisions about their inclusion. Strictly, LLMs or agents did not generate ANY experimental measurements or reported numerical results.

We reviewed the AI-assisted text and presentation to ensure that they preserved the intended technical meaning and accurately reflected the available evidence. All final decisions regarding scientific claims, result interpretation, and manuscript content remained with us, who take responsibility for the accuracy and integrity of the work.

REFERENCES

Ding Chen, Simin Niu, Kehang Li, Peng Liu, Xiangping Zheng, Bo Tang, Xinchi Li, Feiyu Xiong, and Zhiyu Li. HaluMem: Evaluating hallucinations in memory systems of agents. arXiv preprint arXiv:2511.03506, 2025.

Haolong Chen, Liang Zhang, Zhuo Li, Lei Xue, and Guanrxu Zhu. ERSkill: Evolving for skill-guided adaptive memory retrieval. arXiv preprint arXiv:2608.12720, 2026.

Zhaorun Chen, Zhen Xiang, Chaowei Xiao, Dawn Song, and Bo Li. AgentPoison: Red-teaming LLM agents via poisoning memory or knowledge bases. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-4136.

Zihao Cheng, Zeming Liu, Yingyu Shan, Xinyi Wang, Xiangrong Zhu, Yunpu Ma, Hongru Wang, Yuhang Guo, Wei Lin, and Yunhong Wang. Mem^{2}Evolve: Towards self-evolving agents via co-evolutionary capability expansion and experience distillation. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 20784–20831. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.952. URL https://aclanthology.org/2026.acl-long.952/.

Xinle Deng, Ruobin Zhong, Hujin Peng, Xiaoben Lu, Yanzhe Wu, Guang Li, Buqiang Xu, Yunzhi Yao, Jizhan Fang, Haoliang Cao, Junjie Guo, Yuan Yuan, Ziqing Ma, Yuanqiang Yu, Rui Hu, Baohua Dong, Hangcheng Zhu, and Ningyu Zhang. MemTrace: Tracing and attributing errors in large language model memory systems. arXiv preprint arXiv:2605.28732, 2026.

Ishir Garg, Neel Kolhe, Dawn Song, and Xuandong Zhao. MemFail: Stress-testing failure modes of LLM memory systems. arXiv preprint arXiv:2605.26667, 2026.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, et al. The Llama 3 herd of models. arXiv preprint arXiv:2407.21783, 2024. URL https://arxiv.org/abs/2407.21783.

Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. Memory OS of AI agent. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 25961–25970. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.1318. URL https://aclanthology.org/2025.emnlp-main.1318/.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Dongfang Li, Zixuan Liu, Junmai Wang, Jiahe Huang, Fuhao Li, Bonian Jia, Baotian Hu, and Min Zhang. LycheeMemory V2: Efficient long-term memory for LLM agents via semantic segment-level consolidation. arXiv preprint arXiv:2608.12990, 2026. URL https://arxiv.org/abs/2608.12990.

Zhiyu Li, Shichao Song, Hanyu Wang, Simin Niu, Ding Chen, Jiawei Yang, Chenyang Xi, Huayi Lai, Jihao Zhao, Yezhaohui Wang, Junpeng Ren, Zehao Lin, Jiahao Huo, Tianyi Chen, Kai Chen, Kehang Li, Zhiqiang Yin, Qingchen Yu, Bo Tang, Hongkang Yang, Zhi-Qin John Xu, and Feiyu Xiong. MemOS: An operating system for memory-augmented generation (MAG) in large language models. arXiv preprint arXiv:2505.22101, 2025.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13851–13870. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.747.

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.

Joon Sung Park, Joseph C. O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST '23. Association for Computing Machinery, 2023. doi: 10.1145/3586183.3606763.

Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, et al. Qwen2.5 technical report. arXiv preprint arXiv:2412.15115, 2024. URL https://arxiv.org/abs/2412.15115.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-0377.

Zhewen Tan, Yilun Yao, Huiyan Jin, Wenhan Yu, Guoan Wang, Mengyuan Fan, Liang Lu, Feng Liu, Xiangzheng Zhang, Duohe Ma, Tong Yang, and Lin Sun. MemAudit: Post-hoc auditing of poisoned agent memory via causal attribution and structural anomaly detection. arXiv preprint arXiv:2605.23723, 2026.

Bo Tang, Yang Zhang, Guomian Zhuang, Wenqiang Wei, Gaoyang Zheng, Lindong Xie, Yanchao Tan, Feiyu Xiong, Qingyu Yang, Edward Chung, and Zhiyu Li. From memory to skills: Evidence-grounded co-evolution governance for long-horizon LLM agents. arXiv preprint arXiv:2607.16621, 2026.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=ehfRiF0R3a.

Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. Augmenting language models with long-term memory. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-3259.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 2025.

Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H. Chi, Chi Wang, Shuo Chen, Fernando Pereira, Wang-Cheng Kang, and Derek Zhiyuan Cheng. Evo-Memory: Benchmarking LLM agent test-time learning with self-evolving memory. arXiv preprint arXiv:2511.20857, 2025.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking chat assistants on long-term interactive memory. In *International Conference on Learning Representations*, 2025.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-Mem: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-0593. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/19909c36f51abc4856b4560aff3d36d6-Abstract-Conference.html.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, et al. Qwen3 technical report. arXiv preprint arXiv:2505.09388, 2025. URL https://arxiv.org/abs/2505.09388.

Bowen Ye, Yongchao Xu, Zhijian Li, Xiang Yin, Junkai Ma, and Wenzhao Li. CoEvo-Mem: Co-evolving retrieval policy and memory bank for LLM agents. arXiv preprint arXiv:2608.01739, 2026.

Yi Yu, Liuyi Yao, Yuexiang Xie, Qingquan Tan, Jiaqi Feng, Yaliang Li, and Libing Wu. Agentic memory: Learning unified long-term and short-term memory management for large language model agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 21457–21483. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.981. URL https://aclanthology.org/2026.acl-long.981/.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. MemSkill: Learning and evolving memory skills for self-evolving agents. arXiv preprint arXiv:2602.02474, 2026a.

Xing Zhang, Guanghui Wang, Yanwei Cui, Ziyuan Li, Wei Qiu, Bing Zhu, and Peiyang He. Who grades the grader? co-evolving evaluation metrics and skills for self-improving LLM agents. arXiv preprint arXiv:2607.12790, 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. doi: 10.1609/aaai.v38i17.29936.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19724–19731, 2024. doi: 10.1609/aaai.v38i17.29946.

Wei Zou, Runpeng Geng, Binghui Wang, and Jinyuan Jia. PoisonedRAG: Knowledge corruption attacks to retrieval-augmented generation of large language models. In *34th USENIX Security Symposium (USENIX Security 25)*, pp. 3827–3844. USENIX Association, 2025.

Appendix Guide

The appendix develops the mechanisms and evidence behind GovernMem in six parts: formal specification (Appendix A), governed execution and Skill lifecycles (Appendix B), routing implementation (Appendix C), evaluation protocol (Appendix D), supporting results (Appendix E), and detailed experimental figures (Appendix F). The results are organized around experience reuse and its mechanisms, followed by adaptation, transfer, locality, concurrency, and lifecycle traces.

A Formal Method Specification

The following specifications supplement the compact presentation in Section 3.

A.1 Problem Formulation

Let t denote the index of an external event and n the dispatch step of the state machine. We denote the t-th external event as e_t, assign each repair transaction a unique identifier i, and denote its observed outcome after maturity by o_i.

At step n, the persistent adaptive state of GovernMem is

$$Z_n = (M_n, \Theta_n, K_n, E_n). \tag{7}$$

The system also maintains transient computation state H_n and a pending-settlement queue Q_n. The full runtime state is therefore

$$X_n = (H_n, M_n, \Theta_n, K_n, E_n, Q_n). \tag{8}$$

Table 3: State components in GovernMem.

State	Meaning
H	Transient computation for the current step, which can be discarded after the task ends.
M	Persistent memory that the agent will read in future interactions.
Θ	Repair-routing experience that records which repairs are more reliable under different conditions.
K	The repair Skill library that contains the repair actions currently available to the system.
E	Settled historical evidence that is allowed to be retained.
Q	A pending queue of executed repairs whose consequences are not yet mature.
H_{evt}	The immutable history of original events.
G	Fixed governance rules that define who may change which state and what may be learned.

Online operation can change M_n, Θ_n, K_n, E_n, and Q_n, but it cannot change G or H_{evt}. At each step, the state machine processes one dispatch item:

$$\delta_n \in \{\text{External}(e_t), \text{Mature}(i, o_i)\}. \tag{9}$$

$\text{External}(e_t)$ means that a new external event has arrived, so the system checks whether persistent memory contains a failure and repairs it when needed. $\text{Mature}(i, o_i)$ means that the delayed outcome of repair transaction i has matured, at which point the system settles the outcome and checks whether it may enter the evidence set and subsequent learning. State updates have the unified form

$$X_{n+1} = D_G(X_n, \delta_n; H_{\text{evt}}). \tag{10}$$

Here, D_G is a state-transition function constrained by G, and it includes failure diagnosis, action filtering, parameterized routing, shadow execution, state verification, outcome settlement, and experience updates.

GovernMem has three connected components. The governed state machine processes External and Mature dispatch items and organizes each repair as a sequence of state transitions. The parameterized router selects a repair action from the legal action set and adjusts later choices using settled outcomes. Failure Memory, Pattern, and Skill form a hierarchical experience structure that turns specific failures into reusable repair capabilities. Sections 3.2–3.4 describe these components.

A.2 Governed Memory State Machine

GovernMem introduces five types of contracts and closes the repair loop through two dispatch paths, with each contract carrying the state of a successive stage. When an External event arrives, the system encodes the failure observation as a diagnostic Contract, filters the legal actions, and records the selected action. The action then undergoes shadow execution and state admission. Whether the system commits or rolls back, it writes the execution record to Q_n. Once the outcome matures, a Mature event triggers settlement, after which evidence admission and learning attribution update the persistent state. The complete process is

$$\begin{aligned}
\text{External}(e_t) &\to \text{Diagnostic Contract} \to \text{OperatorSpec} \\
&\to A_t \to \text{RouterDecision} \\
&\to \text{shadow execution and state admission} \to \text{PendingRepairOutcome} \\
&\to \text{Mature}(i, o_i) \to \text{RepairReceipt} \\
&\to \text{evidence admission and learning attribution} \to \text{next External event.}
\end{aligned} \tag{11}$$

The five contracts define the failure, constrain operation rights, fix the executed branch, preserve the execution record, and return the delayed outcome. Because each contract takes the output of the previous stage as input, action selection, state modification, and experience updates remain consistent with the state and evidence that produced them.

(1) Diagnostic Contract. The structural checker encodes a failure diagnosis as a Contract that binds the event identifier, observation identifier, event index, current state root, source-manifest root, failure mechanism, compatible repair category, mechanism-specific evidence, and provenance. For a process fault, it records the corresponding process subtype; for state drift, it records an ordered supersession relation; and for adversarial poisoning, it records anomaly evidence and suspect state identifiers. The Contract uses a closed schema and contains no reference answer, dataset label, or future outcome. Its purpose is to turn a failure observation into a reproducible constraint without gold information, which the governance layer uses to determine the repair types allowed for the current event. It does not directly select a specific action.

(2) Action Contract. The governance layer combines the diagnostic Contract, the current state, and K_n to construct the legal action set A_t. Each action is constrained by an OperatorSpec, which declares applicable failures, execution preconditions, read sets, write sets, mutation scope, invariants, safety constraints, and a rollback operation. It defines both what a particular Skill revision may read and modify and which conditions must hold after execution. An action cannot enter A_t if it violates the diagnostic Contract, fails a precondition, lacks validation or registration, or is quarantined. Therefore,

$$a \notin A_t \quad \Longrightarrow \quad P(a_t = a) = 0. \tag{12}$$

Action gating only checks whether an action is legal; it neither predicts the effect of the action nor attributes an outcome. After this check, the governance layer sends A_t and its OperatorSpecs to the router, which can select only within this boundary.

(3) Selection Contract. The router selects an action from A_t and records the decision as a RouterDecision. This record binds the candidate set, legal mask, routing-state hash, base scores, routed scores, selected action, exact Skill revision, and exploration address used for selection. It supports replay of the action choice and prevents a delayed outcome from being assigned to another action or version.

GovernMem does not run a complete rollout for every candidate action at runtime. After the router selects $a_t \in A_t$, the system creates a copy-on-write shadow state from M_n and executes the selected state transition:

$$\widetilde{M} = T_{a_t}(M_n). \tag{13}$$

After shadow execution, state admission checks root consistency, the invariants in OperatorSpec, safety constraints, mutation scope, and failure removal, while a state commit must also pass a CAS check. If another transaction changes the live state root during shadow execution, the current transaction cannot overwrite the updated live state. Let $C_i \in \{0, 1\}$ denote the complete admission result, including the commit-time check. For a dispatch transition starting from M_n,

$$M_{n+1} = \begin{cases} \widetilde{M}, & C_i = 1, \\ M_n, & C_i = 0. \end{cases} \tag{14}$$

If admission succeeds, the system commits the candidate state. Otherwise, it discards the changes from the current transaction, so the transaction remains at the before root. If the failure is caused by a concurrent conflict, the system does not undo a state committed by another transaction. State admission only decides whether a candidate modification may enter M_n; it does not decide whether the repair remains effective in later interactions.

(4) Pending-Outcome Contract. After state admission, both commit and rollback create a PendingRepairOutcome, which is bound to the diagnostic Contract and RouterDecision and written to Q_n. It includes the transaction identifier, selected action, exact Skill revision, before root, shadow root, after root, verification result, and earliest maturity time. PendingRepairOutcome preserves the selection, execution, and admission information until the consequences become observable, thereby

connecting the External path to the later Mature path. For the same transaction,

$$\text{commit} \Longrightarrow \text{after_root} = \text{shadow_root}, \tag{15}$$

$$\text{rollback} \Longrightarrow \text{after_root} = \text{before_root}. \tag{16}$$

The router does not learn directly from the immediate verification result because recurrence of the same failure and downstream effects of the repair may become clear only after later events. While the outcome is maturing, the system may process other events or restart, but PendingRepairOutcome still binds the delayed result to the original action, Skill revision, and state root.

(5) Result Contract. When $\text{Mature}(i, o_i)$ arrives, the scheduler retrieves transaction i from Q_n and creates a RepairReceipt. The receipt binds the original decision, executed Skill revision, success probe, observation time, state roots, commit or rollback status, invariant and safety results, locality cost, recurrence, and provenance. It records what was actually executed and the observable consequences that followed, rather than the outcome expected by the router during selection.

Settlement checks whether the outcome has reached maturity and whether it matches the corresponding PendingRepairOutcome. An outcome that is still immature at the end of evaluation remains pending and counts as neither success nor failure. Outcomes that mature at the same time are processed by their original selection time and transaction identifier, and each transaction can be settled only once to prevent repeated delivery from causing repeated learning.

A settled RepairReceipt must pass evidence admission before it can enter E_n. This check covers provenance, transaction binding, outcome maturity, and the past-only constraint, which requires the original decision to use only information available at decision time. Learning attribution then checks whether the admitted evidence maps correctly to the executed action and its Skill revision. Only evidence that passes attribution can update Θ_n or K_n, while right-censored outcomes update neither state.

The five contracts complete a repair loop across events. The diagnostic Contract defines the failure constraint, OperatorSpec narrows the available capabilities to the legal action set A_t, RouterDecision fixes the executed action and its version, PendingRepairOutcome preserves the immature execution record, and RepairReceipt sends the delayed outcome to evidence admission and learning attribution. State admission, evidence admission, and learning attribution control updates to M_n, E_n, and Θ_n or K_n, respectively. The next External event reads the updated states, and changes to Pattern and Skill lifecycles also take effect from that event. The previous repair can therefore affect the next action choice, but it cannot retroactively change the candidate set used when its own outcome was produced.

A.3 Parameterized Routing and Continual Learning

GovernMem uses a hierarchical residual router to reuse historical repair experience. The router neither generates Skills nor directly modifies MemoryState; it only selects a state transition from the legal action set A_t provided by the governance layer. By combining the current judgment of a frozen backbone with accumulated repair outcomes, it adjusts the relative preference among legal actions.We refer to this complete routing method as the GovernRouter and label it *GovernRouter* in the experimental tables and figures.

The frozen backbone returns a closed JSON object containing a selected legal Skill revision and a score for every legal candidate. Scores must be finite numbers in $[-1, 1]$ and are used directly, without a fitted calibration transform. Invalid outputs are rejected. The prompt contains the decision summary, structural telemetry, a bounded event sample, the memory-state summary, and typed candidate programs and contracts. These scores are model judgments on the utility scale, not calibrated probabilities. The selected action's original score is retained for the delayed residual update.

Let m_t be the number of current legal actions and d_t the routing-feature dimension. We denote the feature vector by $\boldsymbol{\phi}_t \in \mathbb{R}^{d_t}$, the base scores from the frozen backbone by $\mathbf{b}_t \in \mathbb{R}^{m_t}$, and the exploration perturbation by $\mathbf{z}_{n,t} \in \mathbb{R}^{m_t}$. The parameter matrix $W_{n,A_t} \in \mathbb{R}^{d_t \times m_t}$ is a finite projection of persistent routing state Θ_n onto the current features and legal actions. The action-score vector is

$$\mathbf{q}_{n,t} = \mathbf{b}_t + W_{n,A_t}^{\mathsf{T}} \boldsymbol{\phi}_t + \mathbf{z}_{n,t}. \tag{17}$$

The router selects the highest-scoring action in the legal action set:

$$a_t = \arg\max_{a \in A_t} q_{n,t}(a). \tag{18}$$

Here, $\mathbf{q}_{n,t} \in \mathbb{R}^{m_t}$ is a score vector for the m_t legal actions rather than a parameter matrix, and $q_{n,t}(a)$ is the scalar score of action a. The learned parameters correspond to W_{n,A_t}. In the implementation, Θ_n is stored as a sparse coordinate table, so the system only projects the features and actions involved in the current event into W_{n,A_t} instead of rebuilding a complete dense matrix for every event.

Routing experience has global, pattern, and local levels. The global level records the overall outcome of a Skill across contexts. The pattern level records the remaining effect of that Skill under a specific failure Pattern relative to the global level. The local level records additional local differences from observable features. Let $\rho_{t,p}$ be the responsibility of the current failure for Pattern p, and let $x_{t,j}$ be observable feature j. The coordinate weights are 1 at the global level, $\rho_{t,p}$ at the pattern level, and $\rho_{t,p}x_{t,j}/\|x_t\|_1$ at the local level. The system creates no local coordinate when $\|x_t\|_1 = 0$. In the reported routing experiments, the numeric local feature is the event index, $x_t = (t)$, and the diagnosed structural class selects one of three frozen Patterns with responsibility one. Thus the local weight is one for $t > 0$ and no local coordinate is created at $t = 0$; these experiments do not evaluate a richer learned contextual representation.

Each sparse coordinate k maintains precision $\Lambda_{k,n}$ and natural parameter $\eta_{k,n}$. Its experience mean and effective support are

$$\mu_{k,n} = \frac{\eta_{k,n}}{\Lambda_{k,n}}, \qquad s_{k,n} = \Lambda_{k,n} - 1. \tag{19}$$

Let $I_t(a)$ be the coordinate set activated by action a on event t. Let $w_{t,k}$ be the coordinate weight, and let $\tau_{\text{level}(k)}$ be the support threshold at its level. The residual correction from historical experience is

$$R_{n,t}(a) = \sum_{k \in I_t(a)} \mathbf{1}\left[s_{k,n} \geq \tau_{\text{level}(k)}\right] w_{t,k}\mu_{k,n}. \tag{20}$$

Support gating limits the effect of infrequent outcomes on routing. The global level first provides a coarse overall estimate, while Pattern and local terms join scoring only after enough evidence accumulates. If no coordinate reaches its support threshold and exploration is not active, the router uses the original backbone choice, so cold-start behavior remains consistent with the frozen backbone. Appendix C reports the support thresholds and exploration settings used in the experiments.

Exploration starts only after at least two legal Skills have enough global support. The perturbation is generated from the seed, event index, coordinate key, and router version. When the state, event order, candidate set, and router version are unchanged, the system produces the same perturbation and can replay the corresponding action choice; changing any address field also changes the perturbation.

Online updates follow a selected-only constraint: routing state is updated only from actions that were selected, executed, and associated with mature, attributable RepairReceipts. GovernMem evaluates these outcomes using a bounded, cost-sensitive repair utility $u_i^{\text{delay}} \in [-1, 1]$. A repair receives a utility of -1 if it is not committed, is rolled back, or fails the safety, invariant, or mutation-scope checks. Otherwise, utility subtracts penalties for mutation scope, collateral changes, expected execution cost, feedback delay, and recurrence from a base value of one. Appendix D.2 specifies the component definitions, normalization, and weights. The current routing experiments instantiate this utility using development-structural feedback rather than end-task answer accuracy.

We distinguish the reported repair utility from the residual learning target. Let $\hat{u}_i^{\text{pre}}$ denote the stored backbone score of the selected action recorded before execution in repair transaction i, and let u_i^{delay} denote its observed utility after maturity. Here, $\hat{u}_i^{\text{pre}}$ is taken from the selected action's backbone score, rather than a measured utility of the pre-repair state. For an invalid transaction, rollback, or subsequent degradation, we set $u_i^{\text{effective}} = -1$; otherwise, we set $u_i^{\text{effective}} = u_i^{\text{delay}}$. The learning target is

$$r_i = \text{clip}\left(u_i^{\text{effective}} - \hat{u}_i^{\text{pre}}, -1, 1\right). \tag{21}$$

Thus, r_i measures the residual between observed repair utility and its pre-execution prediction, rather than a direct utility difference between the repaired and unrepaired states.

The global level learns r_i directly to estimate the overall prediction residual across contexts. The pattern level learns the remaining error after subtracting the pre-update global mean, and the local level further subtracts the corresponding pre-update pattern mean. Let $y_{i,k}$ denote the level-specific learning target for coordinate k. The coordinates of the selected action are updated by

$$\Lambda_{k,n+1} = \Lambda_{k,n} + w_{i,k}^2, \qquad \eta_{k,n+1} = \eta_{k,n} + w_{i,k}y_{i,k}. \tag{22}$$

The coordinates of unselected actions remain unchanged. This hierarchical update supports cross-context reuse at the global level, correction for failure structure at the pattern level, and fine-grained contextual adjustment at the local level when sufficient evidence is available. Each level learns the residual left unexplained by the preceding levels, reducing repeated incorporation of the full feedback across levels.

A.4 Contract and Router Diagrams

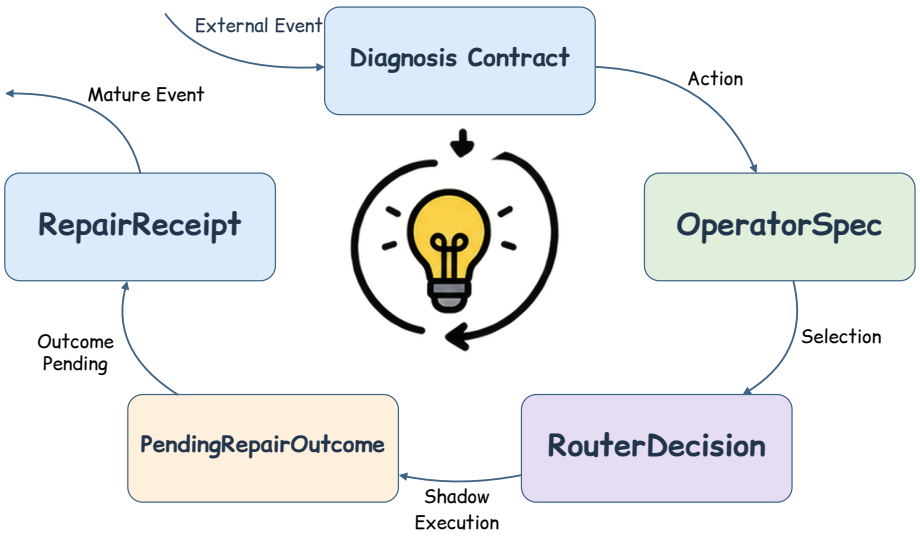


Figure 5: The five-stage contract in GovernMem. Diagnosis and action gating constrain repair selection and execution. Both commit and rollback produce a pending outcome, which is settled after maturity. Admitted and attributable evidence then informs subsequent repairs.

B Governed Execution and Skill Lifecycles

The following specifications connect diagnosis, verified state transitions, mature feedback, and revision management.

B.1 Operational State Machines

GovernMem implements the transition function in Section 3.1 through four nested state machines. The first is the macro scheduler, which chooses whether the next dispatch item is a new external event or a matured result from an earlier repair. The second is the foreground repair transaction, which diagnoses an external event and either commits or rolls back one selected state transition. The third is the background evidence machine, which settles delayed consequences and determines whether they can update persistent experience. The fourth governs Pattern and Skill lifecycles. These machines share identifiers and state roots, but their mutation rights remain separate.

The foreground path is

$$\begin{aligned} \text{DISPATCH} &\to \text{OBSERVE} \to \text{INSPECT} \\ &\to \text{LEGALIZE} \to \text{ROUTE} \to \text{SHADOW} \\ &\to \text{VERIFY} \to \{\text{COMMIT}, \text{ROLLBACK}\} \\ &\to \text{ENQUEUE} \to \text{DISPATCH}. \end{aligned} \tag{23}$$

OBSERVE reads only the deployment-visible DecisionView and current MemoryState. INSPECT derives a structural diagnosis without changing memory. LEGALIZE removes incompatible or

unavailable Skill revisions. ROUTE ranks the remaining revisions but cannot modify live state. SHADOW applies only the selected transition to an isolated copy. VERIFY evaluates the resulting candidate, after which COMMIT is the sole foreground state allowed to replace persistent memory. ROLLBACK discards the candidate mutation but retains the attempted transition as an auditable record. ENQUEUE places that record in Q_n without treating it as mature evidence.

The background path is

$$\begin{aligned} \mathrm{DISPATCH} \to \mathrm{SETTLE} \to \mathrm{QUALIFY} \to \mathrm{ATTRIBUTE} \\ \to \{\mathrm{LEARN}, \mathrm{NO_UPDATE}\} \to \mathrm{DISPATCH}. \end{aligned} \tag{24}$$

SETTLE checks that an observation has matured and is bound to a pending transaction. QUALIFY decides whether the result may enter persistent evidence, while ATTRIBUTE decides whether the result belongs to the executed action and exact Skill revision. LEARN may update Θ_n or K_n only after both checks succeed. In the current implementation, several QUALIFY and ATTRIBUTE predicates are evaluated together by typed receipt eligibility rather than materialized as two independent general-purpose objects. This implementation choice does not merge their permissions: an admitted observation may remain in E_n without updating adaptive state when action attribution is unavailable.

The lifecycle machine contains two coordinated state spaces. Skill content progresses through

$$\mathrm{proposed} \to \mathrm{sandboxed} \to \mathrm{shadow_validated} \to \mathrm{calibrated} \to \mathrm{stable}, \tag{25}$$

whereas runtime service status progresses from probationary to active, quarantined, superseded, or retired. Content state records how far a program has been validated; service state determines whether that revision may enter a legal action set. A content-stable revision can therefore remain unavailable while it is probationary or quarantined.

Stage	Authorized operation	Prohibited operation
OBSERVE	Read DecisionView and bound MemoryState	Read evaluator-only fields or future results
INSPECT	Produce a structural Contract	Mutate persistent memory
LEGALIZE	Remove incompatible candidates	Authorize an action outside the registry
ROUTE	Rank legal Skill revisions	Modify live or shadow state
SHADOW	Modify an isolated copy	Modify live state
VERIFY	Evaluate roots, invariants, safety, and locality	Commit without a successful gate
COMMIT	Replace live memory after CAS	Rewrite immutable source history
ROLLBACK	Restore the transaction's before-root branch	Erase the attempted transition
ENQUEUE	Create a pending result	Use an immature result for learning
SETTLE	Consume a mature bound observation once	Reopen state admission
LEARN	Update eligible routing or lifecycle state	Update an unselected action

Table 4: Stage-local mutation rights in the GovernMem runtime.

B.2 Contract Sequence and Structural Diagnosis

The online transaction is connected by five typed contracts. Each contract binds the output of one stage to the input of the next, so later evidence cannot be reassigned to another diagnosis, action, or version. Table 5 summarizes their bound information and operational roles after the diagnosis procedure is specified.

The Diagnostic Contract uses a closed schema. Process faults carry a process subtype; state drift carries an ordered pair that identifies the superseding and superseded memories; adversarial poison carries anomaly evidence and suspect identifiers. Fields associated with one mechanism are invalid for the other mechanisms. The contract also rejects provenance whose keys or nested values expose evaluator-only markers. This boundary prevents dataset labels, oracle actions, reference repairs, and future outcomes from entering diagnosis or routing.

The structural checker treats immutable source events as a write-ahead history and MemoryState as its materialized projection. It first checks that the DecisionView root matches MemoryState. It then compares the visible event history with the immutable source and audit logs. The check also detects

events that were silently inserted, removed, or reordered. A binding failure terminates inspection rather than producing a repair suggestion.

The checker next derives the expected active members. Valid supersession edges remove an older source event from the active set and add the source-derived replacement. An edge is invalid if either endpoint is absent, if replacements are duplicated, or if the new event is not source-derived. The checker then looks for process faults. These faults include inconsistent projection membership or order, incorrect projection indices, mismatched permission scopes, and stale cache identifiers.

State drift requires an observable lineage relation: an active source-derived audit event must name the older source event it supersedes, and both endpoints must be present and consistently ordered. Adversarial poison requires an untrusted audit event that remains in the active projection and is absent from the quarantine set. Let I_{proc}, I_{drift}, and I_{poison} indicate whether each signal family is present. The conservative classification rule is

$$c = I_{\mathrm{proc}} + I_{\mathrm{drift}} + I_{\mathrm{poison}}, \qquad \mathrm{class}(c) = \begin{cases} \text{clean}, & c = 0, \\ \text{the unique detected class}, & c = 1, \\ \text{unknown}, & c > 1. \end{cases} \tag{26}$$

Unknown cases abstain. GovernMem does not choose the most likely class because the costs are asymmetric: treating poison as an ordinary projection fault, for example, can preserve untrusted content during reconstruction. The checker is therefore a structural consistency mechanism rather than a general semantic correctness model. A semantic update is recognized as state drift only when it carries verifiable lineage.

Contract	Bound information	Role
Diagnostic Contract	Event and incident identifiers, event index, state and source roots, fault mechanism, repair family, mechanism-specific signals, provenance	Converts an observation into a closed, gold-free legality constraint
OperatorSpec	Compatible faults, preconditions, read and write sets, mutation bound, invariants, safety checks, rollback program	Defines what one Skill revision is permitted to read and change
RouterDecision	Legal candidates, router snapshot, backbone and routed scores, selected action and exact revision, exploration address	Makes the actual choice reproducible and version-bound
PendingRepairOutcome	Selection identity, exact revision, before, shadow, and branch-after roots, gate result, commit result, maturity index	Carries foreground execution facts across the delay interval
RepairReceipt	Original selection, exact revision, probe, observation index, roots, commit or rollback status, safety, locality, recurrence, provenance	Represents the mature consequence used by evidence admission and learning attribution

Table 5: Contracts in their runtime order.

B.3 Online Shadow Execution and Root-Bound Admission

The online runtime does not perform a complete answer rollout for every candidate. After structural diagnosis and legalization, the backbone and residual router choose one Skill revision. Only that revision executes in a copy-on-write shadow state. The online sequence is therefore

$$\text{diagnose} \rightarrow \text{legalize} \rightarrow \text{select one} \rightarrow \text{shadow} \rightarrow \text{verify} \rightarrow \{\text{commit}, \text{rollback}\}. \tag{27}$$

A full shadow matrix remains available to the isolated evaluator, which may execute every legal action to compute an oracle, regret, or paired routing metric. Outcomes of unselected actions cannot enter the online router. This separation preserves evaluation coverage without converting evaluator-only counterfactuals into deployment feedback.

For a selected action a_t, shadow execution produces $\widetilde{M} = T_{a_t}(M_n)$. The state-admission gate first checks whether a required mutation changed the shadow. It also checks that immutable logs remain

unchanged and that no active untrusted event survives outside quarantine. The mutation cost must remain within the OperatorSpec bound, and structural inspection of the shadow must return clean. Even a locally valid shadow cannot overwrite a newer live state. The memory store commits by compare-and-swap using the root observed before shadow execution:

$$\mathrm{CAS}(\mathrm{before_root}, \widetilde{M}) = \begin{cases} \mathrm{commit}, & \mathrm{live_root} = \mathrm{before_root}, \\ \mathrm{conflict}, & \mathrm{live_root} \neq \mathrm{before_root}. \end{cases} \tag{28}$$

A conflict leaves the newer live state intact; it does not restore a stale before root over another transaction.

Each transaction binds three roots. The before root identifies the live state from which the selected repair started. The shadow root identifies the exact candidate that passed or failed verification. The after root identifies the state accepted by that transaction branch. A committed RepairReceipt is valid only if

$$\begin{aligned} \mathrm{committed} = \mathrm{true}, \qquad & \mathrm{rolled_back} = \mathrm{false}, \\ \mathrm{resolved} = \mathrm{true}, \qquad & \mathrm{invariants_passed} = \mathrm{true}, \\ \mathrm{safety_violation} = \mathrm{false}, \qquad & \mathrm{after_root} = \mathrm{shadow_root}. \end{aligned} \tag{29}$$

A noncommitted receipt must instead satisfy

$$\mathrm{rolled_back} = \mathrm{true}, \qquad \mathrm{after_root} = \mathrm{before_root}. \tag{30}$$

These equalities prevent the runtime from validating one state and recording another. Recurrence after commit is defined only for a committed receipt. Rollback therefore stops the candidate mutation but does not delete the fact that the action was attempted.

B.4 Delayed Settlement and Exactly-Once Learning

Immediate verification establishes only that the candidate is structurally valid at commit time. Recurrence, delayed integrity failures, or downstream safety effects can appear after later events. GovernMem consequently stores a PendingRepairOutcome containing the selection time, earliest maturity time, selected revision, registered probe, roots, gate results, and commit result. The record remains in Q_n and cannot update the router before maturity.

Before routing external event t, the runtime settles every received receipt whose observation index is at most t and whose selection index is strictly less than t. Ready receipts are processed in the deterministic order

$$(\mathrm{selected_at_event}, \mathrm{selection_id}). \tag{31}$$

This ordering permits evidence from earlier events to affect event t, while preventing event t from learning from its own consequence. Settlement verifies that the pending identifier exists, the maturity index has been reached, the observation is not duplicated or censored, and the selection, Skill revision, probe, and roots match the original transaction.

Exactly-once learning uses three checks. Finalization removes a consumed pending record, duplicate receipt identifiers must have identical content, and an identifier already present in the settled set cannot update the router again. A crash retry can therefore replay the ledger without duplicating an update. Records that have not matured by the evaluation horizon are marked right-censored and reported separately. They receive neither a positive nor a negative reward because the relevant consequence remains unknown.

B.5 Capability Formation and Niche Governance

Failure Memory, Pattern, and Skill form an experience-driven hierarchy with distinct semantics. A Failure Memory record stores one concrete failure instance, including its case, audit-only family, observable features, state or memory hash, context hash, and provenance hash. It establishes that a failure occurred and supplies provenance for later proposals; it is neither a fault label exposed to routing nor an executable repair strategy.

A Pattern summarizes recurring failure structure through a matching predicate, feature signature, parent revisions, derivation kind, and lifecycle state. Failures may bind softly to multiple Patterns

Foreground and delayed outcome	Router consequence	Lifecycle consequence
Rollback	Negative selected-action evidence after settlement	Retained for audit; not a promotion success
Commit followed by recurrence	Negative selected-action evidence	Counts against continued applicability
Commit followed by safety, provenance, or integrity violation	Negative selected-action evidence	May quarantine the exact revision from the next event
Commit with no recurrence or violation	Positive evidence reduced by locality cost	May support probationary promotion
No mature observation at the evaluation horizon	Right-censored; no update	Neither success nor failure

Table 6: Learning and lifecycle effects of delayed outcomes.

through responsibilities $\rho_{f,p}$ satisfying

$$\rho_{f,p} \in [0,1], \qquad \sum_{p} \rho_{f,p} = 1. \tag{32}$$

Pattern formation depends on repeated structure rather than repair success. A successful repair can add evidence to a Pattern, but it does not itself become a Pattern. Pattern revisions progress through candidate, recurring, validated, stable, and deprecated states; split and merge create derived revisions with explicit parents. Split is appropriate when one Pattern persistently requires different Skills in different subcontexts. Merge is proposed when two Patterns exhibit sufficiently similar features, Skill rankings, and feedback behavior.

A Skill is a versioned executable repair program. Its revision binds the program, parameter schema, preconditions, postconditions, success probe, mutation budget, rollback program, parent revisions, producing failure, and content hash. Pattern–Skill bindings represent applicability rather than identity: one Pattern may have competing Skills, and one Skill may apply to several Patterns. Program, parameter, or scope changes create a new revision instead of overwriting the old one, which preserves the meaning of historical RouterDecisions and RepairReceipts.

The three-success rule governs service promotion, not Pattern creation. A probationary Skill needs successful repairs on at least three distinct later failures. These failures must span at least two audit families and must exclude the failure that produced the Skill. No supporting result may be rolled back or derived from gold information. The locked anchor must also show no regression. Formally, for eligible support set S_a of revision a,

$$|S_a| \geq 3, \qquad |\{\mathrm{family}(f) : f \in S_a\}| \geq 2, \qquad f_{\mathrm{origin}} \notin S_a. \tag{33}$$

This rule prevents a candidate from proving itself on the same failure that generated it.

A Niche summarizes the supply of recurring failures and the service provided by Skills over a fixed window. For observations assigned to Pattern p, resource mass is the sum of responsibilities, while unresolved mass sums responsibilities for unresolved observations. Skill occupancy sums the responsibility of selected Skills. Selection entropy H_p yields an effective species count $\exp(H_p)$. With dominant occupancy share d_p, resource mass R_p, unresolved mass U_p, and effective species count N_p, split pressure is

$$\mathrm{split_pressure}_p = \begin{cases} 0, & R_p = 0, \\ \min\left(1, \dfrac{U_p}{R_p} \max(0, N_p - 1)\right), & R_p > 0. \end{cases} \tag{34}$$

The implemented state rule is evaluated in order. Zero resource gives latent, or extinct if the previous state was collapsing or extinct. Resource below 2 gives emerging. Split pressure of at least 0.35 gives branching. An unresolved fraction above 0.75 gives collapsing when the previous state was stable or contested. At least two occupied Skills with dominant share below 0.70 gives contested. Resource of at least 5 with dominant share at least 0.70 gives stable. Other nonzero cases are occupied.

Niche observation does not grant lifecycle authority. A branching state can produce a Pattern-split proposal, and persistent discovery pressure can produce a new-Pattern, new-Skill, or Skill-revision proposal. Safety, provenance, or integrity violations in a mature receipt can trigger quarantine of the exact revision. Split, merge, revision, and retirement remain governance operations. Accepted service-state changes take effect after the current event, so event t reads only lifecycle state already effective before t. Quarantine preserves the program and evidence while removing the revision from subsequent legal sets; supersession links a replacement to its parent; retirement is a recorded terminal withdrawal from service.

All capability and ecology changes are written to a hash-linked append-only ledger. Failure observations, Pattern and Skill revisions, bindings, selections, receipts, posterior snapshots, Niche snapshots, and lifecycle transitions carry ordered event identifiers and content hashes. Recovery replays the ledger and checks the materialized Skill state, settled and right-censored sets, router snapshot, and trusted chain head. The same records therefore support execution, governance, audit, and deterministic recovery.

B.6 Extended Ecological Governance

At the individual-capability level, Failure Memory records concrete faults and provenance, Patterns describe recurring fault structures, and Skills implement executable repairs. Patterns arise from repeated structure; successful repairs add evidence rather than automatically creating Patterns. A probationary Skill becomes eligible for service promotion after successful repairs on at least three distinct later failures spanning at least two audit families, excluding its originating failure and subject to evidence, governance, and locked-anchor checks.

Patterns and Skills use revision management. A Pattern describes a recurring failure structure and its applicability, whereas a Skill implements a verified repair strategy as an executable, verifiable, and rollback-capable state transition. One Pattern may correspond to several competing Skills, and one Skill may apply to several Patterns. When program content, parameter constraints, or applicability changes, the system creates a new revision and preserves the relation between old and new versions. Skill revisions are managed according to verification results and later-case performance, and only registered versions that pass stability verification can enter the legal action set. If a safety, provenance, or integrity anomaly occurs, the governance layer can remove the version from service starting with the next event while retaining its program and historical evidence.

At the capability-set level, a Niche describes the coverage, competition, and differentiation of multiple Skills based on failure resources, Pattern responsibilities, execution constraints, and Skill performance within a time window. The system accordingly maintains fact, structure, capability, and transition state, which preserve failure and execution records, Patterns, versioned Skills, and their performance on later cases. Updating Θ_n changes only action-selection preferences and is therefore routing adaptation, whereas creating, revising, splitting, merging, quarantining, or retiring a Pattern or Skill changes K_n and therefore evolves the capability set.

A Niche is determined dynamically from runtime records in the window, mainly using failure occurrence and recurrence, the unresolved rate, Skill occupancy, and performance differences among Skills. Its lifecycle is

$$\begin{aligned} \text{latent} &\to \text{emerging} \to \text{occupied} \to \text{stable}, \\ \text{occupied} &\to \text{contested} \to \text{branching}, \\ \{\text{stable}, \text{contested}\} &\to \text{collapsing} \to \text{extinct}. \end{aligned}$$

These states describe how failure resources appear and grow, how Skills begin to serve them and reach stable coverage, and how later competition, differentiation, or decline occurs. A Niche only describes the relation between the current capability set and failure resources; it does not directly execute Pattern splitting, Skill creation, quarantine, or retirement. The governance layer decides these operations from registered evidence, and related changes take effect from the next event. The current event therefore cannot use its own result to change its candidate set.

Version records preserve program lineage and supporting evidence. Safety or integrity failures can withdraw a revision without deleting its history. An append-only ledger records repairs, routing updates, and lifecycle changes for audit and recovery. Detailed lifecycle and recovery rules appear in Appendix B.5.

C SPARSE ROUTING STATE AND REPLAYABLE EXPLORATION

Router names. GovernRouter denotes the complete router described in Section 3.3. GovernRouter-P denotes the internal hierarchical posterior-routing control, where P stands for posterior. Both use backbone scores and hierarchical historical state, but differ in their support-gating and exploration rules.

Initialization and ablations. For GovernRouter, a new residual coordinate starts at precision 1 and natural parameter 0. Updates are additive with no decay in this router. Coordinates are keyed by exact Skill revision; a new revision without imported coordinates receives the same empty prior. When adaptive scoring is active, ties are resolved lexicographically by Skill revision ID. Coordinate perturbations are multiplied by their coordinate weights before being added to the action score. The Global, Global+Pattern, and Global+Pattern+Local profiles enable the named residual levels with support gating and no exploration. Full enables all levels plus supported exploration. Full without support gating uses zero residual and exploration-support thresholds, while retaining all three levels. Thus Full versus Global+Pattern+Local also tests exploration.

GovernRouter-P control. The hierarchical control adds weighted Gaussian posterior draws to the backbone scores. Pattern and local contributions require support of at least 2 and 3, respectively. Global contributions have no minimum-support gate. The exploration scale is 0.08, with each coordinate's standard deviation scaled by the inverse square root of its precision. Unlike GovernRouter, this control does not require two globally supported legal Skills before applying exploration.

Experimental features. The routing executor hard-assigns the diagnosed process fault, state drift, or poison class to its corresponding frozen Pattern. Its only numeric local feature is the event index. For positive event indices, normalization makes the local feature weight one, so the reported local coordinates are an additional Pattern-conditioned residual stage rather than a rich contextual model.

Statistical controls. Global Thompson initializes each Skill with $\mathrm{Beta}(1,1)$ and updates (α,β) by $(\max(0,u),\max(0,-u))$ after a mature selected-action outcome u. Niche Thompson applies the same rule to a niche–Skill key. These are signed-utility pseudo-count updates. The contextual control uses $x=(1,|\mathrm{projection}|,|\mathrm{auditlog}|,t)$, initializes diagonal precision $d_j=1$ and $v_j=0$ per Skill, and scores $\sum_j (v_j/d_j)x_j + 0.35\sum_j |x_j|/\sqrt{d_j}$. Selected feedback updates $d_j \leftarrow d_j + x_j^2$ and $v_j \leftarrow v_j + u x_j$. This is the implemented diagonal optimistic control, not full-matrix LinUCB.

Source-state access. In the 24 Matched all-arms reports, GovernRouter-P and GovernRouter import source router snapshots. Thompson and contextual controls start with empty adaptive statistics. The arm labelled Best-global static has an empty calibration prior in all 24 reports and therefore chooses the lexicographically first legal Skill revision under tied zero scores. Its reported result is a fixed-order legal selector, not a development-optimized global policy. These comparisons do not isolate algorithm choice under equal source-posterior initialization.

The vector score in Section 3.3 is evaluated over the current legal set. For a single action, $q_{n,t}(a)$ is a scalar; $\mathbf{q}_{n,t}$ is the vector of scores for the m_t legal actions. The persistent object is not this score vector or an event-by-action matrix. Instead, Θ_n is a sparse map whose coordinates are

$$k_{\mathrm{global}} = (\mathrm{global}, a), \tag{35}$$

$$k_{\mathrm{pattern}} = (\mathrm{pattern}, p, a), \tag{36}$$

$$k_{\mathrm{local}} = (\mathrm{local}, p, j, a). \tag{37}$$

Each coordinate stores precision $\Lambda_{k,n}$ and natural parameter $\eta_{k,n}$, yielding mean $\mu_{k,n}=\eta_{k,n}/\Lambda_{k,n}$ and effective support $s_{k,n}=\Lambda_{k,n}-1$. The corresponding event weights are 1, $\rho_{t,p}$, and $\rho_{t,p}x_{t,j}/\|x_t\|_1$. Local coordinates are omitted when $\|x_t\|_1=0$.

The registered support thresholds in the reported runs are

$$\tau_{\mathrm{global}}=2, \qquad \tau_{\mathrm{pattern}}=4, \qquad \tau_{\mathrm{local}}=8, \tag{38}$$

and exploration requires global support of at least $\tau_{\mathrm{exp}}=4$ for at least two legal Skills. When no residual coordinate is active and exploration is disabled, the selected action remains the frozen

backbone choice. This makes cold-start behavior an explicit fallback rather than an untrained adaptive policy.

Exploration uses deterministic pseudorandom addressing. For coordinate k at event t, the runtime computes

$$h_{t,k} = \mathrm{SHA256}(\mathrm{seed}, t, k, \text{router version}) \tag{39}$$

and draws

$$z_{t,k} = \mathrm{PRNG}(h_{t,k}).\,\mathrm{Normal}\left(0, \frac{\gamma^2}{\Lambda_{k,n}}\right), \qquad \gamma = 0.08. \tag{40}$$

The same seed, event index, coordinate key, and router version reproduce the same perturbation. Different events and Skills use different addresses, so the noise is not a constant bonus attached to one Skill. Reproducing the perturbation alone does not guarantee the same final action: the router snapshot, legal candidate set, Pattern responsibilities, and backbone scores must also match.

Online updates remain selected-only. For an admitted and attributable result, the global coordinate learns the bounded utility difference r_i. Pattern coordinates learn r_i after subtracting the pre-update global mean, and local coordinates further subtract the pre-update Pattern mean. Only coordinates activated by the selected Skill are updated:

$$\Lambda_{k,n+1} = \Lambda_{k,n} + w_{i,k}^2, \qquad \eta_{k,n+1} = \eta_{k,n} + w_{i,k} y_{i,k}. \tag{41}$$

No synthetic feedback is created for an unselected action, including actions evaluated in the offline full shadow matrix.

D Evaluation Protocol and Utility

This section defines the data, comparisons, and utility measurements used by the supporting results.

D.1 Extended Experimental Setup

D.1.1 Datasets and Evaluation Metrics

Existing public datasets usually do not provide paired clean and corrupted states, exact state roots, shadow-execution outcomes for all candidate actions, and safety, locality, and rollback constraints at the same time. We construct GovernMem-RepairStream by adding reproducible process_fault, state_drift, and poison interventions to semantic content from public sources. This produces repair-event streams that contain states, actions, and receipts.

The completed main experiments use HaluMem, MemFail, and MemTraceBench. We select 20 original interaction episodes from each source and compile them into 58, 29, and 49 repair cases, respectively, for 136 cases in total. These cases include clean controls and intervened samples, and each source has both a stationary order and an abrupt order. Cases repeated under different orders or random seeds do not count as new independent samples.

The data protocol defines D_{skill}, D_{router}, D_{cal}, and $D_{\mathrm{lifecycle}}$ for Skill discovery, routing-experience accumulation, threshold calibration, and lifecycle validation, respectively. T_{online} supports online evaluation with scoring before updating, while T_{anchor} is a hidden anchor that returns no outcomes to learning and T_{final} is reserved for final evaluation. The partitions follow the family and lineage identifiers in the manifest.

We separate experience reuse from evaluation on unseen failure families. The source and target sides of the development experience-reuse matrix contain the same cases, which allows the matrix to compare cross-model reuse of persistent state. A separate family-disjoint experiment evaluates frozen Skills on unseen failure families.

The metrics include utility, regret, mutation scope, and safety-related measures. The main table uses stream-macro utility, computed by first averaging utility within each experimental stream and then giving all streams equal weight. The mechanism-ablation table uses event-weighted utility over paired mature events. Significance analysis uses the family-macro paired effect: we compute the utility difference between two methods on the same event, average these differences within each failure family, and then give all families equal weight. A utility difference of 0.01 means one utility point, not a one-percentage-point increase in accuracy.

Confidence intervals are computed with a family-blocked paired bootstrap. Intervention variants, event-order variants, and random-seed variants of the same failure family remain in the same resampling block. All 95% confidence intervals in the paper are pointwise intervals.

Empirical regret in the mechanism ablations uses the best realized utility among all ablation configurations on the same event as its reference, rather than oracle regret over all legal actions. SafeRepairSuccess, FalseCommit, CleanFalseRepair, rollback recovery, and collateral mutation are reported as separate metrics because higher average utility cannot compensate for degradation in a safety metric. Routing experiments use development-structural feedback, whereas independent governance experiments use results from sealed-sidecar evaluation.

D.1.2 BASELINES

The routing experiments compare Random legal, Best-global static, Global Thompson, Niche Thompson, Contextual bandit, GovernRouter-P, and GovernRouter. Random legal selects randomly from the legal actions, while the arm labelled Best-global static uses a fixed ordering of legal Skill revisions in the reported runs: its calibration prior is empty. The other methods accumulate and use action feedback at different levels of conditioning.

GovernRouter-P is an internal hierarchical posterior-routing control. It combines the same backbone scores with posterior contributions at the global, Pattern, and local levels. Pattern and local contributions are activated by their support thresholds, and Gaussian perturbations are applied to active coordinates, including the global level. GovernRouter denotes the complete router described in Section 3.3, which uses explicit support-gated residual corrections, a backbone fallback at cold start, and exploration activated only after sufficient global support.

All methods use the same event order, legal candidate set, feedback-maturity rules, and budget constraints. Static methods do not update online, while adaptive methods receive only mature feedback from their own selected actions. Shadow-execution outcomes of unexecuted actions do not enter online learning. Source initialization differs: GovernRouter-P and GovernRouter import source router state in the Matched comparison, whereas the Thompson and contextual controls start with empty adaptive statistics. Appendix C specifies these initializations and the exact control algorithms.

The mechanism ablations include Frozen backbone, Global, Global+Pattern, Global+Pattern+Local, Full w/o support gate, and the full GovernRouter. We measure the contributions of Global, Pattern, and Local through paired differences after adding the corresponding residuals in sequence, and evaluate support gating by comparing the full configuration with the configuration that removes the gate. Each increment depends on its comparison configuration and therefore cannot be interpreted as an independent, additive contribution.

Future work will extend the evaluation to budget-aligned system-level comparisons with Mem0, MemSkill, and ERSkill.

D.1.3 EXPERIMENTAL SETUP

We use Qwen2.5-14B-Instruct as the source model, with Meta-Llama-3.1-8B-Instruct and Qwen3-14B as the target models. The source stage discovers candidate Skills and accumulates routing experience. The cross-model stage then reuses frozen Skills, legal-action rules, utility settings, and source routing state without updating the target-model weights. Model services run on NVIDIA A800 80GB GPUs, while structural replay and statistical analysis run on CPUs. Backbone predictions are cached per event and shared across routing arms within a run. Appendix C specifies routing configurations and Appendix D.2 gives the exact structural utility, normalization, weights, and a recorded example.

The GovernRouter-only matrix covers two models, three sources, two orders, three random seeds, and four experience conditions, for 144 runs. Matched all-arms uses two random seeds and contains 24 runs, while the mechanism ablations and family-disjoint experiments each contain 24 runs. The A-B-A experiment uses Qwen3 with two random seeds on HaluMem and two on MemTraceBench, for four runs in total. The governance experiment contains six Qwen3 runs and two Llama runs. These run counts are not independent statistical sample counts.

The experience conditions are Reset, Matched, Global, and Global-prefix. Reset clears adaptive routing state, Matched uses source experience that matches the target source and event order, and Global uses source routing state accumulated across sources. Global-prefix adapts Global on a target prefix and scores only the shared suffix. Here, Global is a posterior-initialization condition rather than the Global residual used in the mechanism ablations.

Each event completes its current decision and scoring before mature and attributable historical feedback updates later state. Feedback that is still immature at the end of evaluation is right-censored. All candidate actions use the same legality, state-root, invariant, safety, locality, and CAS checks, while governance ablations remove components according to their variant definitions. Offline shadow-execution outcomes are used for evaluation but not as online feedback for unselected actions.

D.2 Repair Utility and Evaluation Details

The observed utility and the residual learning target are distinct: reported routing utility scores the selected repair, whereas the residual additionally subtracts its stored backbone score.

Exact score. Let L_i be logical locality cost, B_i the declared locality bound, C_i collateral cost, E_i the catalog execution-cost score, D_i normalized feedback delay, and R_i the recurrence indicator. Noncommit, rollback, failed safety or invariants, and $L_i > B_i$ receive -1. Otherwise,

$$u_i^{\text{delay}} = \text{clip}(1 - 0.20\ell_i - 0.20C_i - 0.15E_i - 0.10D_i - 0.75R_i, -1, 1)\,, \tag{42}$$

where $\ell_i = \min(1, L_i/B_i)$ for $B_i > 0$ and $\ell_i = 0$ for $B_i = 0$. These are the fixed implementation weights used by the structural feedback provider.

Measurements. Locality is a logical repair-surface score, not a count of changed tokens. No changed surface gives zero; modification outside the declared write set exceeds the bound and fails admission. Rebuild, cascade, and quarantine-and-rebuild use their declared bound; targeted state supersession costs two and other targeted changes cost one. Collateral cost is the fraction of the six mutable surfaces changed outside the declared write set, capped at one. The six surfaces are projection order, projection index, scope projection, cache identifiers, supersession edges, and quarantine set. Write-set spillover already fails the locality gate rather than being compensated by a small collateral penalty. Execution cost $E_i \in [0, 1]$ is the fixed OperatorSpec cost, not measured tokens or wall-clock latency. For selection index t_i and observation index o_i, $D_i = \min(1, \max(0, o_i - t_i - 1)/10)$.

Structural recurrence proxy. In the development routing provider, $R_i = 1$ when the regime name contains "abrupt" and the selected strategy is not rebuild, cascade, or quarantine-and-rebuild; otherwise $R_i = 0$. This evaluator-side strategy-conditioned proxy is not measured downstream recurrence. It is not part of the router's decision-time numeric feature. The utility evaluates the declared structural benchmark and its cost preferences, rather than general semantic repair quality. The development provider uses structural validity as its commit surrogate; sealed-sidecar governance results separately evaluate commit behavior.

Catalog costs. Execution-cost scores are 0.10 for targeted process restore/order replay, 0.05 for index/scope/cache repair, 0.70 for process projection rebuild, 0.15 for targeted state supersession, 0.65 for state cascade, 0.15 for quarantine-only, and 0.60 for quarantine-and-rebuild.

Recorded example. In the Matched Llama HaluMem abrupt run with seed 20260827, the first GovernRouter repair selects targeted process restoration. Its locality is $L_i = 1$ with $B_i = 2$, collateral cost is zero, $E_i = 0.10$, and observation at event two after selection at event zero gives $D_i = 0.10$. The abrupt proxy gives $R_i = 1$. Its utility is $1 - 0.20(1/2) - 0.15(0.10) - 0.10(0.10) - 0.75 = 0.125$. The recorded backbone score is -1, so the clipped residual is 1. Positive residual feedback means improvement relative to the recorded prediction, not necessarily high absolute repair utility.

Scoring population. Routing summaries use mature repair receipts. Clean or unknown observations that abstain without a selected repair do not supply such receipts, and right-censored outcomes do not update the router. CleanFalseRepair is reported separately in governance evaluation.

E EXPERIENCE REUSE AND SUPPORTING MECHANISM EVIDENCE

The supporting results first examine experience initialization and the mechanisms responsible for routing gains, then characterize adaptation and the operational behavior of governed repair.

E.1 EXPERIENCE CONDITIONS AND ADDITIONAL ABLATION DETAILS

Table 7: Experience conditions. Global here names a pooled initialization condition; Global in the ablation table names a residual level.

Condition	Routing experience
Reset	Starts with cleared routing state.
Matched	Imports source experience from the matching source and event order.
Global	Imports pooled source experience.
Global-prefix	Starts from pooled experience, additionally adapts on a target prefix, and is evaluated on the common suffix.

We ablate the experience hierarchy, support gate, and exploration on 301 paired mature events per model. Global, Global+Pattern, and Global+Pattern+Local enable the named levels with gating and no exploration. GovernRouter adds supported exploration. Full w/o support gate sets the correction and exploration support thresholds to zero.

Accumulated experience improves the backbone's choices. GovernRouter improves family-balanced utility over Frozen backbone by 6.15 points on Llama and 7.74 points on Qwen3 (Figure 21, right). Global experience supplies the largest sequential gain, 4.12 and 7.35 points. Local adds 0.92 and 0.43 points over Global+Pattern, with positive intervals on both models. Pattern adds 0.27 points on Llama and -0.02 points on Qwen3, whose interval crosses zero. The principal gain comes from Global experience, with further refinement from Local updates.

Support gating improves the use of experience. GovernRouter improves utility over the ungated configuration by 2.33 and 0.71 points, with positive intervals on both models. Negative override rates fall from 14.53% to 3.13% on Llama and from 5.88% to 4.82% on Qwen3 (Table 2). On Qwen3, GovernRouter also reduces negative overrides relative to Global+Pattern+Local (4.82% versus 6.44%), with event-mean utilities of 0.7208 and 0.7210. The gate improves aggregate experience use; Appendix Figure 12 reports window-level variation, including windows favoring the ungated configuration.

E.2 ADDITIONAL REUSE AND GOVERNANCE RESULTS

The following tables retain the complete condition comparisons referenced in the main experimental summary.

Table 8: Family-macro effects of experience reuse with 95% confidence intervals. Positive values favor the left condition. Each model uses 49 failure-family blocks. Non-prefix comparisons contain 451 paired events. Prefix comparisons contain 332 paired events on the shared suffix.

Comparison	Llama: effect [95% CI]	Qwen3: effect [95% CI]
Matched − Reset	0.0639 $[0.0504, 0.0779]$	0.0663 $[0.0506, 0.0824]$
Global − Reset	-0.0286 $[-0.0724, 0.0123]$	0.0307 $[0.0130, 0.0478]$
Global-prefix − Global	-0.00014 $[-0.00041, 0]$	0.0074 $[0, 0.0195]$
Global-prefix − Matched	-0.0789 $[-0.1088, -0.0517]$	-0.0310 $[-0.0431, -0.0200]$

Table 9: Repair and commit behavior of governance components. Values aggregate the rates reported by individual runs. SafeRepairSuccess is the safe repair success rate, FalseCommit is the false commit rate, and CAS conflict is the conflict rate. Aggregate rates cannot be converted into event counts from the case totals in this section.

Configuration	Llama: success / false commit / conflict	Qwen3: success / false commit / conflict
Copy-on-write	1.0000 / 0 / 0	1.0000 / 0 / 0
Contract without CAS	1.0000 / 0 / 0	1.0000 / 0 / 0
Contract+CAS	0.1250 / 0 / 0.5385	0.3333 / 0 / 0.3702
Full governance	0.1250 / 0 / 0.5385	0.3333 / 0 / 0.3702

E.3 Distribution Recurrence

The A1–B–A2 experiment tests repair choices under an initial stationary distribution, an abrupt change, and a return to the original distribution. Four Qwen3 reports cover two sources and two order seeds, with 40, 54, and 34 paired mature events across the three phases.

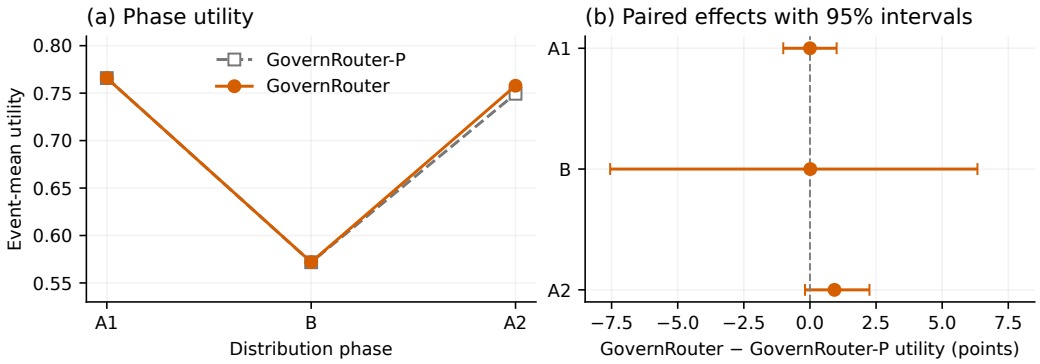


Figure 6: Repair behavior under distribution recurrence. Left: event-mean utility of GovernRouter and GovernRouter-P in A1, B, and A2. Right: GovernRouter minus GovernRouter-P, with equal weighting across failure families and pointwise 95% bootstrap intervals; one point equals 0.01 utility. Each phase uses common mature events. Phase composition differs, so the curves do not measure learning or forgetting on a fixed test set. Results use structural repair feedback.

Figure 6 separates phase utility from uncertainty in the paired comparison. In A2, GovernRouter exceeds GovernRouter-P by 0.85 event-mean utility points and selects targeted repair 14 rather than 12 times, with rebuild selections falling from 22 to 20. The family-balanced effect is 0.93 points, with a $[-0.19, 2.25]$ interval. The records show changed repair choices on return to A, but do not establish a significant return-phase advantage. Full phase values appear in Appendix Table 10; stationary window analyses remain in Appendix Figure 12.

E.4 Supplementary Repair Evaluations

E.4.1 Skill Transfer across Disjoint Failure Families

The separate family-disjoint evaluation tests frozen Skill content on unseen failure families. It contains 24 runs and 12 paired streams, with 22 families and 62 paired events per model. Skills-reset improves over Base-reset by 0.64 family-balanced utility points on Llama, with a $[0, 1.91]$ interval; Qwen3 is tied throughout. Figure 16 reports the paired effects. This test evaluates Skill-content transfer, separately from the shared-case routing-state reuse evaluated in Section 4.2.

E.4.2 Repair Utility and Locality

Figure 7 compares utility and locality cost on the same 301 mature events per model. Both axes weight events equally, unlike the stream-weighted utility in Table 1. Relative to Fixed-order legal,

GovernRouter increases event-mean utility by 5.91 points on Llama and 6.34 points on Qwen3 while reducing locality from 3.412 to 2.767 and 2.957. Several adaptive controls use more local repairs with lower utility. These quantities describe structural repair quality and logical scope; execution time and monetary costs are not measured here.

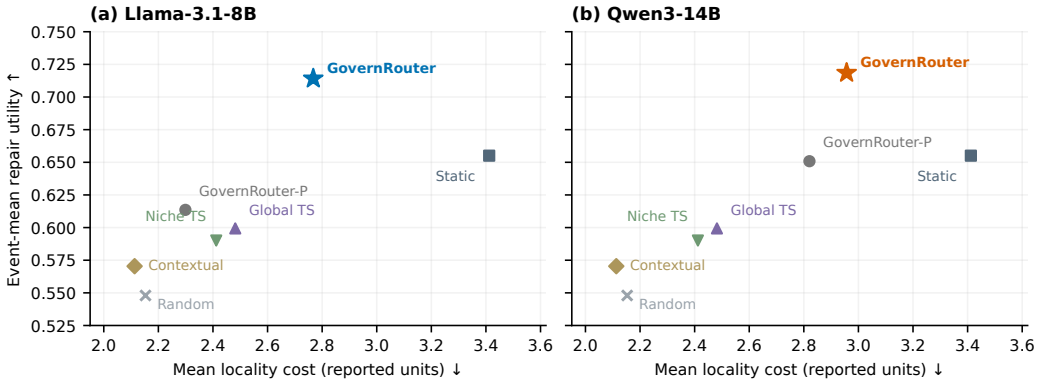


Figure 7: Repair utility and locality cost for all seven routing configurations under Matched initialization. Both axes use event means from the same all-arms reports. GovernRouter has higher utility and lower locality cost than the fixed-order legal selector (labelled Static in the figure) on both models, but several adaptive controls have lower locality cost. Utility already includes cost penalties, so the axes describe related quantities rather than independent measures of quality and efficiency. Points are descriptive means without confidence intervals.

E.4.3 Concurrent-Update Handling

The independent sealed-sidecar evaluation covers 26 Llama cases and 68 Qwen3 cases. CAS-enabled configurations record conflict rates of 0.5385 and 0.3702 and safe-repair success rates of 0.1250 and 0.3333, compared with zero conflicts and unit success without CAS. All evaluated configurations record zero false commits. These results distinguish rejection of stale writes from successful repair; the shared zero false-commit result does not establish a safety advantage. Table 9 and Figure 15 preserve the full comparison and the report-specific aggregation.

E.5 Temporal Adaptation and Skill Ecology

We examine the feedback loop through routing-state trajectories and Skill lifecycle operations. Separate lifecycle runs evaluate registration and version management, with retirement explicitly triggered in replay. Distribution recurrence is evaluated in Appendix E.3.

Feedback revises routing state. On HaluMem abrupt streams under Reset, receipts increase coordinate support and revise Global residual means (Figure 8). The illustrated Llama and Qwen3 coordinates develop positive and negative means, respectively, showing bidirectional prediction adjustments through Equations 4–6.

Version management maintains Skill ecology. Six population-ecology streams record registration, revision, and retirement. Figure 9 shows that registration adds historical revisions while retirement changes their status. Dedicated coverage exercises one deduplication and one supersession while preserving version relations. These traces demonstrate the version and lifecycle operations underlying Skill ecology (Section 3.4).

Concurrent-update results are reported in Appendix E.4.

F Supplementary Experimental Figures

The figures in this section provide detailed views of routing comparisons, mechanism ablations, temporal behavior, governance, and lifecycle operations. They preserve the weighting, missing-value

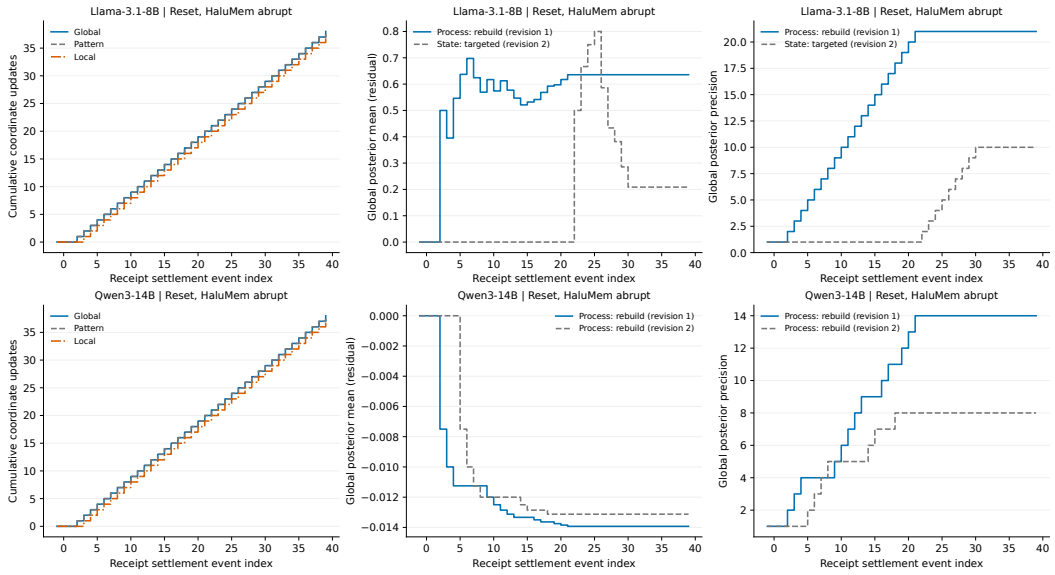


Figure 8: Posterior updates on HaluMem abrupt streams (Reset; seed 20260827): cumulative updates by level and means and precisions of the two most frequently updated Global coordinates. Event -1 denotes the prior. Precision is an internal parameter, not accuracy or calibrated confidence.

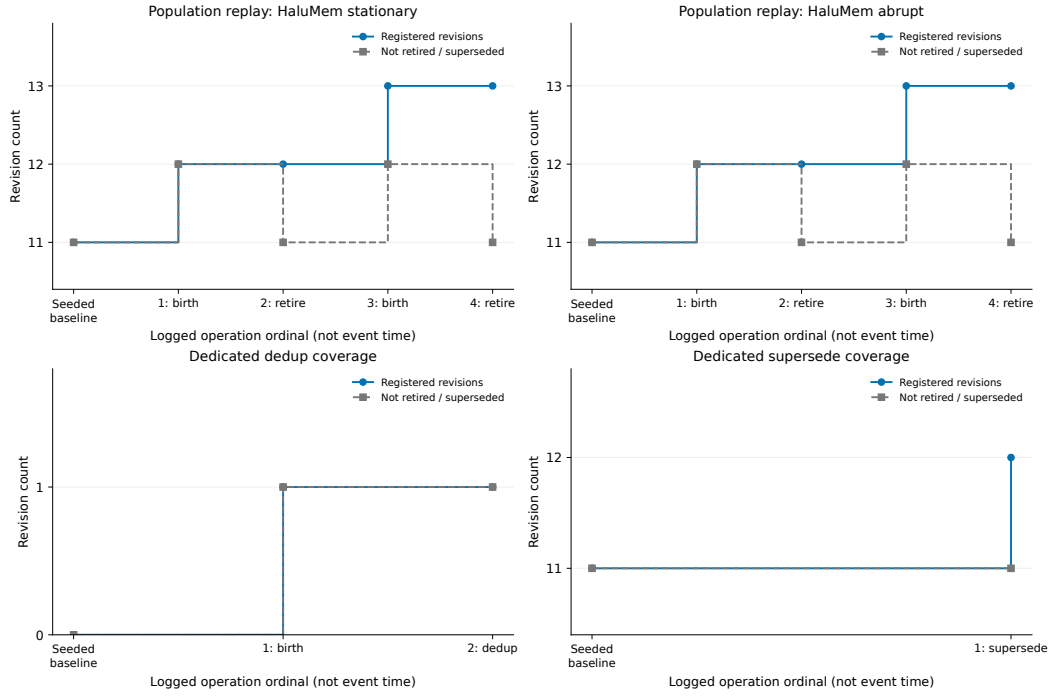


Figure 9: Registry trajectories for two HaluMem add-revision-retirement examples and dedicated deduplication and supersession coverage records. The horizontal axis is logged operation order rather than event time. Registered revisions include historical versions, while the second count excludes retired and superseded revisions and is not a count of deployable Skills.

rules, and evaluation scope of the corresponding tables and main figures. Trajectories describe recorded events; lifecycle plots document operation coverage. Each caption specifies the population and interpretation of its comparison.

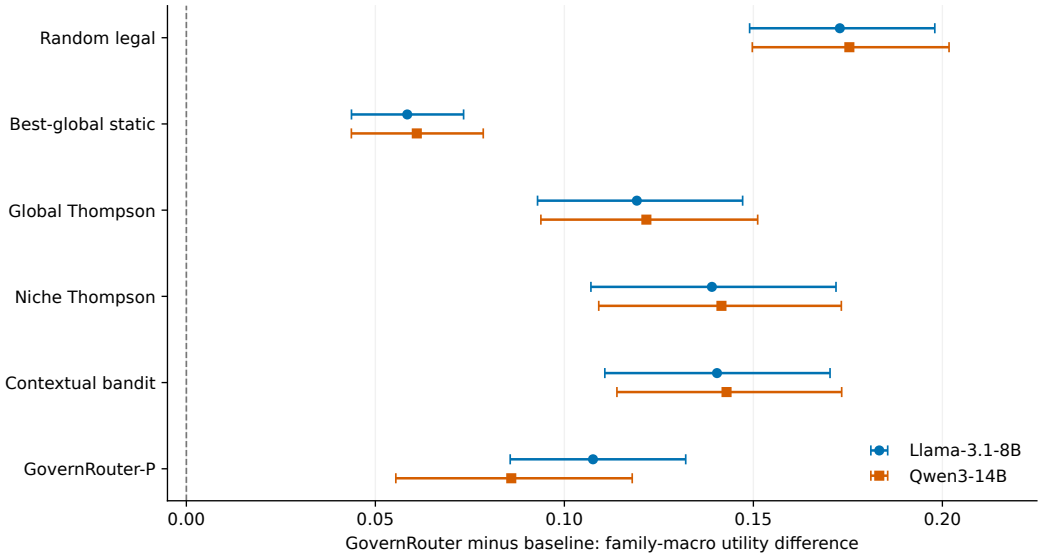


Figure 10: Enlarged family-macro comparison of GovernRouter with six routing controls. Each model contributes 49 failure families and 301 paired mature events. Points are paired utility effects, and bars are pointwise 95% family-blocked bootstrap intervals.

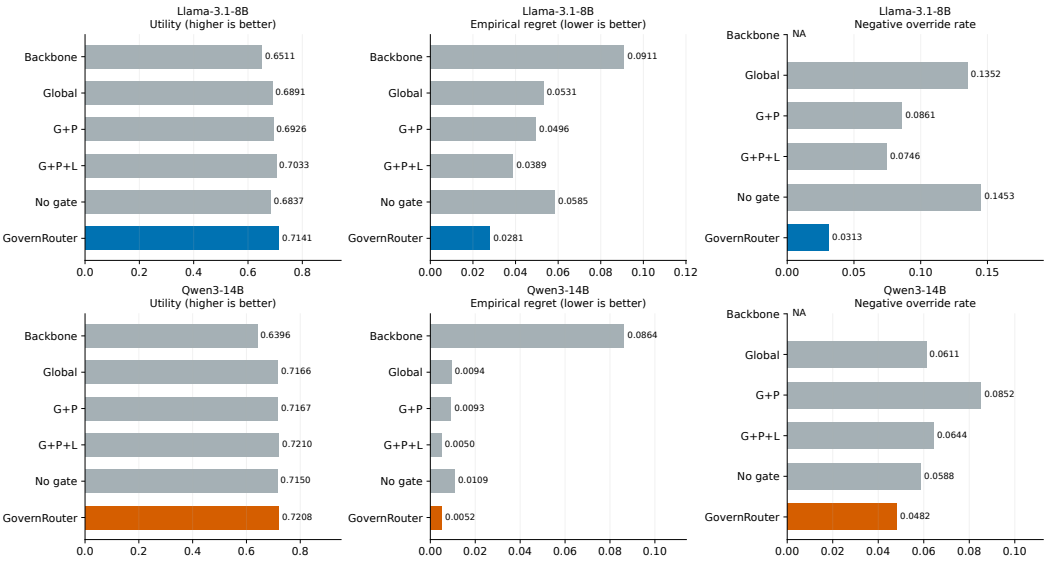


Figure 11: Absolute outcomes for 24 routing-mechanism ablation runs. The panels report event-weighted utility, empirical regret relative to the best realized outcome among the six evaluated configurations, and aggregate negative override rate. Frozen backbone has no override decision, so its override rate is undefined rather than zero.

F.1 Additional Comparison Views

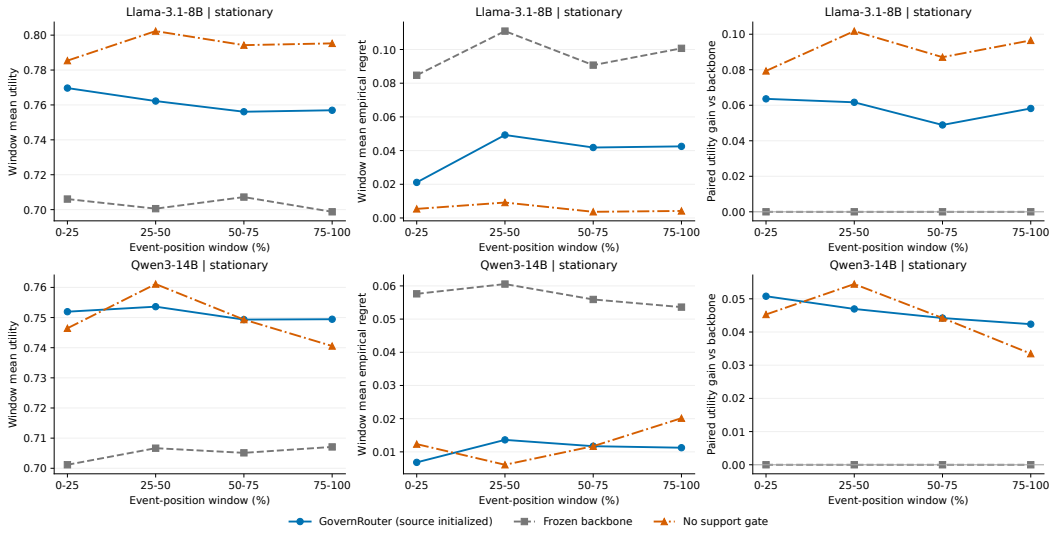


Figure 12: Windowed routing outcomes on stationary ablation streams. The panels report mature-event utility, empirical regret against the six realized ablation outcomes, and paired utility relative to Frozen backbone. Events are aligned by selection position, which does not imply that their receipts were visible at that position. Window composition varies, so the curves do not establish monotonic improvement.

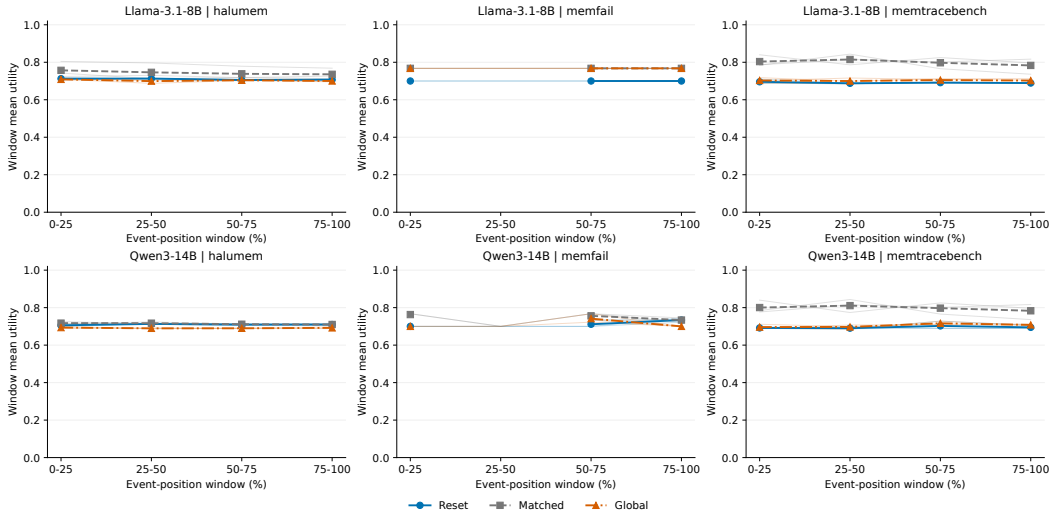


Figure 13: Stationary trajectories under Reset, Matched, and Global initialization for two models and three data sources. Thin lines show individual order seeds, and thick lines show equal-seed means only when all three seeds have observations. The initialization conditions are separate experiments rather than successive stages of one learning process.

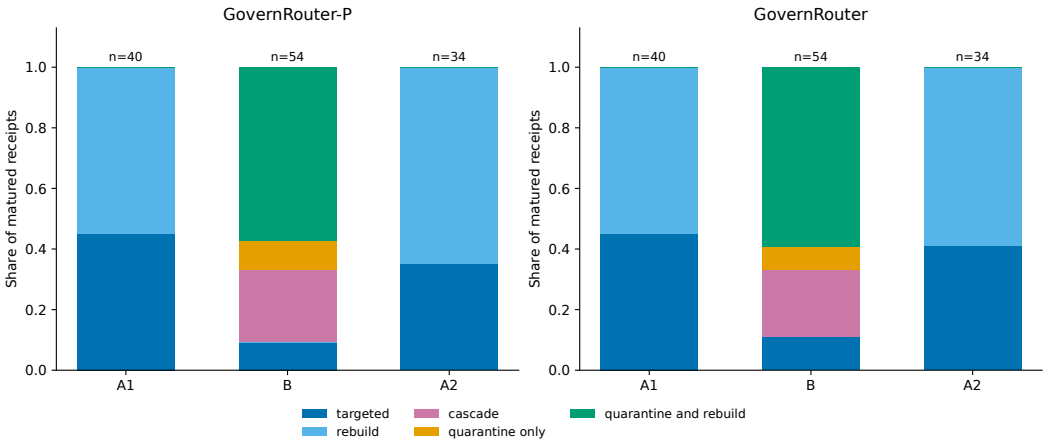


Figure 14: Strategy composition by phase and router in the A-B-A experiment. Counts above bars give mature receipts in each phase. Because failure composition changes across phases, strategy shares describe changes in action selection rather than diagnostic accuracy.

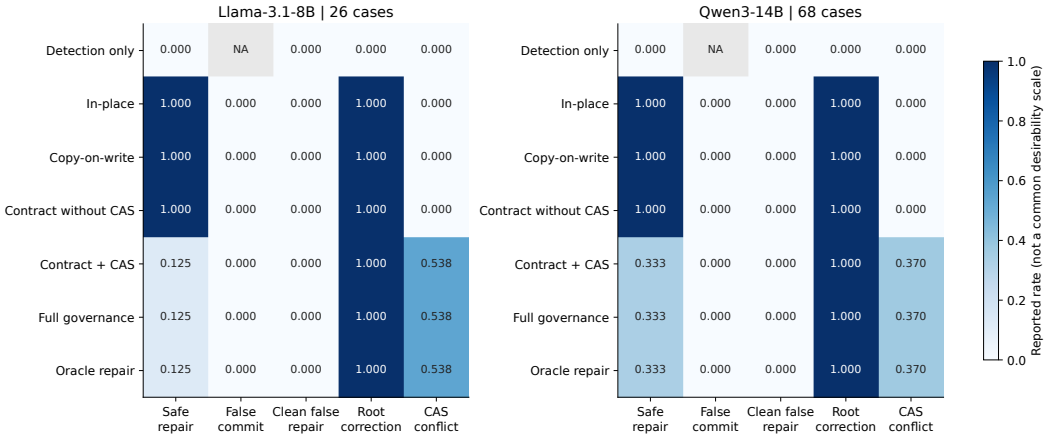


Figure 15: Governance outcomes from eight sealed-sidecar reports. The figure shows safe repair, false commit, clean false repair, root correction, and CAS conflict rates using the aggregation stored in each report; missing values remain undefined. Every method has zero observed false commits, so this metric does not distinguish Full governance from its controls. CAS rejection is a concurrency outcome and does not by itself measure repair ability.

Table 10: A-B-A results by phase. The four runs evaluate Qwen3 on two sources and two random seeds. Utility is the event mean within each phase. The last column reports the family-macro effect of GovernRouter minus GovernRouter-P with a 95% confidence interval. A1, B, and A2 contain 40, 54, and 34 paired events, respectively.

Phase	GovernRouter-P utility	GovernRouter utility	Family-macro effect [95% CI]
A1	0.7659	0.7659	$0\ [-0.0101, 0.0101]$
B	0.5718	0.5719	$0.0001\ [-0.0756, 0.0633]$
A2	0.7493	0.7578	$0.0093\ [-0.0019, 0.0225]$

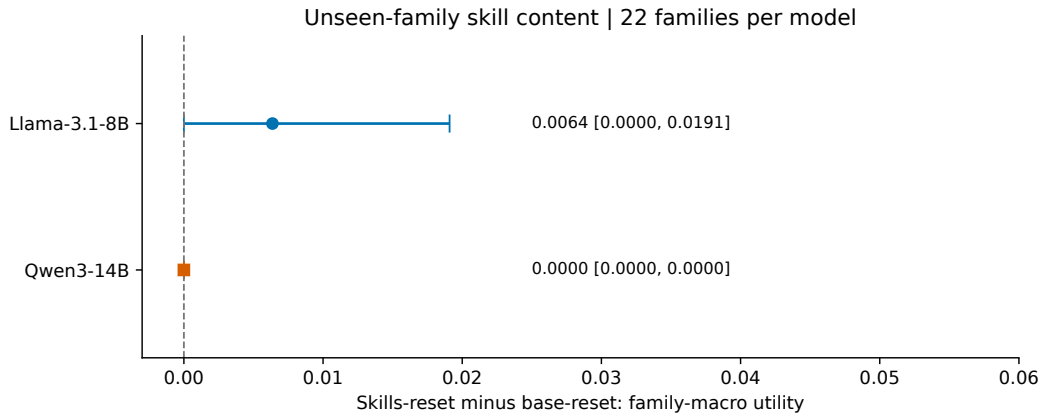


Figure 16: Transfer of frozen Skill content to unseen failure families. The 24 family-disjoint runs contain 12 paired streams, with 22 families and 62 paired events per model. Points and bars show family-macro effects and 95% intervals for Skills-reset relative to Base-reset. Qwen3 is tied on every event, and the Llama interval includes zero.

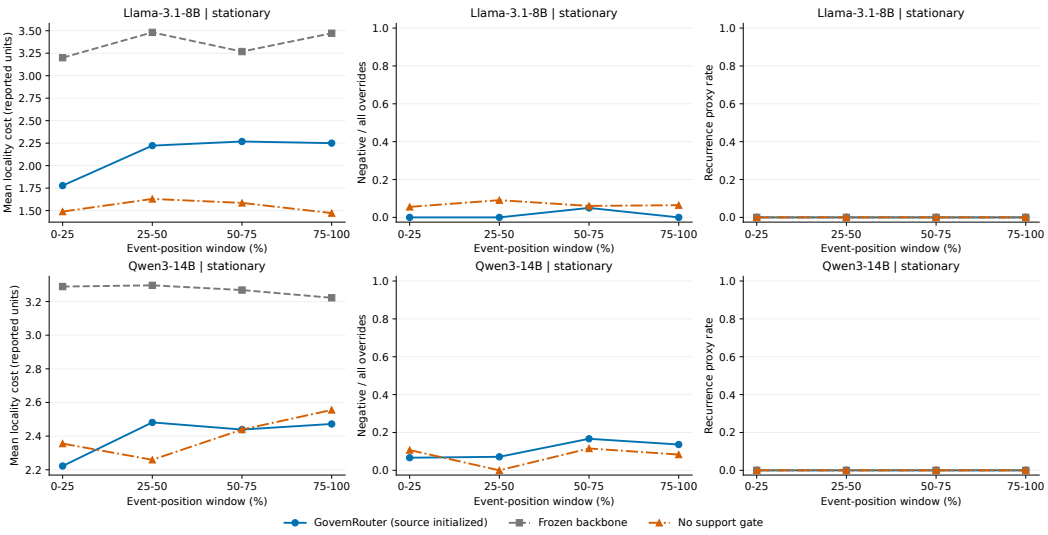


Figure 17: Windowed locality cost and override behavior on the stationary paired events used in Figure 12. Locality cost uses the units stored in the reports. Negative override rate divides negative overrides by all overrides and remains undefined when no override occurs. The recurrence signal is a structural proxy and does not measure long-term risk.

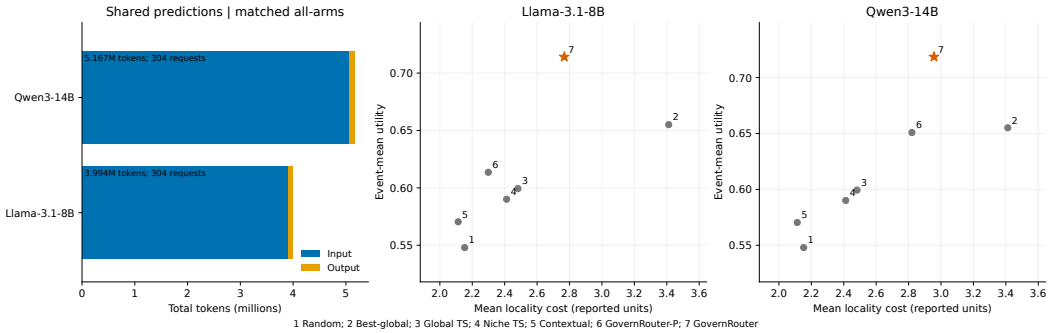


Figure 18: Model calls and routing locality. The left panel counts shared prediction input and output tokens in the Matched all-arms reports, with 12 reports and 304 requests per model; the counts are not per-router inference costs. The right panel compares event-weighted utility with locality cost. The points do not constitute a measured system-level Pareto frontier, and unmeasured monetary and GPU costs are not inferred.

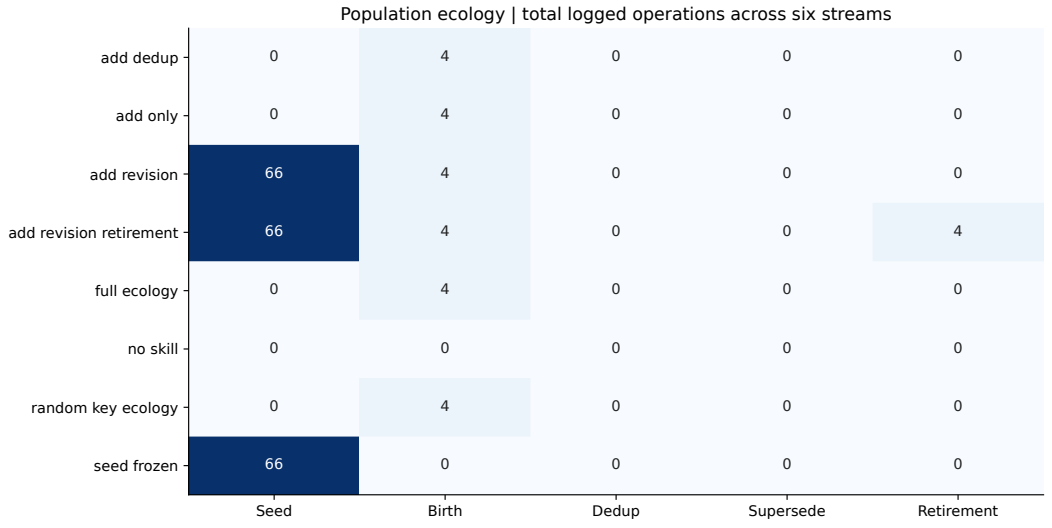


Figure 19: Logged population-ecology operations across six streams per method. Counts include seed, birth, deduplication, supersession, and retirement, including observed zeros. Methods start with different numbers of seeded revisions, so larger registry counts do not imply a better ecology. Dedicated deduplication and supersession coverage records are excluded from these natural replay totals.

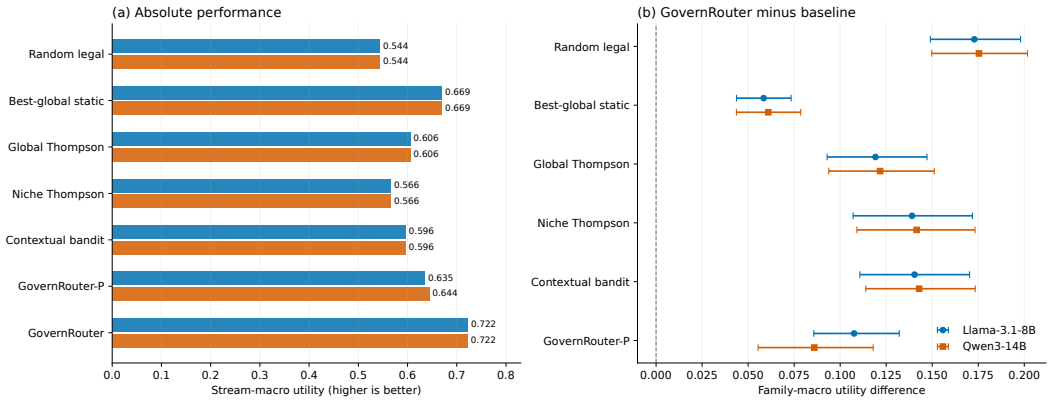


Figure 20: Overall routing comparison under Matched initialization. The left panel reports stream-macro utility across 12 streams per model. The right panel reports the family-macro paired effect of GovernRouter relative to six controls over 49 failure families and 301 paired mature events per model; bars show pointwise 95% family-blocked bootstrap confidence intervals. The panels use different weighting schemes, so their values should not be subtracted directly.

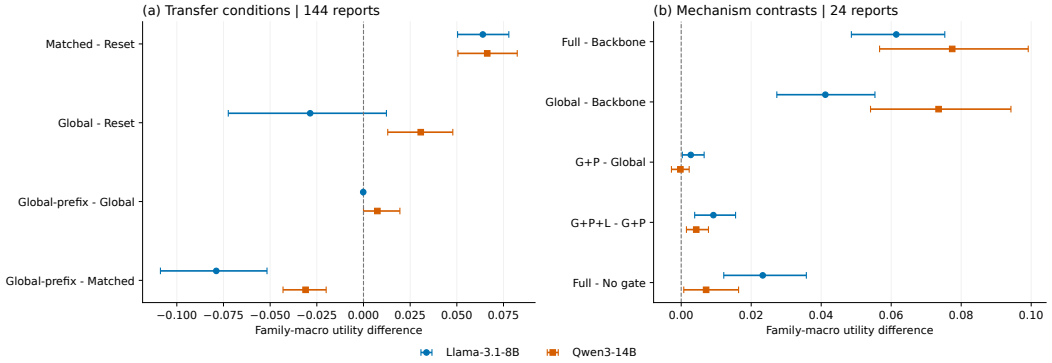


Figure 21: Experience transfer and routing mechanisms. The left panel compares Reset, Matched, Global, and Global-prefix using 144 GovernRouter-only runs; comparisons involving Global-prefix use the shared evaluation suffix. The right panel reports sequential mechanism contrasts from 24 ablation runs. Effects are averaged over 49 failure families, and bars show pointwise 95% family-blocked bootstrap confidence intervals. The contrasts depend on their comparison configurations and are not additive.

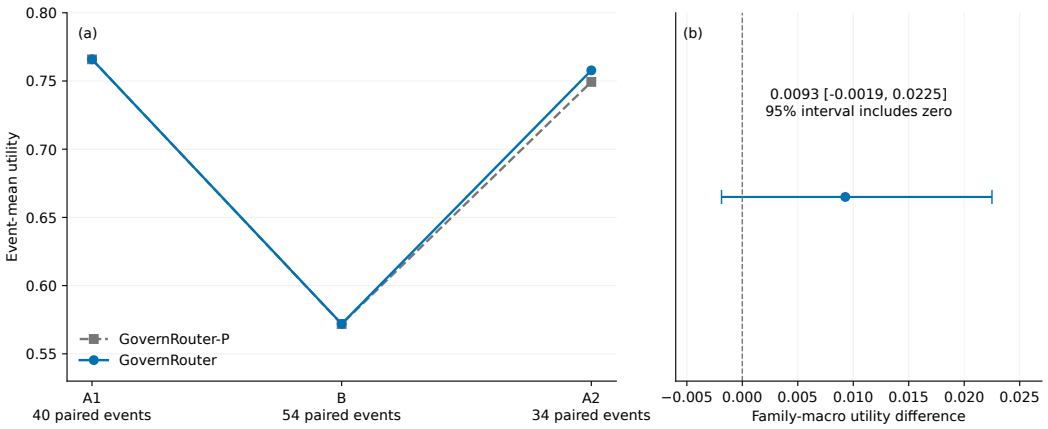


Figure 22: Phase utility and return-phase difference under A-B-A distribution changes on Qwen3. Panel (a) reports event-mean utility for each phase. Solid circles denote GovernRouter, and dashed squares denote GovernRouter-P. Event composition differs across phases, so connecting lines only indicate phase order. Panel (b) reports the paired effect over 26 failure families in A2 with its 95% confidence interval. The vertical dashed line marks zero.