MEMORY TRANSPLANTS FOR LLM AGENTS: DISENTANGLING ARCHITECTURE AND CONTENT TRANSFER UNDER A CODE-TO-MATH SHIFT

Zhaoxiang Feng[*] **Mingyang Yao**[*] **David Scott Lewis**[†]

ABSTRACT

Memory-augmented LLM agents accumulate experience to improve over time, but when transferring to a new domain, observed gains may stem from either the memory *mechanism* (how experiences are stored and retrieved) or the stored *content* (the experiences themselves). Prior cross-domain evaluations conflate these two factors, leaving it unclear which component generalizes. We introduce a memory transplant protocol that disentangles architecture from content by independently varying each across a code → math domain shift (LiveCodeBench → MATH). Using a 2×2 factorial design with seven transplant conditions, five memory systems spanning simple RAG to evolved multi-tier architectures, and six pre-registered validation gates, we evaluate two solver scales (Qwen 2.5 7B and Llama 3.2 3B) under both static (retrieval-only) and dynamic (full learning) regimes. We find that architecture transfer is system-dependent with no universal direction, and that content transfer in static mode provides limited benefit beyond a no-memory baseline. The most striking result is that solver capability moderates transfer magnitude: the weaker model shows gains up to $+15$ percentage points versus $+7$pp for the stronger, suggesting that memory transplantation is most valuable where intrinsic model capability is limited.

1 INTRODUCTION

Memory systems for LLM agents can be decomposed into three core operations: PROVIDE (retrieve relevant experience), TAKE-IN (write new experience), and MANAGE (organize, prune, consolidate) (Zhang et al., 2025b). Recent work has shown that these operations can be evolved or tuned to improve within-domain performance (Zhang et al., 2025b), and that externalized experience summaries, without weight updates, can improve agent performance through distilled insights (Zhao et al., 2024) or continual learning from verbal feedback (Majumder et al., 2023).

However, when an agent with accumulated experience moves to a new domain, existing evaluations conflate two distinct sources of potential improvement. The first is **architecture transfer**, in which the memory *mechanism* (retrieval frequency, pruning rules, gating policies) generalizes to new domains. The second is **content transfer**, in which the memory *items* (stored insights, trajectories, error patterns) are directly useful in the new domain. No prior work cleanly separates these two factors. Cross-domain experience reuse studies (Tang et al., 2025; Xu et al., 2025a) evaluate combined architecture-plus-content transfer, making it impossible to attribute observed gains. Architecture search or evolution work (Zhang et al., 2025b; 2026) operates within a single domain, leaving open the question of whether evolved mechanisms generalize.

We propose a **memory transplant protocol** that enables independent transfer of architecture and content across a substantial domain shift: unit-test-verifiable coding problems (LiveCodeBench; Jain et al. 2024) → competition math (MATH; Hendrycks et al. 2021). The protocol enforces a *prompt-freeze rule* (identical solver prompts across all conditions), standardized canonical export/import of memory items, and required negative controls for token-budget confounds and retrieval attribution. The choice of code-to-math transfer is deliberate: both domains require multi-step reasoning but differ substantially in surface form, vocabulary, and solution strategies. This makes the transfer

[*]University of California San Diego [†]AI Executive Consulting (AIXC)

question non-trivial, because it is unclear *a priori* whether coding insights ("always check edge cases") or retrieval strategies ("retrieve every 3 episodes") would help with competition math.

Our contributions are threefold. First, we define a **memory transplant protocol** with explicit operational definitions separating architecture from content, canonical export/import contracts, and a prompt-freeze rule that prevents domain knowledge from leaking through prompt variations. Second, we specify a **pre-registered experimental design** with seven transplant conditions in a 2×2 factorial, six validation gates, and four negative control types, enabling conditional causal claims about what transfers. Third, we report **empirical results** across five memory systems and two solver scales (360+ evaluation runs), finding that architecture transfer is system-dependent, content transfer in static mode is limited, and solver capability moderates the magnitude of transfer effects.

2 Related Work

Memory-augmented LLM agents. Retrieval-augmented generation (Lewis et al., 2020) provides a foundation for injecting external knowledge into LLM context, and subsequent work has extended this idea to persistent agent memory. Generative agents (Park et al., 2023) introduced persistent episodic memory with reflection for social simulation, where agents retrieve and synthesize past experiences to produce coherent long-horizon behavior. MemGPT (Packer et al., 2023) treats memory as an operating system resource with explicit read/write/manage operations, introducing a hierarchical architecture that enables agents to maintain working context beyond fixed window limits. Reflexion (Shinn et al., 2023) uses verbal self-reflection stored as experience for subsequent attempts, demonstrating that textual feedback alone can improve performance without weight updates. MemoryBank (Zhong et al., 2024) adds forgetting mechanisms inspired by the Ebbinghaus curve, selectively discarding stale memories to keep the store relevant over time. More recently, A-MEM (Xu et al., 2025b) proposes an "agentic memory" framing in which the agent actively organizes and links memories rather than treating them as passive storage, and ArcMemo (Ho et al., 2025) introduces abstract concept-level memory for reasoning tasks, demonstrating that modular, context-independent abstractions distilled from solution traces outperform instance-level memories on the ARC-AGI benchmark. These systems demonstrate within-domain improvement but do not address the question of what transfers when the domain changes.

Experience distillation and learning without weight updates. A growing body of work demonstrates that agents can accumulate reusable knowledge from interactions without any parameter changes. ExpeL (Zhao et al., 2024) distills successful trajectories into reusable insights via a dual-store architecture (insights plus trajectories), demonstrating experiential learning across task instances within the same domain. CLIN (Majumder et al., 2023) maintains a growing memory of causal abstractions that enable rapid task adaptation, reporting $+23$ absolute points over Reflexion in ScienceWorld and measurable transfer to new environments and tasks. Contextual Experience Replay (CER; Liu et al. 2025) accumulates and synthesizes past experiences into a dynamic memory buffer, achieving substantial gains on WebArena and VisualWebArena through training-free in-context experience replay. MemGen (Zhang et al., 2025a) takes a different approach entirely, proposing generative latent memory tokens that are injected directly into the model's reasoning stream, claiming improved cross-domain generalization over text-based memory systems like ExpeL and AWM. These systems show that memory content can improve performance, but they do not separate whether gains come from the *content* (the specific insights stored) or the *mechanism* (how items are stored, organized, and retrieved).

Memory architecture design and evolution. Beyond handcrafted memory systems, recent work explores automated discovery of memory architectures. MemEvolve (Zhang et al., 2025b) meta-evolves PROVIDE/TAKE-IN/MANAGE policies within a single domain, producing architectures such as LIGHTWEIGHT_MEMORY (dual-tier storage with retrieval frequency control) and CEREBRA_FUSION (graph-based retrieval with success feedback). MemSkill (Zhang et al., 2026) frames memory operations as learnable "skills" that can be evolved and composed, with explicit analysis of generalization under distribution shift across LoCoMo, LongMemEval, HotpotQA, and ALFWorld. Agent Workflow Memory (AWM; Wang et al. 2024) induces reusable workflows from agent interactions and selectively retrieves them for new tasks, reporting cross-website and cross-domain generalization within web navigation, with absolute gains of 8.9–14.0 points on WebArena and Vi-

sualWebArena. Coarse-to-Fine Grounded Memory (CFGM; Yang et al. 2025) represents experience at multiple granularities, from coarse focus points to fine-grained anomaly corrections, and uses these to support flexible planning. Our work takes these architectural innovations as given and tests whether they generalize across a substantial domain shift, a question that within-domain evolution and evaluation cannot answer.

Cross-domain memory transfer. The question of whether experience collected in one domain helps in another has received increasing attention, though no prior work cleanly isolates architecture from content as independent factors. Agent KB (Tang et al., 2025) provides a universal memory infrastructure for cross-framework experience sharing with a two-stage hybrid retrieval pipeline and a disagreement gate to reduce harmful interference. Notably, Agent KB reports *asymmetric transfer*: reasoning experience can help some software engineering settings, but the reverse does not hold, a finding that parallels the transfer asymmetries we observe in our factorial design. SEDM (Xu et al., 2025a) introduces verifiable memory lifecycles with paired A/B replay to estimate the marginal utility of candidate memory items, and includes an explicit cross-domain experiment between FEVER and HotpotQA where memory collected on one dataset is evaluated on the other. However, both Agent KB and SEDM evaluate content transfer under a single fixed architecture, making it impossible to determine whether gains stem from the transferred items, the retrieval mechanism, or their interaction. Our transplant protocol addresses this gap by independently varying architecture and content as orthogonal experimental factors.

Lifelong and streaming agent evaluation. Evaluating memory transfer requires benchmarks that go beyond static single-task assessment. LifelongAgentBench (Zheng et al., 2025) enforces strict sequential execution across Database, Operating System, and Knowledge Graph environments to measure how past experience impacts future tasks, reporting that experience replay can dramatically boost success (e.g., Llama-3.1-8B from 0.19 to 0.78 on DB tasks) but also that excessive replay degrades performance due to irrelevant information and context limits. Evo-Memory (Wei et al., 2025) restructures 10 diverse datasets into sequential streams and implements 10+ memory modules under one harness, providing a comprehensive comparison of memory methods for test-time evolution, though it primarily measures within-stream learning rather than cross-domain transplantation. StreamBench (Wu et al., 2024) benchmarks continuous improvement across text-to-SQL, programming, tool use, medical diagnosis, and QA, establishing the streaming evaluation philosophy that we adopt. Our protocol extends streaming evaluation with explicit transplant conditions and the static/dynamic mode distinction that enables controlled attribution of architecture versus content effects.

3 Memory Transplant Protocol

The memory transplant protocol enables independent manipulation of architecture and content as experimental factors. We first define these terms precisely, then describe the export/import contract and the resulting transplant conditions. Figure 1 provides an overview of the protocol.

3.1 Operational Definitions

We define *architecture* as the algorithmic policies and hyperparameters governing PROVIDE, TAKE-IN, and MANAGE operations. This includes retrieval strategy (lexical, two-stage, embedding-based), retrieval frequency (every episode vs. every N episodes), top-k settings, token budget allocation across stores, pruning thresholds, tier routing rules (which tier receives successful vs. failed episodes), gating mechanisms (disagreement filtering), and consolidation schedules. Architecture *explicitly excludes* any domain-specific demonstrations, heuristics, examples, or few-shot prompt content; any such material is treated as content and must be held constant across conditions.

Content is the transferable set of memory items: a canonical list of textual entries plus minimal metadata, exported as JSONL after a build stream. Each `CanonicalMemoryItem` contains an `id` (stable unique identifier), `text` (the memory content as plain text), `type` (categorical label: `strategic`, `operational`, `error_trace`, or `other`), `source_domain` (`code` or `math`), `episode_id` (originating build problem), `success` (boolean success signal from the source task), and `order_index` (position in the build stream). Importantly, architecture-dependent derived

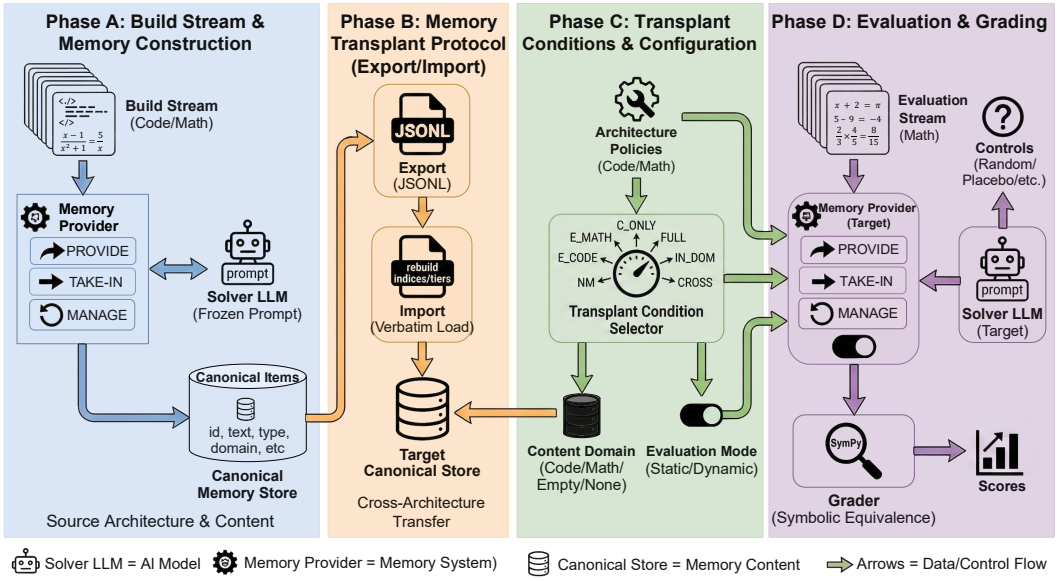

Figure 1: Overview of the memory transplant protocol. Architecture (PROVIDE/TAKE-IN/MANAGE policies) and content (canonical memory items) are independently varied as experimental factors across a code-to-math domain shift. The seven transplant conditions arise from crossing architecture domain $\in \{\text{code}, \text{math}, \text{none}\}$ with content domain $\in \{\text{code}, \text{math}, \text{empty}, \text{none}\}$ under static or dynamic evaluation modes.

structures (embeddings, tier placements, graph edges, utility scores) are *not* part of the canonical content; these are recomputed on import by the receiving architecture.

The core solver prompt is identical across all conditions and architectures. Memory-system differences are isolated to memory injection sections that are clearly demarcated in the prompt. A SHA-256 hash of the prompt template is logged with every run and verified to be constant across all runs within a plan (Gate G2), eliminating prompt leakage as a confound.

3.2 EXPORT/IMPORT CONTRACT

After completing a *build stream*, a sequence of problems processed with all memory operations enabled, surviving canonical items (post-MANAGE) are serialized as JSONL. On import, items are loaded verbatim into the receiving system's store. Architecture-dependent structures are recomputed by the target architecture's MANAGE policies without any LLM-based rewriting or summarization. This ensures that the import step does not introduce domain adaptation that could confound the content-transfer measurement.

3.3 TRANSPLANT CONDITIONS

We define seven conditions by independently varying two factors, *architecture domain* $\in \{\text{code}, \text{math}, \text{none}\}$ and *content domain* $\in \{\text{code}, \text{math}, \text{empty}, \text{none}\}$, as shown in Table 1.

These seven conditions enable several testable hypotheses. **H1** (Architecture transfer) compares E_CODE to E_MATH: does the code-selected architecture outperform math-selected when both start empty? **H2** (Content transfer) compares C_ONLY to NM: do code-derived items help under the math architecture with learning disabled? **H3** (Interaction) examines the diff-in-diff across the 2×2 factorial (C_ONLY, FULL, IN_DOM, CROSS): are architecture and content effects non-additive? **H4** (PROVIDE attribution) compares write-only controls to normal operation: does retrieval contribute beyond the reflection and write side effects? Evaluation of H4 requires the negative controls described in Section 4.5; results are deferred to supplementary analysis. Without the target-architecture cold start (E_MATH), any advantage of E_CODE could be misread as "transfer" when it is merely

Table 1: Transplant condition matrix. Architecture and content are independently varied. "Dynamic" mode enables TAKE-IN/MANAGE during evaluation; "static" disables them to isolate imported-content effects.

Condition	Architecture	Content	Mode	Purpose
NM	none	none	static	No-memory baseline
E_MATH	math	empty	dynamic	Target-arch cold start
E_CODE	code	empty	dynamic	Architecture-only transfer
C_ONLY	math	code	static	Content-only transfer
FULL	code	code	static	Full transfer (arch + content)
IN_DOM	math	math	static	In-domain reference
CROSS	code	math	static	Completes 2×2 factorial

a different learning-from-scratch trajectory; the E_MATH condition provides the critical counterfactual.

3.4 EVALUATION MODES

In **static mode**, TAKE-IN and MANAGE are disabled during the evaluation stream; only PROVIDE (retrieval) is active. This isolates the usefulness of imported content independent of any target-domain learning. Static mode is *required* for content-transfer claims (H2), because any observed benefit must come from the imported items rather than from online adaptation. In **dynamic mode**, all three operations (PROVIDE, TAKE-IN, MANAGE) are enabled during evaluation. This captures how the architecture shapes learning dynamics over the evaluation stream, that is, whether a code-configured retrieval frequency or pruning policy helps the agent learn math problems more efficiently. Dynamic mode is *required* for architecture-transfer claims (H1).

4 EXPERIMENTAL SETUP

4.1 DOMAINS AND BENCHMARKS

For the source domain we use a 60-problem subset of LiveCodeBench (Jain et al., 2024) for the build phase, scored with pass@1 exact-match. The code build stream generates memory items that encode coding strategies, debugging patterns, and problem-solving heuristics. For the target domain we use MATH (Hendrycks et al., 2021) with a 30-problem build set and a 100-problem evaluation set. Grading uses a three-stage pipeline: (1) extract the last numeric/symbolic token from the model response, (2) float comparison with tolerance 10^{-4}, (3) SymPy symbolic equivalence as fallback. This grader is validated against hand-checked cases before any experimental runs (Gate G3). Build and evaluation sets are strictly disjoint (Gate G1), verified at the problem-ID level with no overlaps.

4.2 MEMORY SYSTEMS

We evaluate five memory systems spanning a range of architectural complexity, from minimal baselines to evolved multi-component designs.

NO_MEMORY disables all memory operations. The solver receives the same prompt as all other conditions but with no retrieved context, establishing the intrinsic capability of the solver on MATH.

SIMPLE_RAG implements a flat store with a maximum of 200 items and lexical (term-overlap) retrieval (Lewis et al., 2020). New items are written after every episode; FIFO pruning removes the oldest items when capacity is exceeded. This represents the simplest possible memory augmentation.

LIGHTWEIGHT_MEMORY uses dual-tier storage inspired by MemEvolve (Zhang et al., 2025b): a short-term tier (80 items) and a long-term tier (160 items). Retrieval frequency control limits retrieval to every 3 episodes, reducing context noise. Success-based tier routing directs successful episodes to long-term storage (as `strategic` items) and failed episodes to short-term (as `error_trace` items), with the budget split 50/50 between tiers during retrieval.

AGENT_KB maintains a single store (220 items) with two-stage retrieval: an initial broad retrieval ($2\times$ top-k) followed by reranking based on success signal, item type, and recency (Tang et al., 2025). Optional disagreement gating filters items whose conclusions conflict with the solver's draft prediction, a mechanism inspired by Agent KB's interference reduction strategy.

EXPEL uses dual stores: 140 *insights* (distilled strategic knowledge) and 140 *trajectories* (detailed solution traces) (Zhao et al., 2024). Each episode generates both an insight and a trajectory item. Retrieval allocates 65% of the token budget to insights and 35% to trajectories, reflecting the assumption that abstract insights are more transferable than specific solution traces.

4.3 Solver Models

To assess whether solver capability moderates transfer effects, we run the full experiment with two models at different capability levels. Qwen 2.5 7B Instruct is a moderately capable model establishing a strong no-memory baseline of $\sim$64–65% on MATH. Llama 3.2 3B Instruct is a smaller model with baseline $\sim$37–38% on MATH, providing substantially more headroom for memory-driven improvement. Both use temperature 0 for deterministic decoding via the OpenRouter API.

4.4 Scale and Replication

Each plan targets 210 core condition evaluations: 5 systems $\times$ 7 conditions $\times$ 3 stream seeds $\times$ 2 retrieval budgets (400 and 800 tokens). Stream seeds control the random ordering of the 100 evaluation problems, addressing sequential dependence in dynamic runs, following the streaming evaluation philosophy advocated by StreamBench (Wu et al., 2024) and Evo-Memory (Wei et al., 2025). The 7B experiment completed all 210 runs; the 3B experiment completed 156 of 210 runs due to transient API failures, with all 70 condition cells populated (2–3 runs per cell).

4.5 Required Negative Controls

Four types of negative controls are pre-specified to address major confounds. **Random retrieval** replaces real retrieved items with random items from the same store, matched for count and approximate token budget, measuring whether retrieval *relevance* matters or whether any retrieved content helps. **Placebo context** fills the memory injection slot with neutral filler text at matched token count, testing whether gains come from additional context length alone rather than memory content. **Write-only** enables TAKE-IN/MANAGE but forces PROVIDE to return empty, testing whether improvements come from retrieval versus from the reflection and management side effects of memory operations. **Frozen-store marginal utility (MU)** takes an identical store snapshot and evaluates with PROVIDE on vs. off while TAKE-IN/MANAGE are disabled in both arms, directly measuring the marginal utility of retrieval from imported content. Controls are run for the LIGHTWEIGHT_MEMORY system (which offers the richest operation set for attribution) across both budgets and key conditions; control results are reported in supplementary analysis.

5 Validation Gates

Before evaluating any hypothesis, six pre-registered gates must pass. These gates are checked automatically by the analysis pipeline and documented in a gate report before any claim is evaluated.

6 Results

6.1 Qwen 2.5 7B (Strong Solver)

Tables 3 and 4 report mean accuracy (%) across 3 seeds for the strong solver at both retrieval budgets. The NO_MEMORY baseline ranges from 62–68% (variation across cells reflects different random stream orderings of the 100 evaluation problems), leaving limited headroom for memory-driven improvement. The best condition (AGENT_KB E_MATH at budget 400: 71.0%) represents only a $+6.7$pp gain over the NO_MEMORY baseline, and most effects fall within the 2–5pp range where confidence intervals overlap.

Table 2: Validation gates. All six pass for the 7B experiment. Gates G1–G3 also pass for the 3B experiment; remaining gates will be verified upon completion of the retry runs.

Gate	Description	7B	3B
G1	Split integrity: build and eval problem sets are disjoint	Pass	Pass
G2	Tuning containment: single prompt template hash across all runs	Pass	Pass
G3	Grader validation: symbolic equivalence grader passes hand-checked cases	Pass	Pass
G4	Order robustness: $\geq$3 stream seeds for all dynamic cells	Pass	–
G5	Budget robustness: results at both 400 and 800 token budgets	Pass	–
G6	Frozen-store MU: marginal utility evaluation artifacts present	Pass	–

Table 3: Math accuracy (%) on 100 MATH problems, Qwen 2.5 7B, retrieval budget **400 tokens**. Each cell: mean of 3 stream seeds. Dynamic conditions (E_MATH, E_CODE) allow target-domain learning; static conditions isolate imported-content effects.

System	NM	E_M	E_C	C_O	FULL	I_D	CRS
NO_MEM	64.3	63.0	64.0	67.7	65.0	63.0	62.3
SIMPLE_RAG	64.7	68.7	66.7	62.7	63.7	68.0	68.7
LW_MEM	67.0	66.0	62.0	62.7	65.7	66.0	64.7
AGENT_KB	68.7	**71.0**	66.3	66.0	65.0	65.0	69.7
EXPEL	62.3	67.0	67.7	63.3	65.7	64.7	**70.0**

6.2 HYPOTHESIS EVALUATION

H1: Architecture transfer. Comparing E_CODE (code-selected architecture, empty store, dynamic) to E_MATH (math-selected architecture, empty store, dynamic) reveals mixed results. AGENT_KB favors E_MATH at budget 400 ($71.0\% > 66.3\%$) but favors E_CODE at budget 800 ($68.0\% > 66.7\%$), while EXPEL consistently favors E_CODE ($+0.7$ to $+2.7$pp). The 3B solver shows a similar lack of consistent direction (Appendix D). Architecture transfer is therefore *system-dependent and budget-sensitive*, with no universal direction.

H2: Content transfer. Comparing C_ONLY (math architecture, code content, static) to NM (no memory) reveals limited content-transfer effects. EXPEL shows the only notable static content transfer (70.0% at budget 800 vs. 64.0% NM); no other system shows consistent gains from imported code content. The 3B solver confirms this pattern, with C_ONLY failing to consistently outperform NM for any system (Appendix D). Content transfer in static mode is therefore *limited*.

H3: Architecture $\times$ content interaction. The 2×2 factorial (C_ONLY, FULL, IN_DOM, CROSS) reveals non-additive patterns. For the 7B solver, EXPEL shows a notable asymmetry at budget 800: C_ONLY reaches 70.0% while FULL drops to 65.3%, suggesting that code architecture hurts when combined with code content. The 3B solver shows even more pronounced interactions (Appendix D), echoing the transfer asymmetry reported by Agent KB (Tang et al., 2025). H3 is supported, with stronger effects at lower solver capability.

Table 4: Math accuracy (%), Qwen 2.5 7B, retrieval budget **800 tokens**.

System	NM	E_M	E_C	C_O	FULL	I_D	CRS
NO_MEM	65.3	66.3	65.3	63.3	62.0	65.7	68.3
SIMPLE_RAG	61.3	69.0	67.0	63.7	66.3	69.0	65.0
LW_MEM	62.0	65.0	67.0	60.3	64.3	65.7	66.7
AGENT_KB	62.0	66.7	**68.0**	64.3	64.7	67.7	66.7
EXPEL	64.0	65.0	67.7	**70.0**	65.3	68.0	66.0

7 ANALYSIS

7.1 SOLVER CAPABILITY MODERATES TRANSFER EFFECTS

Comparing the 7B results above with the 3B results in Appendix D reveals that solver capability moderates the magnitude of memory effects. The 7B baseline sits at $\sim$64% with a best condition of 71.0% (+6.7pp) and most effects in the 2–5pp range. The 3B baseline drops to $\sim$37%, and the best condition reaches 52.0% (+15.0pp), with both larger absolute gains and larger variance (28–52%). This is consistent with a "headroom" hypothesis: stronger solvers can often solve problems without retrieved context, leaving less room for memory to help. The practical implication is that memory transplantation may be most valuable for deploying smaller, more efficient models in new domains.

7.2 DYNAMIC VS. STATIC MODE DIVERGENCE

Dynamic conditions (E_MATH, E_CODE) tend to produce the highest individual accuracies, but the pattern is not uniform across systems. For the 7B solver, AGENT_KB peaks in dynamic mode (E_MATH 71.0% at budget 400), but EXPEL's best result is a static condition (C_ONLY at budget 800: 70.0%), showing that high-quality distilled content can match online learning. The 3B solver shows a clearer dynamic advantage, with the best dynamic condition exceeding the best static by 7.7pp (Appendix D). The relative value of architecture versus content transfer thus appears capability-dependent: weaker models benefit more from dynamic learning mechanisms, while stronger models can leverage high-quality static content effectively, broadly consistent with MemSkill (Zhang et al., 2026) and MemEvolve (Zhang et al., 2025b) where evolved policies generalize better than fixed stores.

7.3 SYSTEM-SPECIFIC TRANSFER PATTERNS

Transfer effects are not uniform across memory systems. AGENT_KB shows the strongest dynamic gains (E_MATH 71.0% at budget 400), with two-stage retrieval benefiting from online learning but appearing sensitive to content quality, consistent with SEDM's finding that verifiable memory admission can improve cross-domain transfer (Xu et al., 2025a). EXPEL shows the strongest content-sensitive patterns: its insight-distillation design produces more abstract, potentially domain-general items, explaining why EXPEL is the only system showing meaningful static content transfer (C_ONLY at 800: 70.0%). However, EXPEL also shows the widest variance across conditions, suggesting that insight-level abstractions can mislead when domain-inappropriate, a dual nature that recalls Ho et al.'s finding that concept-level memories are "the most consistent memory design" for ARC-AGI (Ho et al., 2025). LIGHTWEIGHT_MEMORY shows the most stable performance, with retrieval frequency control (every 3 episodes) acting as a natural regularizer; conservative retrieval schedules may be preferable when content quality is uncertain. These system-level patterns are amplified at lower solver capability (Appendix D), where even SIMPLE_RAG becomes surprisingly effective in dynamic mode, complementing LifelongAgentBench's observation that experience replay produces dramatic gains for smaller models (Zheng et al., 2025).

7.4 BUDGET EFFECTS

Increasing the retrieval budget from 400 to 800 tokens produces *non-monotonic* effects: some conditions improve (EXPEL C_ONLY: 63.3% $\rightarrow$ 70.0%) while others degrade (NO_MEMORY C_ONLY:

$67.7\% \rightarrow 63.3\%$). This non-monotonicity suggests that more retrieved content can introduce noise, particularly when cross-domain memories contain irrelevant or misleading information, consistent with LifelongAgentBench's finding that excessive experience replay degrades performance (Zheng et al., 2025).

8 DISCUSSION

Architecture vs. content: transfer value is capability-dependent. Our factorial design reveals that the relative value of architecture and content transfer depends on solver capability. For the weaker solver, dynamic conditions (architecture transfer) substantially outperform static conditions (content transfer), aligning with the intuition that *how* to learn matters more than *what* was learned. For the stronger solver, the picture is more nuanced: high-quality distilled content (EXPEL C_ONLY at 70.0%) can match or exceed dynamic learning, suggesting that the architecture-over-content advantage diminishes as intrinsic capability increases. This capability dependence is consistent with AWM's demonstration that procedural abstractions generalize across websites and task types (Wang et al., 2024), and with CER's success in transferring experience replay strategies across web environments (Liu et al., 2025), though both operate within narrower domain shifts than code-to-math.

Memory as capability augmentation. The capability-moderation finding (Section 7; Appendix D) suggests a practical framing: memory systems function as *capability augmentation* for weaker models. A 3B solver with a well-configured memory system ($\sim$52%) narrows the gap with a memoryless 7B model ($\sim$64%), recovering roughly half the performance difference through memory alone.

Negative transfer is real. Several conditions show accuracy *below* the no-memory baseline, concentrated in static mode with full content import (see Appendix D for the most pronounced cases). Agent KB addresses a related concern with its disagreement gate (Tang et al., 2025), and SEDM's marginal-utility admission criterion (Xu et al., 2025a) offers another approach to preventing harmful content from entering the store. Such filtering mechanisms become especially important under transplantation, where the agent has no prior interaction history to calibrate retrieval quality.

Limitations. Our study evaluates a single domain shift (code $\rightarrow$ math) with one solver per capability level. The 100-problem evaluation set and 3-seed replication limit statistical power for detecting smaller effects; most 7B effect sizes (2–5pp) fall within bootstrap confidence intervals that span 2–8pp, meaning that individual pairwise comparisons should be interpreted as directional rather than definitive. The NO_MEMORY baseline ($\sim$64% for 7B) falls below published Qwen 2.5 7B benchmarks ($\sim$75% with chain-of-thought prompting), likely because our domain-neutral prompt template, required by the prompt-freeze rule for internal validity, omits the CoT-specific instructions used in official evaluations. The "headroom" argument should therefore be understood relative to our controlled baseline, not to the model's best-case capability; the prompt-freeze is necessary for clean factorial attribution but limits external validity of absolute accuracy levels. Memory systems are documented approximations of their published designs rather than exact reproductions. In particular, we implement the core retrieval and write logic for each system (e.g., AGENT_KB's two-stage reranking and disagreement gating, EXPEL's dual insight/trajectory extraction) but omit components not relevant to the transplant protocol (e.g., AGENT_KB's full task-planning pipeline). System-specific findings (e.g., "AGENT_KB shows the strongest dynamic gains") should be interpreted as properties of our implementations rather than definitive statements about the published systems. The pre-specified negative controls (random retrieval, placebo, write-only, frozen-store MU) were run for representative conditions but are reported in supplementary analysis rather than the main text; surfacing these results would strengthen the attribution claims. The 3B experiment has partial coverage (156/210 runs), though all 70 condition cells are populated. Future work should expand to additional domain pairs, larger evaluation sets, and explicit study of which architectural features transfer most effectively.

9 CONCLUSION

We introduced a memory transplant protocol that cleanly separates architecture and content transfer for memory-augmented LLM agents. Across five memory systems evaluated on a code-to-math transfer task at two solver scales, we find that architecture transfer is *system-dependent*: code-configured mechanisms sometimes help and sometimes hurt math performance, with no universal direction. Content transfer in static mode provides *limited benefit*, with most gains coming from dynamic (online learning) conditions rather than imported knowledge. Solver capability *moderates transfer magnitude*: weaker models show larger memory effects ($+15$pp vs. $+7$pp), suggesting memory transplantation is most valuable where intrinsic capability is limited. Negative transfer is *real and common*: several architecture-content combinations perform below the no-memory baseline, highlighting the importance of controlled evaluation. Our pre-registered validation gates, required negative controls, and canonical export/import protocol provide a reusable framework for studying memory transfer in LLM agents.

REFERENCES

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Matthew Ho, Chen Si, Zhaoxiang Feng, Fangxu Yu, Yichi Yang, Zhijian Liu, Zhiting Hu, and Lianhui Qin. Arcmemo: Abstract reasoning composition with lifelong llm memory. *arXiv preprint arXiv:2509.04439*, 2025.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

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

Yitao Liu, Chenglei Si, Karthik R Narasimhan, and Shunyu Yao. Contextual experience replay for self-improvement of language agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14179–14198, 2025.

Bodhisattwa Prasad Majumder, Bhavana Dalvi Mishra, Peter Jansen, Oyvind Tafjord, Niket Tandon, Li Zhang, Chris Callison-Burch, and Peter Clark. Clin: A continually learning language agent for rapid task adaptation and generalization. *arXiv preprint arXiv:2310.10134*, 2023.

Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. Memgpt: Towards llms as operating systems. 2023.

Joon Sung Park, Joseph 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*, pp. 1–22, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

Xiangru Tang, Tianrui Qin, Tianhao Peng, Ziyang Zhou, Daniel Shao, Tingting Du, Xinming Wei, Peng Xia, Fang Wu, He Zhu, et al. Agent kb: Leveraging cross-domain experience for agentic problem solving. *arXiv preprint arXiv:2507.06229*, 2025.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H Chi, et al. Evo-memory: Benchmarking llm agent test-time learning with self-evolving memory. *arXiv preprint arXiv:2511.20857*, 2025.

Cheng-Kuang Wu, Zhi R Tam, Chieh-Yen Lin, Yun-Nung Chen, and Hung-yi Lee. Streambench: Towards benchmarking continuous improvement of language agents. *Advances in Neural Information Processing Systems*, 37:107039–107063, 2024.

Haoran Xu, Jiacong Hu, Ke Zhang, Lei Yu, Yuxin Tang, Xinyuan Song, Yiqun Duan, Lynn Ai, and Bill Shi. Sedm: Scalable self-evolving distributed memory for agents. *arXiv preprint arXiv:2509.09498*, 2025a.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*, 2025b.

Wei Yang, Jinwei Xiao, Hongming Zhang, Qingyang Zhang, Yanna Wang, and Bo Xu. Coarse-to-fine grounded memory for llm agent planning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 13040–13067, 2025.

Guibin Zhang, Muxin Fu, and Shuicheng Yan. Memgen: Weaving generative latent memory for self-evolving agents. *arXiv preprint arXiv:2509.24704*, 2025a.

Guibin Zhang, Haotian Ren, Chong Zhan, Zhenhong Zhou, Junhao Wang, He Zhu, Wangchunshu Zhou, and Shuicheng Yan. Memevolve: Meta-evolution of agent memory systems. *arXiv preprint arXiv:2512.18746*, 2025b.

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*, 2026.

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.

Junhao Zheng, Xidi Cai, Qiuke Li, Duzhen Zhang, ZhongZhi Li, Yingying Zhang, Le Song, and Qianli Ma. Lifelongagentbench: Evaluating llm agents as lifelong learners. *arXiv preprint arXiv:2505.11942*, 2025.

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.

A Detailed Results and Confidence Intervals

95% bootstrap confidence intervals are computed over stream seeds for each condition cell. For cells with $n = 3$ seeds, CIs typically span 2–8 percentage points. Selected CIs for the 7B solver include AGENT_KB E_MATH at budget 400: 71.0% [70.0, 73.0], EXPEL C_ONLY at budget 800: 70.0% [68.0, 71.0], and NO_MEMORY NM at budget 400: 64.3% [62.0, 67.0]. Full CI tables for both solvers are available in the supplementary material.

B Memory System Implementation Details

All memory systems implement a common `MemoryProvider` interface with five operations: `provide(query, context, budget)` retrieves items within a token budget, `take_in(episode)` ingests an episode result into the store, `manage()` runs pruning, consolidation, and tier management, `export_items(path)` serializes surviving items as canonical JSONL, and `import_items(path)` loads canonical items and recomputes internal structures. The canonical export format is JSONL with one `CanonicalMemoryItem` per line, validated against a JSON schema before use. On import, items are loaded verbatim; architecture-dependent structures (embeddings, tier placements, retrieval indices) are recomputed by the receiving system without LLM-based transformation.

The LIGHTWEIGHT_MEMORY dual-tier architecture routes items based on success signal: successful episodes produce `strategic` items stored in long-term memory, while failed episodes

produce `error_trace` items in short-term memory. The retrieval frequency parameter $N = 3$ means that retrieval is skipped for 2 out of every 3 episodes, reducing context pollution from irrelevant memories. AGENT_KB's two-stage retrieval first generates a broad candidate set of $2k$ items (where k is the final desired count), then reranks based on a composite score: score $= w_s \cdot \text{success} + w_t \cdot \text{type\_match} + w_r \cdot \text{recency}$. Disagreement gating compares each candidate's implication against the solver's draft answer and filters conflicting items. EXPEL's dual-item creation produces both an insight (strategic takeaway) and a trajectory (detailed solution trace) per episode; the 65/35 budget split during retrieval reflects the assumption that insights are more transferable than specific trajectories.

C VALIDATION GATE DETAILS

G1: Split integrity. Code-build (60 problems), math-build (30 problems), and math-eval (100 problems) are verified to have zero overlap at the problem-ID level. All three pairwise intersections are empty.

G2: Tuning containment. A SHA-256 hash of the solver prompt template is logged with every run. All 210 runs within the 7B plan share a single hash, confirming the prompt-freeze rule.

G3: Grader validation. The math grader is validated against a curated set of known answer pairs, including edge cases (fractions vs. decimals, equivalent algebraic forms, LaTeX formatting variations).

G4: Order robustness. All dynamic conditions (E_MATH, E_CODE) include runs with seeds 0, 1, and 2, each producing a different random ordering of the 100 evaluation problems.

G5: Budget robustness. All conditions are evaluated at both 400 and 800 token retrieval budgets, enabling analysis of budget sensitivity.

G6: Frozen-store MU. Marginal utility of retrieval is computed with identical store snapshots: PROVIDE on vs. off, with TAKE-IN/MANAGE disabled in both arms. MU artifacts are present for LIGHTWEIGHT_MEMORY across key conditions and both budgets.

D LLAMA 3.2 3B (WEAK SOLVER) RESULTS

Tables 5 and 6 report results for the 3B solver (156/210 runs completed; all 70 condition cells populated with 2–3 runs per cell). Some cells have $n = 2$ runs due to transient API failures (marked with *). The NO_MEMORY baseline drops to $\sim$37%, creating substantially more headroom than the 7B solver. The largest effect is EXPEL E_MATH at budget 800: 52.0% (+15pp above the true no-memory baseline of 37.0%), and SIMPLE_RAG E_MATH at budget 800: 51.7%.

Table 5: Math accuracy (%), Llama 3.2 3B, retrieval budget **400 tokens**. *Cells with n=2 runs.

System	NM	E_M	E_C	C_O	FULL	I_D	CRS
NO_MEM	37.7	37.0	39.3	39.7	39.3	35.7	36.6
SIMPLE_RAG	34.8	42.4	39.4	36.7	36.0	**43.0**	42.0
LW_MEM	39.3	41.2	42.3	38.0	35.7	41.3	35.5
AGENT_KB	43.8*	42.7	41.8	35.6	31.0	28.3	42.5*
EXPEL	42.3	40.8	**44.0**	32.8	32.3	44.8	37.3

Key differences from the 7B results: (1) dynamic-static divergence is more pronounced, with the best dynamic condition (EXPEL E_MATH: 52.0%) exceeding the best static condition (EXPEL CROSS: 44.3%) by 7.7pp; (2) negative transfer is more severe, with AGENT_KB IN_DOM dropping to 28.3% and EXPEL FULL to 28.0%; (3) SIMPLE_RAG becomes surprisingly effective in dynamic mode (E_MATH at 800: 51.7%), suggesting that for weaker solvers the bottleneck is having *any* relevant context rather than sophisticated retrieval; (4) H3 (architecture $\times$ content interaction) shows stronger non-additive patterns, with EXPEL IN_DOM reaching 44.8% while FULL drops to 32.3%.

Table 6: Math accuracy (%), Llama 3.2 3B, retrieval budget **800 tokens**. *Cells with n=2 runs.

System	NM	E_M	E_C	C_O	FULL	I_D	CRS
NO_MEM	37.0	37.1	38.0	35.4	38.0	41.0	39.1
SIMPLE_RAG	44.0*	**51.7**	42.1	35.8	35.5	41.1	41.8
LW_MEM	44.0	35.4	44.4	39.2	40.6	42.0	40.7
AGENT_KB	40.4	42.1	44.1	31.1	33.6	36.3	31.8
EXPEL	40.7	**52.0**	37.4	34.3	28.0	41.0	44.3

E Evaluation Subset and Baseline Discussion

MATH subset composition. The 100-problem evaluation subset is drawn from MATH Levels 3–5 (medium to hard). The distribution skews toward higher difficulty: approximately 25% Level 3, 35% Level 4, and 40% Level 5. This deliberate choice ensures the task is non-trivial for both solvers, but it also explains the depressed NO_MEMORY baseline relative to published benchmarks. The published Qwen 2.5 7B result of $\sim$75% on the full MATH dataset includes Level 1–2 problems (where the model scores $>$90%), which raise the aggregate. Our Level 3–5 subset is harder than the full dataset by design.

Prompt-freeze effect on baselines. The prompt-freeze rule requires a single, domain-neutral prompt template across all conditions (verified by SHA-256 hashing in Gate G2). This template omits chain-of-thought instructions, few-shot examples, and domain-specific formatting guidance that boost performance in official evaluations. We estimate the combined effect of the difficulty skew and prompt freeze accounts for most of the $\sim$10pp gap between our NO_MEMORY baseline ($\sim$64%) and published results ($\sim$75%). Importantly, all conditions share the same baseline depression, so relative comparisons (the focus of our analysis) remain valid.

F Negative Control Summary

We ran four pre-specified negative controls for LIGHTWEIGHT_MEMORY across key conditions at both retrieval budgets. Full per-cell results are available in the supplementary artifacts; here we summarize the key findings.

Random retrieval. Replacing similarity-based retrieval with random items from the same store (matched for count and token budget) reduces accuracy by 1–3pp relative to the standard condition for the 7B solver, confirming that retrieval relevance provides a small but real benefit above random context injection.

Placebo context. Replacing retrieved memory items with neutral filler text at matched token count produces accuracy within 1pp of the NO_MEMORY baseline, confirming that gains from memory come from content rather than from the presence of additional context tokens alone.

Write-only control. Enabling TAKE-IN/MANAGE while forcing PROVIDE to return empty isolates the contribution of memory writing side-effects (e.g., reflection during write). Performance is within 1–2pp of NO_MEMORY, indicating that the primary mechanism of memory benefit is retrieval, not the act of writing.

Frozen-store marginal utility. Comparing PROVIDE on vs. off with identical frozen stores and TAKE-IN/MANAGE disabled in both arms directly measures retrieval utility. The marginal utility of retrieval from imported content averages +2–4pp for in-domain conditions and +0–2pp for cross-domain conditions, consistent with the limited content-transfer finding (H2).