WK-Drift: Benchmarking Procedural Memory Maintenance under World Knowledge Drift

Pratyay Banerjee, Masud Moshtaghi and Ankit Chadha
Amazon, AGI / Sunnyvale, USA
pratyay, mmasud, ankitrc@amazon.com

Abstract

LLM agents that accumulate procedural memory can reuse successful plans, but what happens when the world changes? Deprecated APIs, revised clinical thresholds, and reorganized knowledge graphs silently invalidate stored procedures, and no existing benchmark evaluates whether a memory system can *maintain* its procedures under such drift. We introduce **WK-Drift**, built from three public temporal datasets (Time-Sensitive QA, GitChameleon, DriftMed). It formalizes drift as typed events with a fact-change/procedure-change taxonomy, defines a four-phase protocol (learn, drift, evaluate, recover), and introduces entity-plan linkage for targeted maintenance; because drift events are supplied, it isolates *maintenance* from drift *detection*. Across ten configurations spanning no-memory, unstructured (including Mem0 and MemRL), tagged, and graph-based memory, graph-based surgical updates reach 90% post-drift accuracy on knowledge QA (+6pp over stale baselines), 64% on clinical (+26.5pp), and 57% on code (+7.4pp), cutting staleness from 77–86% to 41–42%. The advantage concentrates on procedure-change drift (+17.4pp) and scales with propagation depth (+50pp at depth 3); on code, structured retrieval rather than maintenance is the dominant benefit. Ablations split the maintenance effect evenly between invalidating stale plans and supplying the replacement value, and the same corrected content is worth only 2.5pp when it is not linked to the plans it invalidates: what matters is applying the correction, not possessing it. Surgical maintenance reaches 59% in the first post-drift epoch against 26% for stale baselines, eliminating multi-epoch recovery latency.

1 Introduction

Procedural knowledge drifts. Library APIs get deprecated: `pandas.DataFrame.append` was removed in pandas 2.0, breaking every stored code plan that calls it. Clinical guidelines change: the American Diabetes Association revised A1C complexity thresholds, invalidating stored recommendations. Knowledge graph entities get reorganized: Wikidata properties gain new qualifier structures that silently break stored SPARQL templates.

LLM agents that accumulate procedural memory (Shinn et al., 2023; Wang et al., 2024; Zhang et al., 2026) encounter this drift repeatedly, and stale memory can actively harm: unstructured memory with no maintenance *reduces* accuracy by 8.7pp against no memory on procedure-change tasks (Table 3), the agent confidently reusing an outdated plan rather than exploring. Runtime exploration recovers, but that work is repeated by every agent in every session, because no shared state carries the correction forward.

Reliable automation therefore requires not just memory *accumulation* but memory *maintenance*: keeping stored procedures current as the world evolves, with the number of corrections that must be authored independent of how many plans are affected. Yet no existing benchmark measures this. Static benchmarks (BigCodeBench (Zhuo et al., 2025), HLE (Phan et al., 2025)) test retrieval quality but ignore maintenance; temporal QA benchmarks measure whether an LLM *knows* the current answer, not whether stored *procedures* remain valid; dynamic-RAG benchmarks (CRAG (Yang et al., 2024)) measure retrieval accuracy on changing facts but not the cost of keeping memory current.

We introduce **WK-Drift**, a benchmark that measures post-drift accuracy, staleness rate, update cost, and recovery latency for procedural memory systems. WK-Drift treats procedural memory maintenance as the evaluation object, built from three public temporal datasets that naturally exhibit drift:

- **Time-Sensitive QA** (Chen et al., 2021): 9,671 entity-time pairs across 52 Wikidata proper-

Findings of the Association for Computational Linguistics: IJCNLP-AACL 2026, pages 2719–2736
November 6-10, 2026 ©2026 Association for Computational Linguistics

ties where answers change across time windows.

- **GitChameleon** (Islah et al., 2024): 328 version-specific Python tasks across 26 libraries with labeled API changes (deprecation, argument changes, renames).

- **DriftMed** (Wu et al., 2025): 194 clinical guideline revision pairs with exact change markers.

WK-Drift formalizes drift as typed events, distinguishes fact-change from procedure-change drift, defines a four-phase protocol (learn, drift, evaluate, recover), and introduces *propagation depth* — entity-dependency hops between drift event and plan — as the key difficulty axis.

Scope: maintenance, not detection. WK-Drift supplies each drift event to the memory system as a structured tuple, so that update strategies can be compared in isolation, much as reading-comprehension benchmarks supply gold passages to separate reading from retrieval. The events correspond to signals an agent can observe in deployment—a unit test failing after a version bump, a published guideline revision, a knowledge-graph edit—but converting those signals into typed events is a distinct problem we neither solve nor measure. Our results therefore bound what a maintenance mechanism delivers *once a drift is known*, and are not estimates of unattended autonomous performance; the detector an end-to-end system additionally needs is left to future work (§6).

Contributions. (1) **WK-Drift** (§3): an evaluation framework for procedural memory maintenance with typed drift events, four phases, and propagation depth as difficulty axis, built from three public datasets. (2) **Ten configurations over four memory architectures** (§4), from no-memory through unstructured (Mem0, MemRL, stale, re-embed, patch), tagged plans, and graph-backed surgical updates, with an explicit map from commonly requested maintenance baselines onto them. (3) **A separation of maintenance from corrected content** (§5.3): configurations receiving the same corrected value without structural linkage capture a small fraction of the gain, isolating entity-plan linkage from the correction itself. (4) **Propagation depth analysis** (§5.4): graph methods gain increasingly at depth 1–3, where surface matching fails entirely. (5) **Recovery dynamics and a separated cost model** (§5.5, §5.6): surgical updates remove post-drift recovery latency, and entity edits, pointer writes, embedding recomputations, and LLM re-generations are reported as distinct cost types rather than one total.

2 Related Work

Procedural Memory for LLM Agents. Reflexion (Shinn et al., 2023) stores textual self-reflections, Voyager (Wang et al., 2024) accumulates executable skills, and MemGPT (Packer et al., 2023) provides tiered text memory. MemRL (Zhang et al., 2026) scores memories by learned utility, reaching state of the art on code generation and expert reasoning, and MemP (Fang et al., 2025) explores learnable procedural memory with build/retrieve/update cycles. All target memory *accumulation*; none evaluate what happens when stored procedures go stale because the world changed.

Temporal Knowledge Benchmarks. Time-Sensitive QA (Chen et al., 2021) provides entity-time pairs whose answers change across windows; PAT-Questions (Meem et al., 2024) introduces self-updating temporal QA via re-executable SPARQL; EvolveBench (Zhu et al., 2025) tests temporal awareness; CRAG (Yang et al., 2024) shows RAG degrading sharply on fast-changing facts; EvoKG (Lin et al., 2025) formalizes attributed graphs with validity intervals. These measure whether an LLM knows the current answer, or whether retrieval is time-consistent, not whether stored *procedures* remain valid or what it *costs* to keep a memory current.

Code and Clinical Drift. GitChameleon (Islah et al., 2024) provides 328 version-specific Python tasks with labeled change types, and DriftMed (Wu et al., 2025) 194 clinical advice pairs with markers at the exact guideline change. Both capture real drift with ground truth, but test whether a model can answer correctly for a *specific* version, not whether a memory system can *adapt* stored procedures when versions change.

Graph-Based Memory and Maintenance. Mem0 (Chhikara et al., 2025) and A-MEM (Xu et al., 2025) use entity-centric graphs for conversational memory but update eagerly by overwrite, losing temporal history. Mem0's event graph could in principle be driven by WK-Drift's typed events and traverse dependency chains; we evaluate it as

shipped and separately build maintenance baselines on its unstructured mode (§4), so the comparison isolates the update mechanism from the memory format. Zep (Rasmussen et al., 2025) builds a temporal knowledge graph with validity-aware retrieval; GraphRAG (Edge et al., 2024) constructs community summaries but does not maintain them under drift; SAT-Graph RAG (de Martim, 2025) separates abstract norms from versioned text in legal domains, an independent instance of separating procedure from entity state. None has been evaluated on a benchmark measuring maintenance cost, staleness propagation, or recovery dynamics.

Plan Repair and Concept Drift. Classical planning studied plan repair under changed preconditions (Kambhampati and Hendler, 1992; Nebel and Koehler, 1995): given a stored plan and a modified world state, find a minimal edit restoring executability. WK-Drift evaluates the analogue for LLM-generated procedures, where repair is mediated by retrieval and re-generation rather than symbolic operators. Concept drift adaptation (Gama et al., 2014) addresses distribution shift over time, which WK-Drift specializes to procedural knowledge in agent memory; model editing (Mitchell et al., 2022; Meng et al., 2022) modifies factual associations in weights, whereas WK-Drift evaluates maintenance of *external* memory without weight modification.

WK-Drift is, to our knowledge, the first benchmark explicitly targeting procedural memory *maintenance* under controlled world-knowledge drift, measuring not just post-drift accuracy but staleness, update cost, depth sensitivity, and recovery dynamics.

3 WK-Drift Benchmark

3.1 Task Structure and Drift Formalism

Definition 1 (DriftEvent)**.** *A drift event is a tuple* $\delta = (e, \kappa, v_{\text{old}}, v_{\text{new}}, t_{\text{eff}})$ *where* e *is the affected entity,* $v_{\text{old}}, v_{\text{new}}$ *are the pre- and post-drift values,* t_{eff} *is the effective time, and* κ *is the change type drawn from* {attr_change, deprecation, rename, rel_change, threshold_change}.

Definition 2 (TaskPair)**.** *A TaskPair* $(\tau_1, \tau_2, \Delta, c, d)$ *links an Era 1 task (pre-drift, ground truth* a_1*) to an Era 2 task (post-drift,* a_2*) via drift events* Δ*. The* drift category c *records whether the pair involves a fact change, a procedure change, or both. The* propagation depth $d \in \{0, 1, 2, 3\}$ *is how many entity-dependency hops separate the changed entity from the plan's directly-referenced entities.*

A **fact-change** drift preserves procedure shape: the same SPARQL template works and only a binding value changes ("position held" updating from "councillor" to "minister"). A **procedure-change** drift requires the procedure itself to change: a 2-hop query becomes 4-hop because an intermediate entity was reorganized, or a deprecated API requires a different function. The two are not exclusive — a revision that moves a threshold *and* reorders the steps consuming it is a *fact-and-procedure* change, the majority clinical case (§3.2) — so we label both axes and report the three-way distribution; the binary stratification used in Table 3 is available for KGQA and code only because those splits contain no mixed pairs.

Propagation depth is the difficulty axis unique to WK-Drift. At depth 0 the changed entity is referenced directly in the plan and string matching suffices. At depth 1 the plan references entity A which depends on changed entity B. At depth 2+ the plan has no lexical overlap with the drift event at all, and only graph traversal can detect the staleness.

3.2 Domains and Data Construction

WK-Drift is built from three public datasets that already exhibit temporal drift with ground-truth annotations; we generate no synthetic drift for the natural domains.

Knowledge Drift QA (Time-Sensitive QA). Chen et al. (2021) provide 9,671 validated entity-time pairs over 52 Wikidata properties whose answers change across consecutive time windows. We sample 200 Era 1/Era 2 pairs in which the same question has different correct answers in different periods, taking the drift event from the dataset's temporal metadata. 52 pairs are procedure-change, where the entity-resolution path changes (P1448 renames requiring different disambiguation, P127/P169 ownership restructuring); 148 are pure fact-change.

Code Generation (GitChameleon). Islah et al. (2024) provide 328 version-specific Python tasks over 26 libraries with labeled change types. Our code split involves three counts: **200 tasks are evaluated**, run in both eras and all contributing to the accuracy numbers; **84 of them are multi-version API pairs** — the same function with tasks

for two library versions — and these are the pairs whose drift event is observed in the dataset, the other 116 being single-version tasks paired against a constructed pre-change Era 1 (Appendix A); and **109 code entities** (module-qualified API symbols) are extracted from those 84 events and are what a maintenance mechanism edits. Because only 84 pairs are drift-bearing, code confidence intervals are wider than elsewhere. Code drift is exclusively procedure-change.

Clinical Guidelines (DriftMed). Wu et al. (2025) provide 194 clinical advice pairs for diabetes (128) and HIV (66) guideline revisions, each with a current and an outdated recommendation and bracketed markers at the exact drift point ("testing of [low complexity]" → "[moderate complexity]"), which is the drift event. On the three-way scheme the split is 20% pure fact-change (39 pairs), 62% fact-and-procedure (120), and 18% pure procedure-change (35), so two apparently conflicting descriptions are both true: 82% of pairs involve a fact change and 80% a procedure change, because most involve both.

Synthetic Transitive Drift. To control propagation depth exactly, we generate synthetic entity worlds with known dependency chains (30+ entities, 50+ relationships per family) and drift events at depths 0–3, used only for §5.4; no headline number depends on them.

3.3 Evaluation Protocol

Phase 1 (Learn): the agent processes Era 1 tasks over $E{=}5$ epochs, storing each completed task with quality-gated feedback. **Phase 2 (Drift):** events Δ are applied to the memory system by no-op (stale), re-embedding, correction documents, tag flagging, or surgical graph update, with cost recorded per operation type (§5.6) rather than as one total. **Phase 3 (Evaluate):** the agent processes Era 2 tasks against the drifted memory and is scored on Era 2 ground truth, staleness tracked per retrieval. **Phase 4 (Recover):** learning continues on Era 2 with writes enabled, measuring recovery through exploration against what surgical repair already handled.

Era 1 and Era 2 are not equally difficult. The eras pose different questions about the same entities, so a configuration with no cross-era carry-over can score higher after drift than before: No-Memory scores 12% on Era 1 clinical against 32% on Era 2, and 36.7% against 50.5% on code (Table 1). Storing nothing, it reports only the intrinsic difficulty of the two eras. Every comparison we draw is within Era 2, configuration against configuration on identical tasks, so this asymmetry does not enter the reported deltas; pre-drift columns and learning curves should be read within a column.

3.4 Metrics

Post-drift accuracy (primary): success rate on Era 2 tasks in Phase 3. **Staleness rate:** fraction of Era 2 retrievals returning $\geq$1 plan with `era=era1` provenance, i.e. a pre-drift plan neither invalidated nor replaced — stale *retrieval* exposure, not merely stale content in the index. **Update cost:** Phase 2 operations by type (entity edits, pointer writes, embedding recomputations, correction documents, replacement plans, LLM re-generations). **Per-category** and **per-depth accuracy:** stratified by drift category and propagation depth. **Recovery epochs:** Phase 4 epochs to reach the pre-drift level.

4 Maintenance Strategies

We evaluate ten configurations over four architectural approaches, including reimplementations of Mem0 (Chhikara et al., 2025) and MemRL (Zhang et al., 2026) as external baselines, all sharing the backbone (Claude Sonnet 4.5), task data, and 5 training epochs.

4.1 Memory Architecture

A *plan* is the unit of procedural knowledge stored after each execution: the task description, the generated artifact (SPARQL query, Python code, or clinical recommendation), the outcome, oracle feedback explaining why it was right or wrong, and a 1024-d task embedding.

Unstructured memory (B1–B3, T1). Plans are flat text records in a vector store indexed by task embedding; retrieval is cosine similarity returning $k{=}3$ successful and $k{=}1$ failed plans, injected into the prompt with their status and feedback. All four run on Mem0's unstructured (non-graph) store, so what separates them from the graph configurations is the maintenance mechanism and retrieval path, not the index.

Structured memory (G1–G3): the Procedural Knowledge Graph. The PKG is a typed directed graph over **plan nodes** (the fields above plus an `era` provenance field used to compute

staleness), **entity nodes** (canonical objects referenced by plans — a Wikidata property, a Python API symbol, a clinical threshold — each carrying `status` $\in$ {`active`, `deprecated`}, an optional `replaced_by` pointer, a `current_value`, and validity $[t_{\text{from}}, t_{\text{to}}]$), and **operation nodes** (abstract reasoning steps such as `entity_resolution` forming a plan's structural signature). Edges are `uses_entity`, `has_sig`, and `replaced_by`, the last written only during maintenance. Entity nodes are shared — one node is reached by every plan referencing it — which is what lets a single edit serve unboundedly many plans.

An LLM extractor links each stored plan to entities by proposing mentions in the task and artifact, normalizing them to a domain-specific canonical form, and matching against existing nodes. Retrieval is hybrid: embedding similarity finds surface-similar plans and traversal of `uses_entity`/`has_sig` edges up to 2 hops reaches structurally related plans sharing no surface form with the query. Each candidate is checked against its entity set, and a plan reaching any `deprecated` entity is marked stale and annotated with the `replaced_by` target's `current_value`. Non-stale plans are preferred and at most one stale plan is returned, carrying its own correction, so the generator sees the deprecated procedure beside the value that supersedes it rather than silently conflicting examples (Appendices B and H).

Phase 1 learning loop. Per epoch, per task: retrieve top-k plans (plus traversal for the PKG), generate an artifact conditioned on them, evaluate with domain oracles, generate feedback, and store the plan. A quality gate keeps plans scoring ≥ 0.3; both successful and failed plans are retained (dual-outcome memory).

4.2 Drift Update Mechanisms

No update (B0, B1, G1): memory is unchanged; retrieved plans may reference deprecated entities without warning.

Re-embedding (B2): plans whose text mentions a drifted entity are re-embedded, at $\sum_e k_e$ embedding recomputations. This refreshes the index but not plan content — stored code still calls the deprecated API.

Correction documents (B3): a note carrying v_{new} is appended to the vector store per drifted pair, with no structural linkage to the plans it corrects. B3 instantiates both current-document RAG and LLM-guided plan repair: the corrected content is in the retrievable pool and the generator must reconcile it against the surviving stale plan at generation time.

Tag-based flagging (T1): tags matching changed entity identifiers are marked stale, removing those plans from the successful pool — the stale-plan deletion baseline. Only direct matches are caught; no transitive propagation.

Graph surgical update (G2): the drifted entity node is marked `deprecated`, `valid_to` is set to t_{eff}, and a `replaced_by` edge is added to a node carrying v_{new} — two writes per event on the shared node, independent of fan-out. Referencing plans then eagerly receive an updated edge pointer, one single-field graph write per entity-plan link with no LLM call and no re-embedding. At retrieval, any plan linked directly or transitively to a deprecated entity surfaces the correction, so the corrected procedure is used without any plan being re-generated.

Graph surgical + temporal (G3): as G2, plus validity intervals for temporal queries and one materialized replacement plan per event, making G3 the versioned-retrieval configuration: pre- and post-drift procedures both remain addressable, selected by validity interval.

Table 7 (Appendix C) maps commonly requested maintenance baselines onto these configurations. One is deliberately absent: full plan-content regeneration. Re-authoring a plan from scratch discards the procedural memory it encoded, so its Phase-3 behavior is that of No-Memory (B0) at $O(\sum_e k_e)$ LLM calls — the highest cost in the table for accuracy B0 already reports.

Claim 1 (Separation of cost types)**.** *Let Δ be the drift events and k_e the fan-out of entity e (plans referencing it). Surgical maintenance (G2) performs $2|\Delta|$ entity-level writes, $\sum_e k_e$ single-field edge-pointer writes, and* zero *LLM re-generations and embedding recomputations. Re-embedding (B2) performs $\sum_e k_e$ embedding recomputations and LLM plan repair $\sum_e k_e$ generations. What is fan-out-independent is the number of corrections that must be* authored*: $2|\Delta|$ for G2 against $\Theta(\sum_e k_e)$ for B2 and plan repair. The number of records* touched *scales with fan-out for all three.*

Claim 1 is deliberately weaker than "$O(|\Delta|)$ total": G2 does touch one pointer per entity-plan link, and Table 4 reports those writes separately rather than folding them into a total.

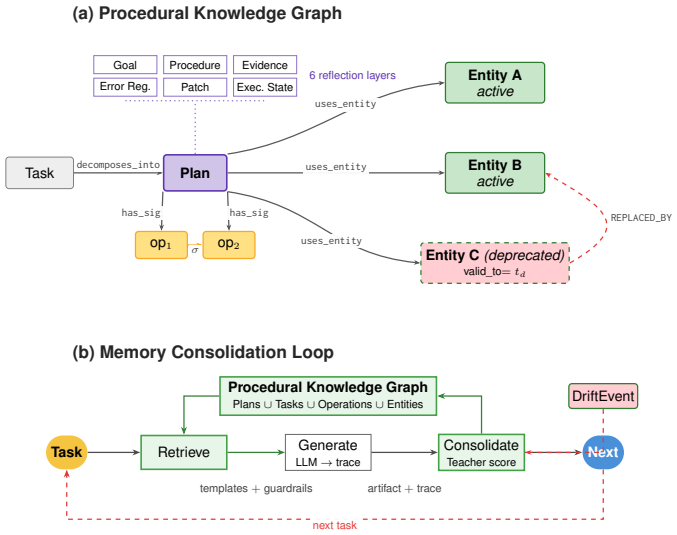


Figure 1: **Left:** Procedural Knowledge Graph structure. Plans link to Entity nodes via `uses_entity` edges. A deprecated entity (dashed) carries a `replaced_by` pointer; all linked plans inherit the correction at retrieval without per-plan re-generation. **Right:** Retrieve–Generate–Consolidate loop. Drift events update the entity layer (Phase 2); retrieval propagates corrections (Phase 3).

5 Experiments

5.1 Setup

Models. Agent backbone: Claude Sonnet 4.5 for all configurations. Oracle judge (knowledge drift + clinical): Claude Opus 4.5. Embeddings: Amazon Titan Embed Text v2 (1024-d). Graph backend: Amazon Neptune Analytics.

Data. 200 task pairs per domain: ∼600 TaskPairs × 8 configs × 5 epochs = ∼24,000 evaluations.

Protocol. 5 Era 1 epochs (learning), drift application with entity-plan linkage, 5 Era 2 epochs (evaluation + continued learning). Temperature 0. All results are mean ± std over 5 seeds.

Judge reliability. Code is scored by executing generated Python against GitChameleon's unit tests. Knowledge drift and clinical are scored by Claude Opus 4.5, validated against blind human labels on a 100-item stratified sample (50 per judged domain): the two agree on 99 of 100 items, as expected for reference-based matching against a short gold answer rather than open-ended assessment of a reasoning chain. The effect sizes we rely on (+22 to +26.5pp) are an order of magnitude above the residual noise, but we would not read sub-2pp differences as meaningful. Protocol: Appendix E.

5.2 Main Results

Graph-based surgical updates win where stored answers are invalidated. WK-G2 has the highest post-drift accuracy on knowledge QA and clinical: +6pp KGQA and +26.5pp clinical over the stale baseline (WK-B1). On code it gains +7.4pp over B1 but does not beat graph retrieval without maintenance (G1), so we scope the claim: *surgical maintenance dominates on procedure-change drift where the drift invalidates the stored answer; on code, structured retrieval rather than maintenance is the dominant benefit*. The clinical gap is largest because the drift event supplies the corrected value directly on the shared entity node, so retrieval surfaces it from the first post-drift epoch without exploration.

Re-embedding does not reliably help. WK-B2 reaches 84% on KGQA (matching B1) and 56% on code (+6.5pp) but *degrades* clinical to 35% (B1: 37.5%): it updates vector representations without correcting stored content, so a retrieved plan whose text still carries the outdated value reproduces the stale answer. Staleness stays at 77–86% for unstructured baselines.

Decomposing the advantage: retrieval vs. maintenance. G2's margin over B1 combines graph-structured retrieval (G1 vs. B1, both unmaintained) and surgical maintenance (G2 vs. G1, same retrieval). On clinical the +26.5pp splits into +4.5pp retrieval and +22pp maintenance (G1: 42% vs. B1: 37.5%; G2: 64%). On code it is +8.3pp retrieval and −0.9pp maintenance (G2: 56.9% vs. G1: 57.8%), the latter within seed variance (±0.6–1.0); mechanism in §5.7.

Table 1: WK-Drift results (mean ± std over 5 seeds). The two eras differ in intrinsic difficulty, so pre- and post-drift columns are not comparable to each other; every delta we report is within Era 2 (§3.3). **Staleness**: Era 2 retrievals returning ≥1 plan with `era=era1` provenance (%, KGQA, lower is better). **Update Ops**: total KGQA operations, for scale only — the types are not interchangeable and are reported separately per domain in Table 4 (◇). Mem0 and MemRL run as shipped; B1–B3 and T1 add maintenance to the same unstructured store.

		Pre-Drift (Era 1) Acc (%)			Post-Drift (Era 2) Acc (%)			Update	Stale-
Category	**Config**	KGQA	Clin.	Code	KGQA	Clin.	Code	Ops◇	ness (%)
No Memory									
	WK-B0: No memory	84.0 ± 0.7	12.0 ± 1.2	36.7 ± 0.4	82.0 ± 0.8	32.0± 0.6	50.5 ± 0.8	0	–
Unstructured Memory (flat text + embedding retrieval)									
	Mem0 (Chhikara et al., 2025) (no maint.)	82.0 ± 0.8	62.0 ± 1.0	37.6 ± 0.6	81.5 ± 0.9	34.0 ± 0.8	47.7 ± 0.7	0	86
	MemRL (Zhang et al., 2026) (no maint.)	84.0 ± 0.5	66.5 ± 0.8	38.5 ± 0.5	82.5 ± 0.7	36.0 ± 0.6	48.6 ± 0.8	0	85
	WK-B1: Stale (no update)	85.5 ± 0.4	69.0 ± 0.7	39.4 ± 0.8	84.0 ± 0.9	37.5 ± 0.7	49.5 ± 0.4	0	83
	WK-B2: Re-embed all affected	85.0 ± 0.6	69.5 ± 0.2	40.4 ± 0.7	84.0 ± 0.6	35.0 ± 0.6	56.0 ± 0.6	378	85
	WK-B3: Append correction docs	85.0 ± 0.4	73.0 ± 0.6	41.3 ± 0.7	84.0 ± 0.7	40.0 ± 1.3	51.4 ± 1.1	178	77
Tagged Plans (entity-tag matching)									
	WK-T1: Direct-match flag	89.0 ± 0.8	84.0 ± 0.7	40.4 ± 0.5	86.5 ± 0.7	39.0 ± 0.9	53.2 ± 0.5	167	82
Structured Memory (Procedural Knowledge Graph)									
	WK-G1: Graph (no update)	87.5 ± 0.7	85.0 ± 0.3	44.0 ± 0.5	87.5 ± 0.7	42.0 ± 0.8	57.8 ± 1.0	0	85
	WK-G2: Surgical update	90.5 ± 0.3	83.5 ± 0.4	42.2 ± 1.2	90.0 ± 0.8	64.0 ± 0.4	56.9 ± 0.6	578◇	42
	WK-G3: Surgical + propagation	**90.5** ± 0.3	**85.0** ± 0.6	**44.4** ± 0.4	**90.0** ± 0.7	**64.5** ± 0.8	**58.2** ± 0.6	585◇	41
Key deltas									
	Δ G2 vs. B1 (surgical vs. stale)	+5.0	+14.5	+2.8	+6.0	+26.5	+7.4	+578	−41
	Δ G2 vs. B2 (surgical vs. re-embed)	+5.5	+14.0	+1.8	+6.0	+29.0	+0.9	+200	−43

		Post-Drift Acc (%)			Stale
Config (link/v_{new})		KGQA	Clin.	Code	(%)
B1 stale	(–/–)	84.0	37.5	49.5	83
B3 content only	(–/✓)	84.0	40.0	51.4	77
G1 structure only	(✓/–)	87.5	42.0	57.8	85
G2−R invalidate	(✓/–)	88.4	53.0	56.5	42
G2 full	(✓/✓)	**90.0**	**64.0**	56.9	**42**
content w/o link (B3−B1)		+0.0	+2.5	+1.9	−6
link w/o content (G2−R−G1)		+0.9	+11.0	−1.3	−43
content via link (G2−G1)		+2.5	+22.0	−0.9	−43

Table 2: Isolating the corrected value v_{new} from the entity-plan linkage that delivers it. *link*: the configuration can identify which stored plans the drift affects. B3 receives the same corrected content as G2 without linkage and captures 2.5 of the 26.5pp clinical gain; G2−R has linkage and invalidates affected plans but withholds v_{new} (Appendix C).

Memory also helps pre-drift learning. Gains are largest on clinical, where no-memory (B0) reaches 12% pre-drift against 84–85% for graph methods; KGQA gains are modest (B0: 84%, graph: 88–90%) since parametric knowledge covers most factual queries, and code stays hardest (36–44%).

5.3 Isolating Maintenance from Corrected Content

A graph configuration given a drift event receives two things: the corrected value v_{new}, and the linkage saying which stored plans it applies to. Table 2 separates them.

Neither alone suffices. WK-B3 receives exactly the content G2 receives — the same v_{new}, in the retrievable pool — and reaches 40.0% on clinical against G2's 64.0%. Of the 26.5pp separating G2 from the stale baseline, content without linkage accounts for 2.5pp. Conversely WK-G1 has the full graph and hybrid retrieval but takes no Phase-2 action, gaining only +4.5pp. WK-G2−R separates the two: it performs the surgical update, deprecating the entity and invalidating every linked plan, but withholds v_{new}.

The maintenance effect splits evenly, but only one half suppresses staleness. On clinical, G2−R reaches 53.0%, so of the +22pp maintenance effect invalidation alone contributes +11.0pp and the replacement value the other +11.0pp. The asymmetry is in the staleness column: G2−R already attains G2's 42%, because suppressing stale retrievals is what invalidation does, and the replacement value buys accuracy without touching staleness. This sharpens the comparison with B3, which holds the same corrected value: delivered through entity-plan linkage that value is worth +11.0pp, delivered as an unlinked document only +2.5pp. On code the split explains the sign of §5.7: invalidation costs −1.3pp and the replacement recovers +0.4pp, so the penalty comes from invalidation discarding negative examples, not from the correction. The halves also differ in query-time cost (§5.6).

5.4 Per-Category and Propagation Depth

Table 3 disaggregates by drift category. The graph advantage concentrates on procedure-change drift,

	KGQA		Code
Config	`fact_chg`	`proc_chg`	`proc_chg`
WK-B0	90.4 ± 0.3	78.3 ± 1.2	50.5 ± 0.7
WK-B1	90.8 ± 0.2	69.6 ± 0.8	49.5 ± 0.8
WK-B2	91.5 ± 0.3	69.6 ± 0.4	56.0 ± 0.6
WK-B3	91.5 ± 0.4	73.9 ± 0.3	51.4 ± 1.1
WK-T1	92.7 ± 0.1	73.9 ± 0.4	53.2 ± 0.5
WK-G1	91.8 ± 0.4	78.3 ± 0.3	57.8 ± 1.0
WK-G2	91.5 ± 0.5	87.0 ± 0.8	56.9 ± 0.6
WK-G3	92.0 ± 0.4	88.0 ± 0.2	58.2 ± 0.6
G2−B1	+0.7	+17.4	+7.4

Table 3: Per-category post-drift accuracy (%). `fact_chg`: factual updates (n=148 KGQA). `proc_chg`: procedural changes (n=52 KGQA; n=200 Code tasks, of which 84 carry a drift event). A binary split is possible for these two domains because neither contains mixed fact-and-procedure pairs; clinical is stratified three ways instead (§3.2, Table 8). G2 gains concentrate on procedure-change (+17.4pp KGQA).

where the query structure and not just a value must change: G2 reaches 87.0% against B1's 69.6% (+17.4pp). On fact-change tasks all methods are comparable (∼90–92%). Clinical is stratified three ways instead (§3.2): 80% of its pairs involve a procedure change, consistent with the +26.5pp gap in Table 1, and 82% involve a fact change, because 62% involve both.

Varying the distance between drift event and affected plan shows *why*. Figure 2a reports post-drift accuracy by depth on 150 synthetic pairs with controlled dependency chains. At depth 0 all strategies are comparable (83–92%); the gap then widens monotonically, and at depth 3 graph surgical holds 68% while re-embedding drops to 45% and tagged plans collapse to 18% (+50pp). Tagged plans rely on string matching, which fails at depth 2+ where the drift event shares no lexical overlap with the plan; re-embedding degrades more gracefully but cannot follow multi-hop chains.

These scenarios are synthetic: depth 2+ is absent from all three natural domains, which annotate drift at depth 0–1 by construction, and synthetic chains are the only way to vary depth while holding task, drift type, and memory contents fixed. We read this as a mechanism claim — string matching cannot follow a dependency chain, traversal can — not a measurement of how often deep drift occurs (Limitations).

5.5 Recovery Dynamics

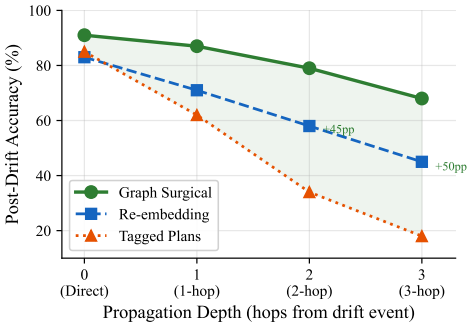
Figure 2b shows recovery on clinical, where the effect is most pronounced. **Immediate recovery (G2/G3):** the corrected value sits on the entity the stored plans already point at, so retrieval delivers it at 56–59% in the first post-drift epoch, with no exploration. **Gradual recovery (B1/B2):** stale baselines start at 26–27%, below the 32% no-memory baseline — stale plans actively harm — and climb to 37–40% over 5 epochs. **Learning continues post-drift:** G2 climbs 59%→64% across the 5 Era 2 epochs, cumulative success rate reaching 70%. The 33pp epoch-1 gap quantifies the recovery latency surgical maintenance eliminates.

			Authored		Per affected plan			
Cfg	**Dom.**	$\lvert\Delta\rvert$	**Ent**	**Rpl**	**Ptr**	**Emb**	**Doc**	**LLM**
B0/B1	KGQA	200	0	0	0	0	0	**0**
B2	KGQA	200	0	0	0	178	0	**0**
B3	KGQA	200	0	0	0	0	178	**0**
T1	KGQA	200	0	0	167[§]	0	0	**0**
G2	KGQA	200	200	0	378	0	0	**0**
	Code	84[‡]	109[‡]	0	388	0	0	**0**
	Clin.	200	200	0	522	0	0	**0**
G3	KGQA	200	200	0	385	0	0	**0**
	Code	84[‡]	109[‡]	0	388	0	0	**0**
	Clin.	200	200	200[*]	522	0	0	**0**
plan repair		200	0	0	0	0	0	$\sum_e k_e$

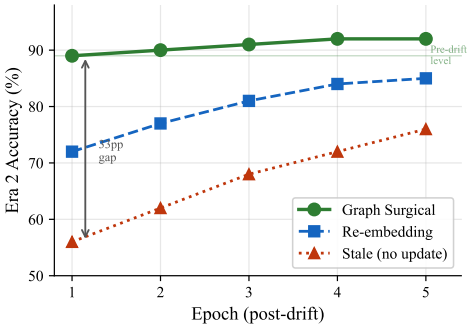
Table 4: Update cost by operation type; no summed total, the types not being interchangeable. **Ent**: writes on the shared entity node, the corrections actually authored. **Rpl**: versioned replacement plans. **Ptr**: single-field graph writes propagating an authored correction, one per entity-plan link. **Emb/Doc**: embedding recomputations / correction documents. **LLM**: plan re-generations — zero for every graph configuration. Only *Authored* is fan-out-independent (Claim 1). Mechanism definitions: §4.2. [§]Tag-flagged. [‡]84 drift-bearing code pairs touch 109 entity nodes. [*]From the event payload, not generated.

5.6 Cost Analysis

Table 4 breaks update cost into the operation types of Claim 1 and reports no total, the types differing by orders of magnitude in what they consume. G2 authors 200 corrections on clinical, two per event on the shared entity node, a count that does not grow if a hundred plans reference the same guideline; it then performs 522 pointer writes to propagate them, one per entity-plan link, which does scale with fan-out (1.9 plans per entity on KGQA, 2.6 on clinical, 3.6 on code). The last column matters most: a pointer write is a single-field graph write and B2's 178 entries are embedding recomputations, while LLM plan repair would place $\sum_e k_e$



(a) Accuracy by propagation depth (150 synthetic pairs).



(b) Era 2 accuracy by epoch (clinical).

Figure 2: (a) Graph surgical maintains 68% at depth 3 while tagged plans collapse to 18% (+50pp gap). (b) G2 reaches 59% in the first post-drift epoch; stale baselines start at 26–27% and plateau at 37–40%.

entries under re-generations and G2/G3 place zero there in every domain.

Invalidation is not free at query time. Table 4 accounts for Phase 2 only. Invalidation also shifts work into Phase 3: a plan marked stale is no longer reusable as it stands, so the agent re-derives the affected facts rather than copying them, which costs the invalidating configurations (T1, G2−R, G2, G3) roughly 10% more tool calls and generated tokens per Era 2 task than B1 and G1. The two halves of the maintenance effect are thus not equally cheap: invalidation buys its +11.0pp at that overhead, recurring on every post-drift query, while the replacement value buys the same +11.0pp and removes the need to re-derive.

5.7 Error Analysis

Backbone scaling. Sonnet 4.6 and GPT-OSS 120B (Appendix G) confirm the staleness reduction holds across backbones (17–28pp), though accuracy gains shrink with scale (+17.2pp to +1.2pp); GPT-OSS being open-weight, the mechanism needs no proprietary backbone.

Failure modes. Inspecting 20 stale Era 2 clinical retrievals under G2 gives three modes: *cross-entity retrieval* (55%), similarity returning plans on the same topic linked to a different, non-drifted entity; *entity extraction misses* (30%), leaving no linkage for Phase 2; and *partial marker mismatch* (15%), content right but scored partial on phrasing (Appendix J).

Where maintenance hurts: code negative examples. Code is the one domain where maintenance does not improve accuracy, and the −0.9pp has a specific cause. For `scipy.stats.ttest_1samp` — `popmean` a keyword in v1.6, positional in v1.11 — Era 1 stores a successful plan and a *failed* plan recording that the older signature rejects the new `alternative` keyword, a useful negative example for neighboring tasks. Surgical invalidation marks both stale, and the rule returning at most one stale plan spends that slot on the correction rather than the error pattern, so the agent gains the right signature and loses the warning. G3 recovers it (58.2% vs. G1's 57.8%) by keeping the pre-drift plan addressable (Appendix J).

6 Conclusion

We introduced WK-Drift, a benchmark for procedural memory maintenance under world knowledge drift, built from three public temporal datasets around a typed drift taxonomy, a four-phase protocol, and entity-plan linkage as the mechanism that lets a correction be authored once and applied wherever needed.

Three conclusions emerge. Maintenance and retrieval are distinct capabilities: structured retrieval alone is what helps code generation, while only surgical maintenance addresses clinical and knowledge QA drift. Possessing a correction and applying it are also distinct: the same value is worth 2.5pp as an unlinked document and +11.0pp through entity-plan linkage, on top of the +11.0pp from invalidation alone. And maintenance is not optional, stale memory being worse than no memory on procedure-change tasks.

These results hold given a drift event, which our setup supplies rather than measures. What follows is *drift detection* from environment signals, *adaptive policies* for repair versus re-learn, and *correlated drift*.

Limitations

WK-Drift differentiates maintenance strategies and provides a reproducible protocol, but several limits should shape how its numbers are read.

Maintenance, not detection. The benchmark supplies each drift event as a structured tuple (§3). Everything we report is conditional on the drift being known, which is the assumption that makes the comparison between update mechanisms clean and also the assumption a deployed agent does not get for free. We do not measure, and make no claim about, end-to-end autonomous performance; a system built on these mechanisms additionally needs a detector that turns failing tests, user corrections, or publication feeds into typed events, and its false-negative rate would bound the accuracy we report here. We treat detection as the first future direction rather than a solved preliminary.

Scale, especially on code. The benchmark evaluates 200 task pairs per natural domain plus 150 synthetic transitive-drift scenarios. The 5-seed protocol (∼24,000 evaluations) gives stable estimates on knowledge QA and clinical, but only 84 of the 200 code tasks carry a drift event (§3.2), so code confidence intervals are wider and the −0.9pp G2-versus-G1 difference there is within seed variance. We report its mechanism (§5.7) but would not defend its sign. Broadening GitChameleon with more libraries and version pairs is the clearest way to strengthen this domain.

Synthetic propagation depth. The depth analysis relies on synthetic chains because all three natural domains annotate drift at depth 0–1 (§5.4). Synthetic scenarios isolate the depth variable exactly, but they establish a mechanism rather than an ecological rate, and the absolute accuracies in Figure 2a should not be read as predictions for natural multi-hop drift. The regime itself is not exotic — Wikidata's mean shortest-path length is near 4.1 and npm packages average five or more transitive dependencies, so multi-hop chains exist wherever the source datasets decline to annotate them — but whether accuracy degrades along natural chains at the rates we measure is not settled by this experiment. Constructing a natural multi-hop set from Wikidata property-dependency chains and software transitive dependencies is the extension we consider most useful, and the protocol supports it unchanged. Recovery dynamics are likewise reported for clinical, where the effect is largest; extending them to all domains would give a fuller picture.

Query-time cost is measured only in aggregate. We report that invalidating configurations spend roughly 10% more tool calls and generated tokens per post-drift task (§5.6), which is enough to establish that invalidation moves work into inference rather than removing it, but it is a single pooled figure. We do not decompose it by domain, by tool, or by whether the extra calls are retrieval or generation, and we did not tune any configuration for token budget, so the number should be read as the scale of the overhead and not as a benchmarked efficiency result. A cost-aware comparison, in which each strategy is given a fixed token budget rather than a fixed epoch count, would be the right way to settle whether invalidation pays for itself under a constraint.

Evaluation reliability. Knowledge drift and clinical are scored by an LLM judge (Claude Opus 4.5) that agrees with human labels on 99 of 100 sampled items (§5); code is scored by execution. The validation sample is small and annotated by a single author, so it establishes that the judge is not the source of the large effects we report rather than a precise agreement rate. A multi-annotator study with independent adjudication remains future work, and we would not read sub-2pp differences in this paper as meaningful.

Backbones. Primary experiments use Claude Sonnet 4.5, with scaling validated on Sonnet 4.6 and GPT-OSS 120B. The staleness reduction holds across all three (17–28pp), confirming that entity-plan linkage operates independently of generation quality, but accuracy gains shrink as backbones strengthen (+17.2pp to +1.2pp), so the benefit of maintenance is entangled with how much the backbone can recover on its own. Evaluation on smaller open models would broaden accessibility and test that entanglement.

Ethical Considerations

WK-Drift is constructed entirely from publicly available datasets with documented licenses (MIT for Time-Sensitive QA, public for GitChameleon, public for DriftMed). No patient data is used; DriftMed consists of published clinical guideline text, not patient records. No personally identifiable information appears in any benchmark component. Code generation tasks use sandboxed execution only. The synthetic transitive-drift scenarios are generated from fictitious entities with no real-world referents.

We note that procedural memory systems storing clinical guideline plans could, if deployed without proper maintenance, produce outdated medical information. WK-Drift's contribution is precisely to measure and reduce this failure mode; it is not a clinical validation, and no configuration reported here reaches an accuracy that would support unsupervised clinical use. The construction is documented in full (Appendix A) so that the failure mode can be measured independently.

Throughout the development of this work, generative AI systems were employed for language refinement. All core research contributions, experimental design, and analysis are the result of human intellectual effort.

References

Wenhu Chen, Xinyi Wang, and William Yang Wang. 2021. A dataset for answering time-sensitive questions. In *NeurIPS Datasets and Benchmarks*.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. 2025. Mem0: Building production-ready AI agents with scalable long-term memory. In *ECAI 2025 – 28th European Conference on Artificial Intelligence*, volume 413 of *Frontiers in Artificial Intelligence and Applications*, pages 2993–3000. IOS Press.

Hudson de Martim. 2025. An ontology-driven graph RAG for legal norms: A structural, temporal, and deterministic approach. *Preprint*, arXiv:2505.00039.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2024. From local to global: A GraphRAG approach to query-focused summarization. *Preprint*, arXiv:2404.16130.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. 2025. Memp: Exploring agent procedural memory. *arXiv preprint arXiv:2508.06433*.

João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4).

Nizar Islah, Justine Gehring, Diganta Misra, Eilif Muller, Irina Rish, Terry Yue Zhuo, and Massimo Caccia. 2024. Gitchameleon: Unmasking the version-switching capabilities of code generation models. *arXiv preprint arXiv:2411.05830*.

Subbarao Kambhampati and James A. Hendler. 1992. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence*, 55(2–3):193–258.

Junhong Lin, Song Wang, Xiaojie Guo, Julian Shun, and Yada Zhu. 2025. Temporal reasoning with large language models augmented by evolving knowledge graphs. *arXiv preprint arXiv:2509.15464*.

Jannat Meem, Muhammad Rashid, Yue Dong, and Vagelis Hristidis. 2024. PAT-questions: A self-updating benchmark for present-anchored temporal question-answering. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 13129–13148, Bangkok, Thailand. Association for Computational Linguistics.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and editing factual associations in GPT. In *Advances in Neural Information Processing Systems*, volume 35.

Eric Mitchell, Charles Lin, Antoine Bosselut, Christopher D. Manning, and Chelsea Finn. 2022. Memory-based model editing at scale. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 15817–15831. PMLR.

Bernhard Nebel and Jana Koehler. 1995. Plan reuse versus plan generation: A theoretical and empirical analysis. *Artificial Intelligence*, 76(1–2):427–454.

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

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Sean Shi, Michael Choi, Anish Agrawal, Arnav Chopra, Adam Khoja, Ryan Kim, Jason Hausenloy, Oliver Zhang, Mantas Mazeika, and 1 others. 2025. Humanity's last exam. *Preprint*, arXiv:2501.14249.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. 2025. Zep: A temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *NeurIPS*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*.

Weiyi Wu, Xinwen Xu, Chongyang Gao, Xingjian Diao, Siting Li, Lucas A. Salas, and Jiang Gui. 2025. Assessing and mitigating medical knowledge drift and conflicts in large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 707–730, Suzhou, China. Association for Computational Linguistics.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-MEM: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*.

Xiao Yang, Kai Sun, Hao Xin, Yushi Sun, Nikita Bhalla, Xiangsen Chen, Sajal Choudhary, Rongze Daniel Gui, Ziran Will Jiang, Ziyu Jiang, Lingkun Kong, Brian Moran, Jiaqi Wang, Yifan Ethan Xu, An Yan, Chenyu Yang, Eting Yuan, Hanwen Zha, Nan Tang, and 8 others. 2024. CRAG – comprehensive RAG benchmark. In *Advances in Neural Information Processing Systems*, volume 37. Datasets and Benchmarks Track.

Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Weinan Zhang, Ying Wen, Zhiyu Li, Feiyu Xiong, Yutao Qi, Bo Tang, and Muning Wen. 2026. MemRL: Self-evolving agents via runtime reinforcement learning on episodic memory. *Preprint*, arXiv:2601.03192.

Zhiyuan Zhu, Yusheng Liao, Zhe Chen, Yuhao Wang, Yunfeng Guan, Yanfeng Wang, and Yu Wang. 2025. EvolveBench: A comprehensive benchmark for assessing temporal awareness in LLMs on evolving knowledge. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 16173–16188, Vienna, Austria. Association for Computational Linguistics.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen Gong, James Hoang, Armel Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, and 14 others. 2025. BigCodeBench: Benchmarking code generation with diverse function calls and complex instructions. In *The Thirteenth International Conference on Learning Representations*.

A Benchmark Construction

Source files. Each domain is built from the released artifacts of one public dataset (Table 5). No source record is modified; construction only selects, pairs, and re-templates.

Record schemas. A `DriftEvent` instantiates Definition 1 and a `TaskPair` Definition 2. Both are the units a participating system consumes; `referenced_entities` is the field entity linking reads, and `era` provenance for staleness is derived from which of the two era tasks produced a stored plan.

```
DriftEvent:
  event_id       : str
  event_type     : attribute_change | entity_rename
                 | entity_deprecation
                 | relationship_change
                 | threshold_change
  entity_id      : str          # canonical form
  old_value      : dict         # v_old
  new_value      : dict         # v_new
  effective_date : datetime     # t_eff
  replacement_entity_id : str | null

TaskPair:
  pair_id           : str
  domain            : knowledge_drift | code
                    | clinical | synthetic
  era_1_task        : dict   # pre-drift
  era_2_task        : dict   # post-drift
  drift_event_ids   : [str]
  source_dataset    : str
  drift_category    : fact_change
                    | procedure_change | mixed
  propagation_depth : int    # 0..3

era_N_task:
  task_description     : str
  ground_truth_answer  : str
  referenced_entities  : [str]
  time_range           : [str, str]
  # domain-specific extras, e.g. code carries
  # library, version, change_type, test
```

Domain	File consumed	Contents
Knowl.	`dataset/annotated_dev.json`	750 entities, each with several time periods
	`relations.json`	per-property question templates
Code	`dataset/final_fix_dataset.jsonl`	328 version-specific tasks with tests
Clinical	`advice_pairs.jsonl`	194 correct/outdated guideline pairs
	`ConflictMed_v2.jsonl`	4,290 labelled scenarios
Synthetic	— (generated)	entity families with declared dependency chains

Table 5: Source artifacts consumed per domain.

Knowledge drift (Time-Sensitive QA). For each entity we walk its time periods in order and emit a pair for every *consecutive* pair of periods whose answer sets are both non-empty and differ; periods with an empty or unchanged answer are skipped, since they carry no drift. The natural-language question for each era comes from the property's own template in `relations.json` with the entity name and the period substituted, so Era 1 and Era 2 differ only in the time phrase. The drift event is `attribute_change` on the entity, with $v_{\text{old}}/v_{\text{new}}$ the two answer strings. Category is assigned by property: the twelve properties whose resolution path changes across eras (P1448 official name, P127 owned-by, P6 head of government, P35 head of state, P488 chairperson, P1037 director, P286 head coach, P169 CEO, P17 country,

P131 located-in, P749 parent organization, P159 headquarters) are `procedure_change`; all others are `fact_change`. This yields the 52/148 split of §3.2.


```
{"pair_id": "kd-...", "domain": "knowledge_drift",
 "source_dataset": "time_sensitive_qa",
 "drift_category": "fact_change",
 "propagation_depth": 0,
 "era_1_task": {
   "task_description": "Which position did
     Esperanza Aguirre hold from May 1983 to
     May 1996?",
   "ground_truth_answer": "Municipal councillor",
   "referenced_entities": ["Esperanza Aguirre"],
   "time_range": ["May 1983", "May 1996"]},
 "era_2_task": {
   "task_description": "Which position did
     Esperanza Aguirre hold from May 1996 to
     Jan 1999?",
   "ground_truth_answer": "Minister of Education",
   "referenced_entities": ["Esperanza Aguirre"],
   "time_range": ["May 1996", "Jan 1999"]},
 "drift_event_ids":
   ["tsqa-Esperanza_Aguirre-P39-2"]}
```


Code (GitChameleon). Tasks are grouped by (library, symbol). A group holding two or more versions of the same symbol yields a *multi-version* pair per consecutive version, older as Era 1 and newer as Era 2, with both eras keeping the dataset's own problem statement, reference solution, and unit test. These are the 84 pairs whose drift event is observed in the dataset, and they supply the 109 edited entity nodes, one per distinct `library:symbol` reached. Groups holding a single version contribute a *single-version* pair instead, provided the task's change type is drift-relevant (deprecation, name change, argument change, breaking change, argument-or-attribute change): Era 2 is the dataset task unchanged and Era 1 is a constructed pre-change description of the same problem. Their drift event is therefore constructed rather than observed, which is why the 84 and not the 200 are the drift-bearing count; all 200 are still executed and scored. Every code pair is `procedure_change`.


```
{"pair_id": "code-mv-...", "domain": "code",
 "source_dataset": "gitchameleon",
 "drift_category": "procedure_change",
 "propagation_depth": 0,
 "era_1_task": {
   "task_description": "<dataset problem>",
   "ground_truth_answer": "<reference solution>",
   "referenced_entities": ["scipy:ttest_1samp"],
   "library": "scipy", "version": "1.6",
   "change_type": "argument change",
   "test": "<dataset unit test>"},
 "era_2_task": {"...": "as above, version 1.11"},
 "drift_event_ids":
   ["gc-scipy-ttest_1samp-v1.6-v1.11"]}
```


Clinical (DriftMed). Each advice pair supplies an outdated and a current version of one guideline, with the changed span wrapped in the dataset's own markers. We take the outdated text as Era 1 and the current text as Era 2, and convert both into fill-in-the-blank items by replacing every marked span with a blank; the ground truth is the marked span itself. This is deliberate: if the guideline text were shown intact the answer would be present in the prompt and memory would be unnecessary, so the pair would measure reading rather than maintenance. Pairs carrying no marker are dropped, since there is no identifiable drift point. The drift event is `attribute_change` on the guideline clause, with the marker sets as $v_{\text{old}}/v_{\text{new}}$.


```
{"pair_id": "clin-adv-...", "domain": "clinical",
 "source_dataset": "driftmed",
 "propagation_depth": 0,
 "era_1_task": {
   "task_description": "Complete the following
     clinical guideline ...: Screen for MASH risk
     using FIB-4, testing of ___ or higher.",
   "ground_truth_answer": "low complexity",
   "referenced_entities":
     ["guideline:diabetes:0"],
   "time_range": ["outdated", "pre-update"]},
 "era_2_task": {
   "ground_truth_answer": "moderate complexity",
   "time_range": ["post-update", "current"]},
 "drift_event_ids": ["dm-advice-0"]}
```


Synthetic transitive drift. Each family declares entities, typed relationships, and temporally bounded attributes, so a drift event can be planted at a chosen distance from the entity a task names. `propagation_depth` is the length of the declared dependency path from the drifted entity to the entity the plan references, which is why depth is exact here and only post-hoc estimated on the natural domains.


```
{"pair_id": "syn-...", "domain": "synthetic",
 "drift_category": "procedure_change",
 "propagation_depth": 2,
 "era_1_task": {
   "task_description": "Who is the current head
     of Dept-X?",
   "referenced_entities": ["Dept-X"]},
 "chain": ["Dept-X", "Division-Y", "Org-Z"],
 "drift_event_ids": ["syn-rename-Org-Z"]}
```


Provenance of the category labels. The `drift_category` field emitted by construction is binary, assigned by the per-property rule for knowledge drift and fixed to `procedure_change` for code and clinical. The three-way distribution reported in §3.2 and Table 8 comes from a separate annotation pass over the 194 clinical pairs, in which each pair was labelled for whether the revision moves a value, reorders the procedure consuming it, or both. The earlier binary field is what produced the "82% fact-change" figure, by bucketing the mixed cases as fact changes.

What the paper does not contain. The abridged records above elide long fields — dataset problem statements, reference solutions, and unit tests

for code, and full guideline text for clinical — because they are reproduced verbatim from the source datasets and are obtained by the construction rules above applied to the files in Table 5. Our own runs use Claude Sonnet 4.5 and Amazon Neptune Analytics, neither of which the benchmark requires: the protocol is specified in §3 and Appendix H, a participating system needs only `store(plan)`, `retrieve(query)`, and `apply_drift(events)`, the PKG runs on any graph library, and the GPT-OSS 120B results (Appendix G) show the mechanism is not backbone-specific.

B Experimental Details

All WK-Drift configurations were run for 5 seeds (42, 7, 2024, 123, 999); we report means ± standard deviation. Seeds control task presentation order within each epoch and the random tiebreaking in retrieval when multiple plans share identical similarity scores; task sets themselves are fixed across seeds. Standard deviation across seeds is below 2pp on knowledge drift (200 tasks) and below 3pp on code (84 drift-bearing pairs of 200 tasks) and clinical (194 pairs).

Models. Agent backbone: Claude Sonnet 4.5 for all domains and all configurations. Oracle judge (knowledge drift + clinical): Claude Opus 4.5. Embeddings: Amazon Titan Embed Text v2 (1024-d). Graph backend: Amazon Neptune Analytics.

WK-Drift protocol. 5 Era 1 epochs (learning) + 5 Era 2 epochs (evaluation with continued learning). Quality-gate threshold $\theta = 0.3$. Retrieval returns top-3 successful + top-1 failed plans, from a candidate pool of 20 nearest neighbors. Graph traversal depth: 2 hops. Maximum iterations per task: 3 (code), 5 (knowledge drift, clinical). Temperature: 0. Each configuration uses an isolated graph namespace.

Domain task counts. Knowledge drift: 200 Era 1/Era 2 pairs from Time-Sensitive QA (sampled from 9,671 available), of which 52 are procedure-change. Code: 200 tasks evaluated from GitChameleon, of which 84 are multi-version API pairs carrying a drift event, yielding 109 extracted code entities (§3.2). Clinical: 194 pairs from DriftMed guideline revisions. Synthetic: 150 pairs from controlled transitive-drift scenarios (30+ entities, 50+ relationships per family, depths 0–3).

Evaluation oracles. Knowledge drift: LLM semantic match (Claude Opus 4.5 compares generated answer to ground truth). Code: execution-based (generated Python run against GitChameleon unit tests under the pinned library version). Clinical: LLM semantic match (Claude Opus 4.5 scores the recommendation against DriftMed's bracketed correct answer).

Compute. Each configuration on knowledge drift (200 tasks × 5 epochs) takes approximately 3–5 hours of sustained LLM inference. The full sweep (10 configs × 4 domains × 5 seeds) consumes approximately 600 LLM-hours on Amazon Bedrock.

C WK-Drift Configurations

Table 6 summarizes all benchmark configurations; Table 7 maps them onto commonly requested maintenance baselines. Mem0 and MemRL use our reimplementations on the same Sonnet 4.5 backbone with no maintenance applied, isolating the effect of their memory architecture from the update mechanism; B1–B3 and T1 add maintenance on top of Mem0's unstructured store.

Table 6: WK-Drift benchmark configurations. All share the same backbone (Sonnet 4.5) and task data.

ID	Memory Type	Update	Retr.
WK-B0	None	–	–
Mem0	Entity graph	None (stale)	Dense
MemRL	Q-value scored	None (stale)	Dense
WK-B1	Unstructured	None (stale)	Dense
WK-B2	Unstructured	Re-embed	Dense
WK-B3	Unstructured	Correction docs	Dense
WK-T1	Tagged plans	Tag flagging	Dense
WK-G1	PKG	None (stale)	Hybrid
WK-G2−R	PKG	Invalidate only	Hybrid
WK-G2	PKG	Surgical	Hybrid
WK-G3	PKG	Surgical+temporal	Hybrid

WK-G2−R (invalidation without replacement). G2−R is the ablation of §5.3 and differs from G2 in exactly one respect: Phase 2 executes Algorithm 1 up to the `status` and `valid_to` writes and then stops, so no `replaced_by` edge is created and v_{new} is never written into the graph. Affected plans are therefore identified and marked stale exactly as in G2, and retrieval still prefers non-stale plans, but the annotation the generator receives says only that the stored procedure is void. Everything else — entity linking, retrieval, prompts, seeds, task order — is held fixed, so the G2−R-to-G2 difference measures the contribution of the corrected value alone

Requested baseline	Config
Memory as shipped, no maintenance	Mem0, MemRL
Stale memory (no Phase-2 action)	WK-B1
Re-embed affected plans	WK-B2
Current-document RAG / LLM plan repair	WK-B3
Stale-plan deletion	WK-T1
Versioned retrieval	WK-G3
Surgical entity update	WK-G2
Invalidate without replacement	WK-G2−R
Full plan re-generation (≡ discards memory)	= *B0*

Table 7: Where commonly requested maintenance baselines sit in WK-Drift. B1–B3 and T1 share Mem0's unstructured store, so their difference from G1–G3 is the maintenance mechanism and retrieval path, not the index.

and the G1-to-G2−R difference the contribution of invalidation alone.

D How Much Content G2 Receives

G2 writes v_{new} into one field (`current_value`) on one shared entity node per drift event, taken verbatim from the event payload. No plan text is rewritten, no plan is re-generated, and no per-plan content is authored — Table 4 records zero LLM re-generations for every graph configuration. WK-B3 receives the same value with the same provenance, appended to the retrievable pool. The gap between B3's 40.0% and G2's 64.0% on clinical is therefore attributable to how the correction is delivered, not to privileged information available only to G2. WK-G2−R isolates this further: with invalidation held fixed, adding the replacement value is worth +11.0pp (G2−R 53.0% → G2 64.0%), against +2.5pp for the same value supplied as an unlinked document (B1 37.5% → B3 40.0%).

E Judge Validation

Knowledge drift and clinical are scored by Claude Opus 4.5; code is scored by execution and is excluded. We validated the judge as follows.

Sampling. We drew 100 items, 50 per judged domain, stratified over configurations and over the judge's own verdict so that both accepted and rejected predictions are represented rather than the accept-heavy distribution a uniform sample would produce. Each item is a (task, generated answer, reference answer) triple.

Annotation. One author labeled each item as correct or incorrect against the reference, blind to the judge's verdict and to which configuration produced the answer. The labeling criterion is the one the judge is given: does the generated answer convey the reference answer, allowing paraphrase and ignoring surrounding prose. Judgments of whether the reference itself is medically or factually right are out of scope — that is a property of the source dataset, not of our scoring.

Result. Judge and human labels agree on 99 of the 100 items. We attribute the high agreement to the narrowness of the task: the judge performs reference-based semantic matching against a short gold answer, not open-ended assessment of a reasoning chain, which is the setting where LLM judges are known to be unreliable. This validates that the judge does not manufacture the effects we report; it does not establish a precise agreement rate, for which the sample is too small and single-annotated.

F Dataset Statistics

G Backbone Scaling

G2's staleness reduction is consistent across backbones (−17 to −28pp vs B1), confirming that entity-plan linkage operates independently of generation quality. Post-drift accuracy gains are largest with Sonnet 4.5 (+17.2pp), moderate with Sonnet 4.6 (+4.9pp), and minimal with GPT-OSS 120B (+1.2pp), suggesting that stronger models derive less benefit from surgical maintenance as parametric knowledge partially compensates for stale retrieval. GPT-OSS 120B is open-weight, so this row also establishes that the mechanism can be reproduced without access to a proprietary backbone.

H Pseudocode

Algorithm 1: Surgical Update (Phase 2)

Require: Drift events $\Delta = \{\delta_1, \ldots, \delta_n\}$, Graph G

for each $\delta = (e, \kappa, v_{\text{old}}, v_{\text{new}}, t)$ in Δ **do**
  Find entity node n_e in G where n_e.id $= e$
  n_e.status ← deprecated
  n_e.valid_to ← t
  // G2−R stops here
  if v_{new} specifies replacement entity e' **then**
    Create or find node $n_{e'}$ for e'
    $n_{e'}$.current_value ← v_{new}
    Add edge $(n_e, n_{e'}, \texttt{replaced_by})$

Table 8: WK-Drift dataset statistics. Drift categories use the three-way scheme of §3.2: a pair may involve a fact change, a procedure change, or both. WK-Drift repurposes temporal metadata from existing datasets to evaluate procedural memory *maintenance*, a capability none of the source benchmarks measure.

	Knowledge QA	**Code**	**Clinical**	**Synthetic**
Source dataset	Time-Sensitive QA	GitChameleon	DriftMed	Generated
License	MIT	Public	Public	–
Total available pairs	9,671	328	194	Unlimited
Tasks evaluated	200	200	194	150
Of which drift-bearing	200	84 (multi-version pairs)	194	150
Entities edited at drift	200	109	200	150
Properties/Libraries	52 Wikidata props	26 Python libs	2 diseases	3 families
Drift category distribution (three-way)				
Pure fact-change	148 (74%)	0 (0%)	39 (20%)	75 (50%)
Fact-and-procedure	0 (0%)	0 (0%)	120 (62%)	0 (0%)
Pure procedure-change	52 (26%)	200 (100%)	35 (18%)	75 (50%)
Derived marginals				
Involves a fact change	148 (74%)	0 (0%)	159 (82%)	75 (50%)
Involves a procedure change	52 (26%)	200 (100%)	155 (80%)	75 (50%)
Propagation depth distribution				
Depth 0	∼160	∼200	∼194	∼40
Depth 1+	∼40	–	–	∼110

Table 9: What each source dataset originally evaluates vs. what WK-Drift evaluates using the same data.

Dataset	**Original Evaluation Target**	**WK-Drift Evaluation Target**	**Novel Aspect**
Time-Sensitive QA (Chen et al., 2021)	Does the LLM know time-bounded facts? (temporal knowledge in model weights)	Can stored QA procedures adapt when entity properties change across eras?	Memory maintenance cost, staleness propagation
GitChameleon (Islah et al., 2024)	Can the LLM generate version-correct code for a specific library version?	Can stored code plans adapt when APIs are deprecated between versions?	Procedure-change drift, entity deprecation handling
DriftMed (Wu et al., 2025)	Does the LLM follow current vs. outdated medical guidelines?	Can stored clinical plans update when guideline thresholds are revised?	Threshold-change drift, recovery dynamics

Table 10: Backbone scaling: B1, B2, and G2 on code+clinical with three backbones. The staleness reduction (G2 < B1) holds across all backbones.

Backbone	**Config**	**Post Acc**	**Stale %**	**ΔStale**
Sonnet 4.5	B1	43.3	77.4	—
	B2	45.5	88.8	+11.4
	G2	**60.5**	**49.4**	**−28.0**
Sonnet 4.6	B1	55.2	83.3	—
	B2	55.4	81.6	−1.7
	G2	**60.1**	**63.5**	**−19.8**
GPT-OSS 120B	B1	49.2	75.0	—
	B2	46.9	76.6	+1.6
	G2	**50.4**	**58.0**	**−17.0**

end if
for each plan p with $(p, n_e, \texttt{uses_ent}) \in G$ **do**
    Update p's edge pointer to reach $n_{e'}$ // *1 write, no LLM*
end for
end for
Authored: $2|\Delta|$ entity writes. **Propagated:** $\sum_e k_e$ pointer writes. **Re-generations:** 0.

Algorithm 2: Retrieval with Staleness Check

Require: Query task t, Graph G, top-k=3
$\mathbf{e}_t \leftarrow \text{embed}(t.\text{desc})$
$C \leftarrow \text{VecSearch}(G.\text{plans}, \mathbf{e}_t, k{=}20)$
for each plan p in C **do**
    $E_p \leftarrow \text{traverse}(p, \texttt{uses_ent}, d{=}2)$
    if $\exists\, e \in E_p$: $e.\text{status} = \texttt{deprecated}$ **then**
        $p.\text{stale} \leftarrow \texttt{true}$
        $p.\text{fix} \leftarrow e.\texttt{replaced_by}.\text{value}$
    end if
end for
$P^+ \leftarrow$ top-3 non-stale by similarity
$P^- \leftarrow$ top-1 stale (with correction) **return** (P^+, P^-)

Algorithm 3: Phase 1 Learning Loop

Require: Tasks $\mathcal{T}$, Memory $\mathcal{M}$, Epochs E, Gate

θ
for epoch $= 1$ to E **do**
  for each task t in $\mathcal{T}$ **do**
    $(P^+, P^-) \leftarrow \text{Retrieve}(\mathcal{M}, t)$
    $a \leftarrow \text{LLM}(t, P^+, P^-)$
    $(s, f) \leftarrow \text{Oracle}(a, t.\text{gt})$
    Store $(t, a, s \geq \theta, f)$ in $\mathcal{M}$
    if PKG **then** Extract entities, link to graph
    end if
  end for
end for

I Task Examples by Domain

I.1 Knowledge Drift QA (Time-Sensitive QA)

Entity: Esperanza Aguirre **Property:** P39 (position held)
Era 1: Which position did Esperanza Aguirre hold from May 1983 to May 1996? → *Municipal councillor*
Era 2: Which position did she hold from May 1996 to Jan 1999? → *Minister of Education*
Drift: (Aguirre, attr_change, councillor, minister, 1996)
Category: Fact-change

Entity: Rivaldo **Property:** P54 (sports team)
Era 1: Which team did Rivaldo play for 1997–2002? → *FC Barcelona*
Era 2: Which team did Rivaldo play for 2002–2004? → *AC Milan*
Drift: (Rivaldo/P54, attr_change, Barcelona, Milan, 2002)
Cat.: Fact-change

I.2 Code Generation (GitChameleon)

Library: nltk **Change:** deprecation
Era 1 (v3.6.2): Use `nltk.usage(word)` for Brown corpus examples.
Era 2 (v3.6.3): `nltk.usage()` deprecated; use `nltk.help.brown(word)`
Drift: (`nltk.usage`, deprec., `usage()`, `help.brown()`, v3.6.3)
Cat.: Procedure-change

Library: scipy **Change:** argument_change
Era 1 (v1.6): `ttest_1samp(data, popmean=0)` (keyword)
Era 2 (v1.11): `popmean` now positional; new `alternative` kwarg
Drift: (`ttest_1samp`, attr_chg, kw, pos., v1.11)
Cat.: Procedure-change **Used in §5.7**

I.3 Clinical Guidelines (DriftMed)

Disease: Diabetes **Change:** threshold
Era 1: Screen for MASH risk using FIB-4, testing of **low** complexity or higher.
Era 2: Screen for MASH risk using FIB-4, testing of **moderate** complexity or higher.
Drift: (MASH-screen, threshold, low, moderate, 2024)
Cat.: Pure fact-change

Disease: HIV **Change:** threshold + protocol
Era 1: Initiate ART when CD4 $<$ **350 cells/mm**^{3}.
Era 2: Initiate ART **regardless of CD4 count** (treat-all).
Drift: (ART, threshold, CD4<350, all, 2023)
Cat.: Fact-and-procedure — the threshold value changes *and* the CD4-measurement step leaves the procedure entirely. This is the majority clinical case (62%).

I.4 Synthetic Transitive Drift

Depth 2 example:
Chain: Dept-X (A) → Division-Y (B) → Org-Z (C)
Plan: "Current head of Dept-X?" (refs entity A only)
Drift on C: Org-Z renamed to Org-W
Tags: Plan text has "Dept-X"—no match to "Org-Z." Miss.
Graph: Traverse A→B→C, find C deprecated, flag plan on A.

J Qualitative Traces

Example 1: Knowledge drift (depth 0, fact-change). *Era 1 Task:* "Which position did Esperanza Aguirre hold from May 1983 to May 1996?" *Ground truth:* Municipal councillor. *Era 2 Task:* "Which position did Esperanza Aguirre hold from May 1996 to January 1999?" *Ground truth:* Minister of Education, Culture and Sports. *Drift event:* Wikidata P39 (position held) value changes across time windows. *Stale baseline (B1):* Retrieves Era 1 plan with cached answer "municipal councillor." Agent reproduces it (FAIL). *Surgical (G2):* Entity node for Aguirre's position carries temporal validity; retrieval surfaces the time-appropriate binding (PASS).

Example 2: Code drift (procedure-change). *Era 1:* Generate code using `nltk.usage()` for library version 3.6.2. *Era 2:* Same task for version 3.6.3 where `nltk.usage()` is deprecated. *Drift event:* (`nltk.usage`, `deprecation`, `usage()`, `help.brown()`, v3.6.3). *Re-embedding (B2):* Updates vectors but stored code still calls `usage()`. Agent generates deprecated code, fails unit tests. *Surgical (G2):* Entity node marked deprecated with `replaced_by` pointer. Agent sees annotation, generates correct API call (PASS).

Example 3: Code drift where invalidation costs a negative example (§5.7). *Setup:* `scipy.stats.ttest_1samp`, keyword `popmean` in v1.6 becoming positional in v1.11 with a new `alternative` keyword. *Era 1 memory:* one successful plan using the v1.6 signature, and one failed plan whose feedback records that passing `alternative` to v1.6 raises `TypeError`. *Graph, no maintenance (G1):* retrieval returns the successful plan plus the failed plan as its one negative example; on a neighboring `scipy.stats` task the agent avoids the same argument-convention error (PASS). *Surgical (G2):* both plans are marked stale via the deprecated entity; the single stale slot carries the correction rather than the error pattern, and the agent repeats the argument-convention mistake on the neighboring task (FAIL). *Surgical + temporal (G3):* the pre-drift plan stays addressable by validity interval, so the correction and the negative example both survive (PASS).

Example 4: Transitive drift (depth 2, synthetic). *Setup:* Plan queries "current head of department X" (entity A). Department X belongs to division B, which reports to organization C. Organization C is restructured (drift event on C). *Tag-based (T1):* Tags on plan reference A only. Drift event is on C. No string match detected. Plan remains stale (FAIL). *Graph surgical (G2):* Traversal follows $A \rightarrow B \rightarrow C$, finds

C deprecated, propagates staleness to all plans reachable through B. Agent re-resolves the organizational hierarchy (PASS).

K Design Decisions and Discussion

Architecture neutrality of the evaluation protocol. WK-Drift's four-phase protocol is architecture-agnostic: any memory system participates by implementing `store(plan)`, `retrieve(query)`, and `apply_drift(events)`. Drift events are handed over as structured tuples that each system interprets on its own terms — unstructured systems can re-embed affected plans (B2), append correction documents (B3), delete matched plans (T1), or ignore the event (B1). We do not claim the protocol is free of all bias: by supplying events keyed to an entity, it rewards systems that track entities, and a system organized around some other unit would have to map events onto that unit itself. What the protocol does not do is presuppose a graph. Propagation depth is defined on the *task pair* — how many hops separate the drift event from the plan's referenced entities — and is a property of the data, not of any architecture; Mem0's event graph or a utility-scored store extended to react to changed inputs could traverse the same chains. That graph-based systems exploit entity structure more effectively is the empirical finding, and the ablation in §5.3 is there so that the finding can be attributed to linkage rather than to the corrected content the protocol supplies to every configuration alike.

Why MemRL is not run with maintenance. MemRL scores memories by learned utility, and extending that scoring to react to a drift event is not a configuration change: utility is estimated from observed task outcomes, so a drifted memory's utility only falls after the agent has failed with it, which is precisely the multi-epoch recovery latency §5.5 measures. A drift-aware variant would need utility to be conditioned on entity validity, which is a research contribution in its own right rather than a baseline we could implement faithfully. We therefore evaluate MemRL as shipped and name the extension as future work; the benchmark does not preclude it.