A DECONFOUNDED MEMORY MECHANISM FOR MACHINE LEARNING ENGINEER AGENT SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

The memory mechanisms of large language model (LLM)-based agents enable cross-episode self-improvement, expanding their ability to automate the end-to-end process of machine learning (ML) research and development (R&D). However, existing ML engineer agents typically promote or demote hypotheses in memory based on the final outcome of the episode in which they are tested. Such outcome-based memory updates are highly confounded: a negative outcome may result from factors unrelated to the hypothesis itself, causing valid hypotheses to be incorrectly demoted and permanently excluded from subsequent research. We refer to genuinely valid hypotheses that receive a confounded negative outcome in an R&D episode as false-negative hypotheses. In this paper, we propose DEC-MEM, a principled memory mechanism for deconfounding episode outcomes. When an R&D episode yields a negative outcome, DEC-MEM first counterfactually probes the episode to identify the causes of the outcome, collecting thorough evidence within a minimal computational budget. We then employ a write policy that stores each hypothesis together with the attributed causes of the outcome and supporting evidence to guide future R&D episodes. During memory retrieval in future episodes, we rank stored memory items using a novel metric that uncovers items associated with confounded negative outcomes, giving their hypotheses an opportunity to be re-examined rather than permanently discarded. In addition, we propose novel evaluation metrics to quantify how well a memory mechanism identifies false-negative hypotheses and re-examines them in later episodes. Extensive experiments across R&D streams show that DEC-MEM identifies false-negative hypotheses more effectively than state-of-the-art baselines and improves cumulative R&D performance by re-examining these hypotheses. Our code is available at `https://anonymous.4open.science/r/DEC-MEM-2D12`.

1 INTRODUCTION

Recent advances in large language models (LLMs) (Naveed et al., 2025; Zhao et al., 2026) have enabled LLM-based agents to act as machine learning (ML) engineers (Takallou et al., 2026; Gu et al., 2026; Jiang et al., 2025) and automate the process of ML research and development (R&D). Given an ML task, these agents can conduct a sequence of R&D episodes, each consisting of hypothesis proposal, code implementation and execution, and performance evaluation (Chan et al., 2025; Wijk et al., 2025; Starace et al., 2025). To improve R&D performance across episodes, ML engineer agents maintain a memory of previous hypotheses and experimental outcomes, from which relevant experiences are retrieved to guide subsequent research (Zhao et al., 2024; Liu et al., 2025).

However, existing memory mechanisms typically promote or demote a hypothesis based on the final outcome of the episode in which it is tested (Shinn et al., 2023; Zhao et al., 2024; Jiang et al., 2025; Liu et al., 2025). Such outcome-based updates implicitly assume that the performance of an episode reliably reflects hypothesis validity. In fact, a negative episode outcome can be confounded: it may arise from an invalid hypothesis, but also from a semantically incorrect implementation, an insufficient compute budget, or an unsuitable evaluation protocol (Huang et al., 2023; Wijk et al., 2025; Starace et al., 2025; Kon et al., 2026). Consequently, a genuinely valid hypothesis may be incorrectly demoted after one negative outcome and never re-examined. We refer to such hypotheses as *false-negative hypotheses*, and their exclusion from subsequent episodes as *false-negative lock-in*.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Here, we provide an illustrative example, shown in Fig. 1, based on a representative ML engineer agent system named AIDE (Jiang et al., 2025). Generally, AIDE treats an R&D stream as building a tree, where each node is an episode; the hypothesis for the next episode is proposed based on the hypothesis associated with the best-performing leaf node. Consider an ML task for which AIDE explores two candidate solutions. Candidate s_1 instantiates the hypothesis that a neural network is suitable for the task, while s_2 adopts a decision tree. Although the neural-network hypothesis is promising, an insufficient training budget prevents its optimization from converging, causing s_1 to underperform s_2. Since AIDE expands its search from better-performing candidates, it proceeds from s_2 to produce s_3, while the hypothesis underlying s_1 is unlikely to be reconsidered (Jiang et al., 2025). The negative outcome of s_1, however, does not imply that its hypothesis is invalid. A desired memory mechanism should deconfound the negative outcome of s_1 by attributing its underperformance to the insufficient training budget, then re-examine the neural-network hypothesis with a larger budget. This produces $\hat{s}_3$, which substantially outperforms s_3. Despite its urgency, addressing false-negative lock-in is non-trivial due to the following three challenges. First, the **Evidence Attribution Gap**: execution logs of the R&D episode alone are insufficient for determining why an episode fails, since failures caused by silent code bugs, insufficient training, or inappropriate evaluation may not be clearly revealed in logs (Starace et al., 2025; Kon et al., 2026). Second, the **Writing Policy Gap**: existing memory mechanisms rarely preserve the real causes and supporting evidence associated with unsuccessful episodes (Zhang et al., 2024; Jiang et al., 2025; Liu et al., 2025). Even if a hypothesis is later re-examined, the agent may repeat the same mistake because the cause of its previous failure was not recorded. Third, the **Retrieval Policy Gap**: existing agents retrieve hypotheses according to relevance and effectiveness, while neglecting false-negative hypotheses which can potentially provide significant gains (Park et al., 2023; Jiang et al., 2025; Liu et al., 2025).

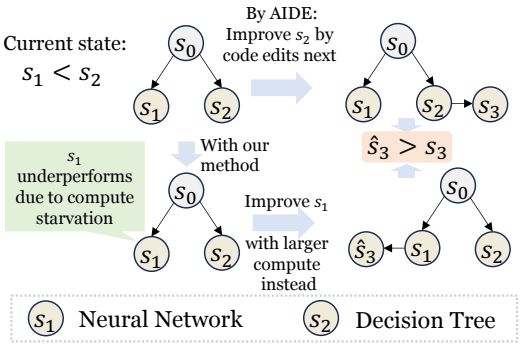

Figure 1: Demonstration of how a deconfounded memory mechanism alleviates false-negative lock-in and improves the R&D performance of ML engineer agent systems.

To tackle the above challenges, we propose a novel **DEC**onfounded **MEM**ory (**DEC-MEM**). To address the first challenge, DEC-MEM performs causal evidence attribution after each R&D episode. It first constructs an initial belief distribution over potential failure causes from the execution logs and then performs counterfactual probing, which modifies and re-executes a part of the episode to acquire additional evidence and an updated cause distribution. To address the second challenge, DEC-MEM adopts an evidence-aware writing policy that stores each hypothesis with its attributed causes and evidence. Instead of discarding a hypothesis after one negative outcome, DEC-MEM removes it only after repeated examinations consistently attribute the failures to the hypothesis itself. To address the third challenge, DEC-MEM employs an uncertainty-aware retrieval policy that balances the exploitation of effective hypotheses with the re-examination of potential false-negative ones. Specifically, part of the retrieval budget prioritizes relevance and effectiveness, while the remaining part prioritizes unexploredness and uncertainty. This design increases the opportunity for false-negative hypotheses to be re-examined while preserving memory quality. Additionally, we propose a set of evaluation metrics to characterize how well a memory mechanism can identify the real cause of a negative outcome and allow re-examination of false-negative hypotheses.

Our contributions are summarized as follows: (1) **Problem Formulation.** We formulate and investigate the novel problem of designing memory mechanisms under confounded R&D outcomes for ML engineer agents, and propose novel metrics to study it quantitatively. (2) **Method Design.** We propose DEC-MEM, which attributes the real causes behind the negative outcomes, records hypotheses with their causes and evidence, and retrieves false-negative hypotheses for re-examination. (3) **Experimental Evaluation.** We conduct extensive experiments on ML engineer agent benchmarks, demonstrating that DEC-MEM can effectively attribute the real causes of negative R&D outcomes and improve R&D performance by re-examining false-negative hypotheses.

2 PRELIMINARIES

We denote an R&D stream of T episodes by $\{e_t\}_{t=1}^{T}$, where the t-th episode is written as $e_t = (q_t, h_t, c_t, l_t, y_t)$: q_t is the ML task query, h_t the hypothesis, c_t the code implementation, l_t the episode execution log, and y_t the measured performance. Let $\mathcal{M}$ denote the memory storage. Many existing ML engineer agent systems write to and retrieve from memory through $\mathcal{M}_t = \mathsf{Write}(\mathcal{M}_{t-1}, e_t)$ and $\mathcal{R}_{t+1} = \mathsf{Retrieve}(\mathcal{M}_t, q_{t+1})$, where $\mathcal{R}_{t+1}$ is selected based on relevance to q_{t+1} and outcome-based effectiveness.

3 NOVEL METRIC & PROBLEM DEFINITION

We introduce the false-negative F1 (FNF1) and lock-in rate (FNLR) to quantify the extent to which an ML engineer agent can detect if the negative outcome of an episode is confounded and if it permanently excludes valid hypotheses after observing the confounded negative outcomes.

Definition 1 (False-Negative Recall & Lock-In Rate)**.** *Consider an R&D stream $\{e_t\}_{t=1}^{T}$ and let $\mathcal{E}_- = \{e_t : Neg(e_t) = 1\}$ denote its negative episodes. For each cause $z \in \mathcal{Z} = \{\mathrm{HYP}, \mathrm{HNS}, \mathrm{CPT}, \mathrm{MSN}\}$, compute precision Prec_z and recall Rec_z from the attributed cause $\hat{z}_t$ and ground-truth cause z_t over $\mathcal{E}_-$, and define $\mathrm{F1}_z = \frac{2\,\mathrm{Prec}_z\,\mathrm{Rec}_z}{\mathrm{Prec}_z + \mathrm{Rec}_z}$. The false-negative F1 (FNF1) is defined as the macro-averaged class-wise F1: $\mathrm{FNF1} := \frac{1}{|\mathcal{Z}|}\sum_{z\in\mathcal{Z}} \mathrm{F1}_z$. Let $u = (q, h)$ denote an ML task–hypothesis pair, and let $\mathrm{Conf}(e_t) = 1$ indicate that its outcome is* confounded negative*: the hypothesis is valid, but the observed failure is attributed to the harness (HNS), compute budget (CPT), or performance measurement (MSN), rather than the hypothesis itself (HYP). Define the first confounded negative episode for u as $\tau(u) := \min\big\{t \in [T] : (q_t, h_t) = u,\ \mathrm{Neg}(e_t) = 1,\ \mathrm{Conf}(e_t) = 1\big\}$. u is locked in if it is never re-examined after that episode:*

$$\mathcal{F} := \big\{u \in \mathcal{U} : \nexists\, s > \tau(u) \text{ such that } (q_s, h_s) = u\big\}.$$

For a stream with $\mathcal{U} \neq \varnothing$, its false-negative lock-in rate is $\mathrm{FNLR} := \frac{|\mathcal{F}|}{|\mathcal{U}|}$.

FNF1 evaluates memory writing, measuring if the memory mechanism can identify the real cause of the negative outcome, while FNLR evaluates retrieval, measuring if genuinely valid hypotheses can be re-examined in later research episodes after their first episodes with negative outcomes.

Additionally, we define cumulative performance gain (CPG) to measure R&D performance. Assume that performance scores are oriented such that larger values mean better performance, and let y_t denote the observed outcome of episode t, i.e. the change in performance of the proposed hypothesis relative to the baseline pipeline for that episode's problem q_t. The CPG of an R&D stream is then $\mathrm{CPG} := \sum_{t=1}^{T} \max(y_t, 0)$. That is, CPG sums the effects of the episodes that have positive outcomes. In cases where a stream contains multiple tasks with different metric scales, their performance scores are calculated for each task and normalized before being summed as CPG.

Problem 1 (Memory under Confounded R&D Outcomes)**.** *Given an R&D stream $\{e_t\}_{t=1}^{T}$ in which each outcome y_t depends jointly on the hypothesis h_t and confounding causes z_t, design the memory operators* Write *and* Retrieve *to minimize the expected FNLR while maximizing the FNF1 and CPG under fixed memory and computational budgets:*

$$\underset{\textit{Write, Retrieve}}{\text{minimize}} \quad \mathbb{E}[\,\mathrm{FNLR}], \qquad \underset{\textit{Write, Retrieve}}{\text{maximize}} \quad \mathbb{E}[\mathrm{CPG}, \mathrm{FNF1}].$$

Remark 1. *In our work, we assume that all possible causes of the negative outcome of an R&D episode can be attributed to one of the four causes: hypothesis (HYP), harness (HNS), compute budget (CPT), and performance measurement (MSN) following the process of an R&D episode.*

4 METHODOLOGY

In this section, we first present an overview of the proposed DEC-MEM memory mechanism, followed by the detailed design of its three components: gathering evidence for episode failure causes, writing hypotheses as memory items, and retrieving them.

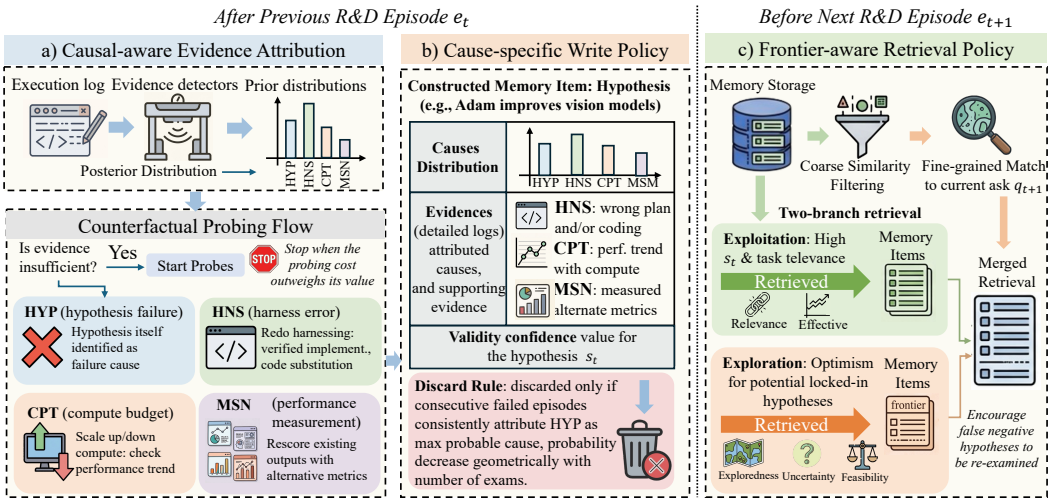

Figure 2: Overview of our proposed DEC-MEM memory mechanism.

4.1 Overview

As shown in Fig. 2, DEC-MEM updates memory after an R&D episode e_t and retrieves from the memory storage before the next episode e_{t+1}. When an episode receives a negative outcome, DEC-MEM estimates the distribution over possible causes from its execution log and, when useful, gathers further evidence through cost-aware counterfactual probes. It then stores the hypothesis with the real cause and associated evidence, so a negative outcome does not automatically demote the hypothesis. Before e_{t+1}, DEC-MEM retrieves both relevant hypotheses and underexplored hypotheses whose validity remains uncertain. This gives potentially false-negative hypotheses opportunities to be re-examined and thus improves the overall R&D performance of the agent system.

4.2 Causal-aware Cause Attribution and Evidence Acquisition

We introduce how DEC-MEM identifies the real cause of an episode with a negative outcome. Specifically, we seek evidence supporting our real cause determination from two levels of sources, namely the execution log that the episode has produced and the counterfactual probing experiments on the episode. This process will stop when the memory mechanism has acquired enough evidence.

Execution Log Examination. Since the execution log l of an episode e is a handy resource without the need for any additional compute, we inspect the log to build the initial belief distribution for real causes. To extract organized evidence from the execution log, we design k evidence detectors $\{d_i\}_{i=1}^{k}$, where each is an agent instructed with a hand-written rule to track whether the execution log matches a specific pattern, e.g., a non-finite loss, or a badly trained classifier that always outputs the same class given any input. A detector alarms, denoted by $d_i = 1$, if it detects its instructed pattern when scanning the execution log. Then, we handcraft probability values $P(d_i|z)$ specifying how likely each detector d_i is to alarm given the episode e's real cause z based on our belief. Now, we can naturally write our estimated real cause distribution as $P(z|l) \propto \Pi_{i,d_i=1} P(d_i|z)$. However, the execution log is, in many cases, insufficient to identify the real failure cause, as elaborated in the evidence attribution gap in Section 1. Therefore, we proceed to perform counterfactual probing on the episode to acquire more precise evidence with a small amount of additional compute.

Counterfactual Probing. Generally, each R&D episode goes through hypothesis proposal, code implementation, code execution, and performance evaluation. These steps correspond one-to-one to the potential causes of negative outcomes of an episode: HYP, HNS, CPT, and MSN, as specified in Remark 1. To examine a particular cause, a counterfactual probing experiment re-runs the corresponding step under an intervention that eliminates that cause, executes all subsequent steps without further intervention, and inspects the resulting execution artifacts for evidence.

Specifically, the counterfactual probe for HNS replaces the original code with a verified correct implementation of the hypothesis. If the episode no longer produces a negative outcome , it indicates

that the original harness causes the negative outcome. For CPT, we vary the compute budget during code execution and examine the performance trend. We first reduce the compute budget because this probe is less costly. If the reduced-budget run performs comparably to the original episode, performance has likely already plateaued w.r.t. compute, suggesting that the original budget was sufficient. We therefore consider CPT not a likely cause for the negative outcome. Conversely, if the original run outperforms the reduced-budget run, performance may still improve with additional compute, motivating a scale-up probe. Improvement under the increased budget provides evidence that insufficient compute causes episode failure. Finally, the counterfactual probe for MSN re-scores the episode's existing outputs using an alternative metric aligned with the task objective. If the episode receives a positive outcome under this metric, the result provides evidence that inappropriate performance measurement was a likely cause of the observed negative outcome.

Sequential Evidence Selection. To avoid unnecessary computation, DEC-MEM adaptively determines which counterfactual probe set $\mathcal{J}$ to run and when to stop. Let z denote the true cause of a negative outcome and z' the attributed cause. We structure the misattribution loss $L(z', z)$ according to the consequences of an incorrect memory update: $L(\text{HYP}, z) > L(z', \text{HYP}) > L(z', z)$, for $z, z' \neq \text{HYP}$ and $z' \neq z$, with $L(z, z) = 0$. Incorrectly demoting the hypothesis incurs the greatest loss because it discards a potentially valid hypothesis. Failing to identify an invalid hypothesis incurs a smaller loss because the hypothesis remains available for re-examination. Confusing two non-hypothesis causes incurs the smallest loss because either update retains the item.

Given the current distribution P over causes of a negative outcome, inferred from the execution log, the Bayes risk of stopping probing and immediately writing to memory is $R(P) = \min_{z'} \sum_z P(z)L(z', z)$. For each remaining probe j, let u_j denote its outcome, c_j its computational cost, and $P^{u_j}(z) \propto P(z)P(u_j \mid z)$ the posterior after observing u_j. DEC-MEM selects the probe with the greatest expected risk reduction per unit of computation:

$$j^* = \arg\max_j \frac{R(P) - \mathbb{E}_{u_j}[R(P^{u_j})]}{c_j}, \quad \text{and stops if the maximum ratio is at most } \gamma. \tag{1}$$

The numerator is the value of information of probe j, while γ controls the trade-off between cause attribution accuracy and computation. After probing, DEC-MEM combines the log-based prior with all observed probe outcomes with $P(z \mid l, \mathcal{J}) \propto P_0(z \mid l) \prod_{j \in \mathcal{J}} P(u_j \mid z)$.

4.3 Cause-Specific Writing Policy

DEC-MEM writes each memory item as $m_t = (h_t, \mathcal{E}_t, s_t)$, where $\mathcal{E}_t$ denotes the evidence and $s_t \in [0, 1]$ denotes the confidence, produced by our memory writing agent, that h_t is valid. Although the distribution $P_t(z|l, \mathcal{J})$ over causes is maintained in the memory item to inform future R&D episodes, the most probable cause (i.e., $\hat{z} = \arg\max_z P_t(z|l, \mathcal{J})$) determines how DEC-MEM updates the memory items to avoid high noise when they are used in future R&D episodes.

First, when HNS is the most probable cause, DEC-MEM records the diagnosed harnessing error and its corrected implementation. Second, if CPT or MSN is the most probable cause, it records the evidence gathered from probing the compute budget or evaluation metric, respectively, while retaining the hypothesis in memory for re-examination. Note that in the first two cases, we do not reduce the hypothesis's validity confidence as the result is confounded. Finally, when HYP itself is identified as the most probable cause, we attach evidence to memory and update its validity confidence. Upon retrieval, the stored evidence can guide the agent to improve on later R&D episodes, such as reusing a corrected implementation, adjusting the compute budget, or changing an evaluation metric.

We discard a hypothesis conservatively. Specifically, a hypothesis h_t is discarded only when its validity confidence falls below a predefined threshold and it has accumulated negative evidence from episodes with negative outcomes under at least ν distinct combinations of task, model, compute budget, and evaluation metric. Requiring evidence from distinct settings reduces the chance that a single unfavorable setting causes a potentially valid hypothesis to be discarded.

4.4 Frontier-Aware Retrieval Policy

Before the next episode e_{t+1}, DEC-MEM retrieves both task-relevant hypotheses with high validity confidence and the hypotheses that may have been overlooked after confounded negative

outcomes. Given task query q_{t+1} and retrieval budget B, it first filters memory items by compatibility with the task's problem type and data modality. An *exploitation branch* then selects $B/2$ items based on fine-grained task relevance and validity confidence s_m. From the remaining candidate hypotheses, an *exploration branch* selects the $B/2$ items with the highest frontier scores: $S_{\text{frontier}}(m) = H(s_m)\left(\frac{1}{1+n_m}\right)^{\lambda}\rho(m)$, where $H(s_m)$ is the binary entropy of validity confidence, n_m is the number of pieces of evidence already collected for the hypothesis on the task, and λ controls the preference for less-examined hypotheses. The feasibility factor $\rho(m)$ downweights candidates requiring excessive compute. Thus, the exploration branch prioritizes computationally feasible hypotheses whose validity remains uncertain after limited examination. By selecting uncertain, underexplored hypotheses, the exploration branch gives potential false-negative hypotheses chances for re-examination.

5 Empirical Evaluations

We evaluate our proposed DEC-MEM by examining the following four research questions: **RQ1:** Does DEC-MEM alleviate false-negative lock-in, thus improving cumulative R&D performance compared with existing memory mechanisms? **RQ2:** How does each component of DEC-MEM contribute to its overall performance? **RQ3:** How do the probing cost coefficient γ (in Equ. 1) and the hypothesis discarding threshold ν (in Section 4.3) change the behavior of DEC-MEM? **RQ4:** Is there a case study in which DEC-MEM accurately attributes failure causes and protects valid hypotheses under confounded negative outcomes to achieve optimized R&D performance?

5.1 Experiment Settings

We introduce the experiment settings. For more details, please see Appendices E and F.

Evaluation Protocol. As in Definition 1, exact computation of FNLR requires knowing whether each hypothesis is valid. Establishing this ground truth requires evaluating the hypothesis with a correct harness under varied compute budgets and evaluation metrics. Therefore, we first curate 12 hypotheses (half valid and half invalid) for each ML task based on a common task-specific base pipeline. For each hypothesis associated with an ML task, we implement it correctly, evaluate it at four compute budgets using two performance metrics, and record the resulting outcomes in a table. During the controlled evaluation, the agent system proposes from this curated hypothesis set, while its memory, attribution, and retrieval decisions remain online and affect subsequent episodes.
Remark. This controlled design isolates memory behavior: every candidate is evaluated with a correct implementation across multiple compute budgets and metrics, enabling exact computation of FNF1 and FNLR while preserving online memory interaction across episodes.

Datasets and Baselines. For datasets, we carefully select eight tasks drawn from the task families of MLE-bench Lite (Chan et al., 2025), a representative benchmark of ML engineer agent systems. For baselines, we extract the memory mechanisms from current ML engineer agent systems AIDE (Jiang et al., 2025), AIRA (Hambardzumyan et al., 2026), and MLEvolve (Du et al., 2026). Besides, we also examine representative agentic memory mechanisms Reflexion (Shinn et al., 2023), ExpeL (Zhao et al., 2024), A-MEM (Xu et al., 2026b), Mem0 (Chhikara et al., 2025), ReasoningBank (Ouyang et al., 2026), and Memp (Fang et al., 2026). Additionally, we compare with a baseline termed "no-memory" which does not have a memory mechanism across agents' R&D episodes.

Backbone LLMs and Evaluation Metrics. We implement the agents with various backbone LLMs, including GPT OSS 120B Agarwal et al. (2025), Qwen 3.5 397B (Team, 2026), and DeepSeek-v4-pro (Xu et al., 2026a). Our evaluation aims to measure false-negative lock-in and general performance of the R&D episodes across the research stream conducted by the agents. In terms of lock-in measurement, we utilize our proposed FNF1 and FNLR. In terms of general performance, we adopt cumulative performance gain (CPG, defined in Section 3) and probe cost. Note that the probing cost is a normalized $[0, 1]$ value that faithfully reflects its relative computational cost.

5.2 Effectiveness of DEC-MEM

Table 1: R&D stream performance (10 seeds) with 30 R&D episodes per stream. The upper methods are existing memory mechanisms for agent systems and the bottom methods are memory mechanisms extracted from ML engineer agent systems. The best and second-best values are highlighted in **bold** and underlining.

	DeepSeek-v4-pro			GPT-OSS-120B			Qwen3.5 397B		
Memory mechanism	FNLR↓	FNF1↑	CPG↑	FNLR↓	FNF1↑	CPG↑	FNLR↓	FNF1↑	CPG↑
DEC-MEM (Ours)	**0.392**±0.190	**0.375**±0.148	**0.396**±0.230	**0.198**±0.090	**0.402**±0.083	**0.509**±0.283	**0.212**±0.140	**0.453**±0.097	**0.487**±0.160
No memory	0.436±0.130	0.100±0.037	0.201±0.117	0.375±0.192	0.110±0.025	0.142±0.088	0.402±0.154	0.114±0.029	0.165±0.105
Reflexion	0.452±0.181	0.103±0.023	0.266±0.161	0.518±0.177	0.088±0.022	0.451±0.258	0.549±0.248	0.112±0.032	0.360±0.253
Memp	0.582±0.139	0.124±0.038	0.185±0.130	0.505±0.255	0.105±0.026	0.138±0.117	0.598±0.173	0.116±0.019	0.062±0.045
ExpeL	1.000±0.000	0.114±0.023	0.228±0.157	1.000±0.000	0.102±0.023	0.327±0.128	1.000±0.000	0.117±0.023	0.241±0.166
Mem0	1.000±0.000	0.110±0.037	0.236±0.196	1.000±0.000	0.102±0.029	0.367±0.165	1.000±0.000	0.094±0.024	0.418±0.244
ReasoningBank	1.000±0.000	0.106±0.019	0.368±0.224	0.993±0.023	0.110±0.030	0.323±0.155	1.000±0.000	0.115±0.031	0.389±0.201
A-MEM	1.000±0.000	0.120±0.021	0.342±0.182	1.000±0.000	0.114±0.016	0.309±0.203	1.000±0.000	0.116±0.020	0.312±0.240
MLEvolve	0.874±0.124	0.093±0.031	0.308±0.172	0.908±0.073	0.115±0.022	0.258±0.104	0.940±0.089	0.119±0.026	0.371±0.266
AIDE	0.891±0.111	0.103±0.023	0.389±0.160	0.916±0.075	0.109±0.025	0.226±0.137	0.910±0.109	0.116±0.028	0.387±0.250
AIRA	0.972±0.066	0.099±0.025	0.355±0.202	0.962±0.050	0.102±0.025	0.319±0.204	0.982±0.038	0.116±0.031	0.292±0.200

Table 2: R&D stream performance (10 seeds) under the same compute budget (experiment time for each stream and each of its episodes). The best and second-best values are in **bold** and underlining.

	DeepSeek-v4-pro			GPT-OSS-120B		
Memory mechanism	FNLR↓	FNF1↑	CPG↑	FNLR↓	FNF1↑	CPG↑
DEC-MEM (Ours)	**0.181**±0.159	**0.393**±0.085	0.815±0.484	**0.281**±0.178	**0.438**±0.151	**0.756**±0.354
No memory	0.337±0.119	0.088±0.025	0.300±0.104	0.612±0.140	0.109±0.015	0.119±0.053
Reflexion	0.328±0.309	0.118±0.015	0.629±0.278	0.529±0.167	0.110±0.027	0.649±0.330
Memp	0.320±0.135	0.131±0.015	0.589±0.469	0.513±0.134	0.130±0.031	0.305±0.127
ExpeL	0.991±0.019	0.131±0.024	0.510±0.265	1.000±0.000	0.131±0.024	0.479±0.123
Mem0	0.994±0.014	0.124±0.022	**0.821**±0.154	1.000±0.000	0.127±0.018	0.623±0.311
ReasoningBank	1.000±0.000	0.128±0.025	0.604±0.155	0.987±0.030	0.122±0.016	0.453±0.197

We answer **RQ1** by comparing DEC-MEM with general-purpose memory mechanisms for agent systems and memory in existing ML engineer agents. The evaluation is along three axes. First, we compare systems over the same number of episodes, giving each the same number of opportunities to write and retrieve cross-episode memory. Second, we compare memory mechanisms under the same compute budget by limiting the time for each R&D stream and episode. Tables 1 and 2 show that DEC-MEM achieves the lowest FNLR and the highest FNF1 and CPG across almost all baselines and LLM backbones under both conditions. Third, we evaluate R&D in open environments without providing the agents with hypothesis choices. Given financial constraints, we test one representative engineer agent AIDE (Jiang et al., 2025) with a GPT-OSS-120B backbone (Agarwal et al., 2025) on four MLEbench-Lite datasets, using five seeds per task and the same time budget. We score completed streams using a medal score derived from Kaggle's official performance metric (see Appendix H.4). We can observe from Figure 3 that DEC-MEM improves R&D effectiveness on all four tasks.

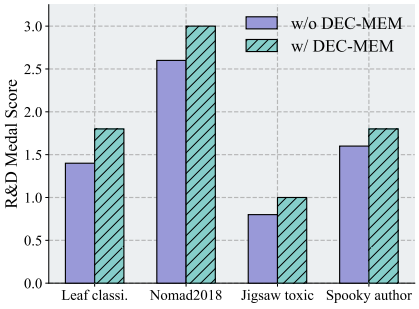

Figure 3: Performance of AIDE w/ and w/o DEC-MEM on open-world R&D streams on four MLEBench-Lite tasks. DEC-MEM effectively improves R&D performance of ML engineer agents.

5.3 Ablation Study

We examine **RQ2**. We ablate the major components of DEC-MEM on DeepSeek-v4-pro (Xu et al., 2026a); the results are shown in Fig. 4. First, we skip execution log examination (LE in Fig. 4) or replace the causal-aware evidence acquisition module with directly prompting an LLM with the execution log (CA in Fig. 4) for evidence, which raises FNLR, indicating the reliability of our proposed evidence acquisition procedure. Second, removing the cause-specific memory writing policy (CW in Fig. 4) also increases FNLR to 0.497 and 0.520, respectively, showing that both strong evidence and its use in memory writing are necessary. Lastly, removing frontier-aware retrieval (FR in Fig. 4) causes significant R&D degradation, increasing FNLR from 0.392 to 0.790 and reducing

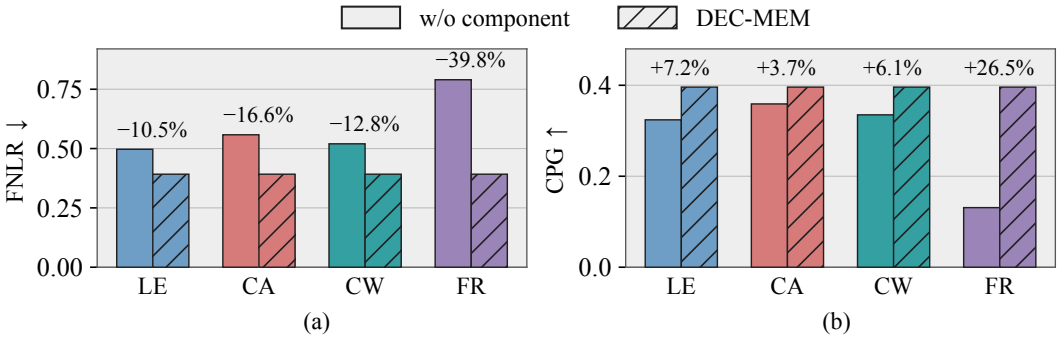

Figure 4: Ablation results on DeepSeek-v4-pro (Xu et al., 2026a), where hatched bars denote the complete DEC-MEM model. From left to right, we compare DEC-MEM with its execution log examination (LE), causal-aware attribution (CA), writing (CW) and retrieval (FR) policies ablated.

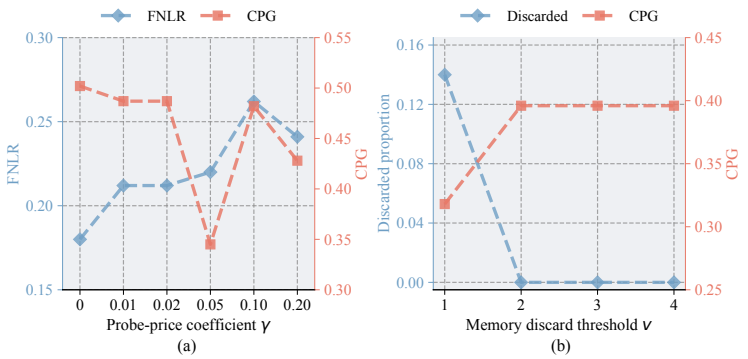

Figure 5: Parameter sensitivity of DEC-MEM. (a) Varying the probe-price coefficient γ changes the FNLR and CPG, revealing the trade-off between additional evidence and probing cost. (b) Varying the memory-discard threshold ν improves CPG and reduces the number of hypotheses discarded.

CPG from 0.396 to 0.131. This supports our claim that preserving a hypothesis in memory is insufficient unless unresolved hypotheses can re-enter later R&D episodes.

5.4 Parameter Analysis

We investigate **RQ3** by conducting R&D streams with agents powered by a DeepSeek-v4-pro backbone (Xu et al., 2026a), varying both the probing compute-cost coefficient γ (see Section 4.2) and the memory-discarding threshold ν (see Section 4.3). γ balances the benefit of acquiring additional evidence against its computational cost. DEC-MEM stops probing when the expected reduction in attribution risk per unit of compute falls below γ, indicating that further probing is no longer cost-effective. Consequently, as shown in Fig. 5 (a), a larger γ leads to fewer probes. The resulting evidence is less sufficient for accurately attributing the causes of underperforming R&D episodes, leading to a higher FNLR and lower CPG. The steep changes in these metrics further demonstrate the importance of appropriately tuning γ to achieve a favorable balance

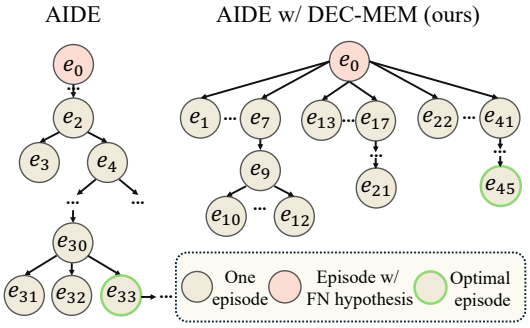

Figure 6: Case study. AIDE R&D stream w/ and w/o DEC-MEM. DEC-MEM revisits a valid hypothesis after a harness fault, significantly improving R&D performance.

between probing cost and performance. From Fig. 5 (b), we observe that increasing the memory discard threshold can improve CPG, as it avoids potentially useful hypotheses being discarded.

5.5 Case Study

We answer **RQ4** with a case study of AIDE (Jiang et al., 2025) powered by a DeepSeek- v4-pro (Xu et al., 2026a) backbone on an MLE-Bench Lite task about online comment sentiment classification (cjadams et al., 2017). As shown in Fig. 6, we present two R&D streams with and without our DEC-MEM. In the two streams, we apply the same harness fault (i.e., a silent bug) to the initial R&D episode e_0. We can observe that the AIDE w/o DEC-MEM (left side) demotes the hypothesis of e_0 because of its negative outcome and never revisits it. However, we observe that the agent equipped with DEC-MEM can attribute the cause as non-hypothesis and revisit e_0 in some later episodes, and finally one branch derived from a re-examination of e_0 leads to substantially better performance (macro-F1 0.9192) compared to that of vanilla AIDE (macro-F1 0.8842).

6 Related Work

Machine Learning Engineer Agents. LLM-based agents are used to automate machine learning (ML) research and development, including solution design, code implementation, experiment execution, and performance evaluation. AIDE (Jiang et al., 2025) formulates ML engineering as tree search over code, while R&D-Agent (Yang et al., 2025) separates research from development and supports parallel exploration across candidate solutions. Recent work has moved toward more structured research processes. MLE-Ideator decouples ideation from implementation and trains the ideation component with execution-based rewards (Zhang et al., 2026), whereas MARS combines budget-aware search, modular implementation, and comparative reflection across candidate solutions (Chen et al., 2026). These systems improve how hypotheses are generated, implemented, and explored, but generally use R&D outcome to decide which hypothesis should guide later episodes. Our work instead studies the reliability of this feedback signal and addresses cases in which implementation, resource, or evaluation failures cause a valid hypothesis to appear ineffective.

Memory and Experience Reuse for LLM Agents. Memory enables LLM agents to improve without updating model parameters by storing interactions and retrieving them in later decisions. Generative Agents introduced a memory stream that combines recency, importance, and relevance with higher-level reflection (Park et al., 2023), while Reflexion stores verbal feedback from previous trials in an episodic buffer (Shinn et al., 2023). ExpeL extracts transferable insights from collections of successful and failed trajectories and recalls both insights and examples at inference time (Zhao et al., 2024). More recent memory architectures emphasize long-horizon organization and maintenance: H-Mem models temporal evolution and semantic relations through tree and graph structures (Yu et al., 2026), and Infini Memory consolidates observations into maintainable topic documents that are inspected through iterative retrieval (Ji et al., 2026). Within ML engineering, MARS derives comparative lessons across search branches (Chen et al., 2026), while ML-Master 2.0 hierarchically distills transient execution traces into stable knowledge for ultra-long research streams (Zhu et al., 2026). These methods improve memory representation, consolidation, and retrieval. However, they do not explicitly separate hypothesis validity from the potentially confounded outcome of the episode in which the hypothesis was tested. DEC-MEM complements this literature by attaching attributed causes and supporting evidence to hypothesis memories and by retaining uncertain hypotheses until repeated examinations support their rejection.

7 Conclusion

In this work, we identify false-negative lock-in as a critical limitation of outcome-based memory mechanisms currently used in ML engineer agents. Therefore, we propose a principled deconfounded memory mechanism that attributes real causes of negative R&D outcomes through cost-aware counterfactual probing, preserves cause-specific evidence during memory writing, and re-examines unresolved hypotheses during retrieval. Extensive experiments demonstrate that DEC-MEM consistently reduces false-negative lock-in while improving cumulative R&D performance. Our work highlights the importance of distinguishing unsuccessful episodes from invalid hypotheses when designing memory mechanisms for self-improving ML engineer agents.

AI USE STATEMENT

In this work, we used generative AI tools for improve the clarity, grammar, concision, and organization of the manuscript. All of the paper's claims, methodology, experiments, and conclusions are independently developed by the authors. The LLMs evaluated as agent backbones constitute part of the experimental methodology and are reported in the corresponding experimental sections. We take responsibility for the final content of this work, including text, claims or artifacts produced with AI.

ETHICS STATEMENT

This work uses public ML benchmarks and does not involve human participants, private information, or personally identifiable data. Although DEC-MEM aims to improve the reliability of autonomous ML engineer agents, such systems may still propagate erroneous hypotheses, consume substantial computational resources, or be misused in high-stakes applications. DEC-MEM does not guarantee the correctness of generated hypotheses or implementations; therefore, human oversight and independent validation remain necessary in real-world deployments.

REPRODUCIBILITY STATEMENT

We provide the formulation, algorithmic components, evaluation protocol, datasets, backbone models, baselines, metrics, and hyperparameter analyses needed to reproduce our experiments in Appendix. Each main result is evaluated over multiple random seeds, and simulated probe costs are defined relative to the estimated average computation required by one R&D episode. We will release the implementation, prompts, configurations, predetermined hypothesis sets, cached experimental outcomes, and evaluation scripts to facilitate independent reproduction.

REFERENCES

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. Mle-bench: Evaluating machine learning agents on machine learning engineering. In *ICLR*, 2025.

Jiefeng Chen, Bhavana Dalvi Mishra, Jaehyun Nam, Rui Meng, Tomas Pfister, and Jinsung Yoon. MARS: Modular agent with reflective search for automated AI research. In *ICML*, 2026.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv*, 2025.

cjadams, Jeffrey Sorensen, Julia Elliott, Lucas Dixon, Mark McDonald, nithum, and Will Cukierski. Toxic comment classification challenge. `https://www.kaggle.com/competitions/jigsaw-toxic-comment-classification-challenge`, 2017. Kaggle.

Shangheng Du, Xiangchao Yan, Jinxin Shi, Zongsheng Cao, Shiyang Feng, Zichen Liang, Boyuan Sun, Tianshuo Peng, Yifan Zhou, Xin Li, et al. Mlevolve: A self-evolving framework for automated machine learning algorithm discovery. *arXiv*, 2026.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory. In *ACL*, 2026.

Yang Gu, Hengyu You, Jian Cao, Muran Yu, Haoran Fan, and Shiyou Qian. Large language models for constructing and optimizing machine learning workflows: A survey. *ACM Trans. Softw. Eng. Methodol.*, 2026.

Karen Hambardzumyan, Nicolas Baldwin, Edan Toledo, Rishi Hazra, Michael Kuchnik, Bassel Al Omari, Thomas Simon Foster, Anton Protopopov, Jean-Christophe Gagnon-Audet, Ishita Mediratta, et al. Aira_2: Overcoming bottlenecks in ai research agents. *arXiv*, 2026.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. Mlagentbench: Evaluating language agents on machine learning experimentation. *arXiv*, 2023.

Suozhao Ji, Baodong Wu, Zehao Wang, Lei Xia, Qingping Li, Ruisong Wang, Wenbo Ding, Zhenhua Zhu, Boxun Li, Guohao Dai, et al. Infini memory: Maintainable topic documents for long-term llm agent memory. *arXiv*, 2026.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. Aide: Ai-driven exploration in the space of code. *arXiv*, 2025.

Patrick Tser Jern Kon, Qiuyi Ding, Jiachen Liu, Xinyi Zhu, Jingjia Peng, Jiarong Xing, Yibo Huang, Yiming Qiu, Jayanth Srinivasa, Myungjin Lee, et al. Exp-bench: Can ai conduct ai research experiments? In *ICLR*, 2026.

Zexi Liu, Yuzhu Cai, Xinyu Zhu, Yujie Zheng, Runkun Chen, Ying Wen, Yanfeng Wang, Weinan E, and Siheng Chen. ML-Master: Towards AI-for-AI via integration of exploration and reasoning. *arXiv*, 2025.

Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. A comprehensive overview of large language models. *ACM Trans. Intell. Syst. Technol.*, 2025.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *ICLR*, 2026.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *UIST*, 2023.

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

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. PaperBench: Evaluating AI's ability to replicate AI research. *arXiv*, 2025.

Mohammad Ali Takallou, Amit Gautam, and Prathik Shirolkar. Llms for end-to-end machine learning: A comprehensive review. *IEEE Access*, 2026.

Qwen Team. Qwen3. 5-omni technical report. *arXiv*, 2026.

Hjalmar Wijk, Tao Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Josh Clymer, Jai Dhyani, Elena Ericheva, Katharyn Garcia, Brian Goodrich, Nikola Jurkovic, Megan Kinniment, Aron Lajko, Seraphina Nix, Lucas Sato, William Saunders, Maksym Taran, Ben West, and Elizabeth Barnes. RE-Bench: Evaluating frontier AI R&D capabilities of language model agents against human experts. In *ICML*, 2025.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv*, 2026a.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *NeurIPS*, 2026b.

Xu Yang, Xiao Yang, Shikai Fang, Yifei Zhang, Jian Wang, Bowen Xian, Qizheng Li, Jingyuan Li, Minrui Xu, Yuante Li, Haoran Pan, Yuge Zhang, Weiqing Liu, Yelong Shen, Weizhu Chen, and Jiang Bian. R&D-Agent: An LLM-agent framework towards autonomous data science. *arXiv*, 2025.

Jiawei Yu, Yixiang Fang, Xilin Liu, and Yuchi Ma. H-Mem: A novel memory mechanism for evolving and retrieving agent memory via a hybrid structure. *arXiv*, 2026.

Yunxiang Zhang, Kang Zhou, Zhichao Xu, Kiran Ramnath, Yun Zhou, Sangmin Woo, Haibo Ding, and Lin Lee Cheong. Learning to ideate for machine learning engineering agents. In *EACL*, 2026.

Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model based agents. *arXiv*, 2024.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *AAAI*, 2024.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Zican Dong, Yupeng Hou, Beichen Zhang, Yingqian Min, Junjie Zhang, Peiyu Liu, et al. A survey of large language models. *Frontiers of Computer Science*, 2026.

Xinyu Zhu, Yuzhu Cai, Zexi Liu, Bingyang Zheng, Cheng Wang, Rui Ye, Yuzhi Zhang, Linfeng Zhang, Weinan E, Siheng Chen, and Yanfeng Wang. Toward ultra-long-horizon agentic science: Cognitive accumulation for machine learning engineering. *arXiv*, 2026.

Table 3: Summary of notation used by DEC-MEM.

Symbol	Description
$e_t = (q_t, h_t, c_t, l_t, y_t)$	The t-th R&D episode
$z_t \in \mathcal{Z}$	Latent cause of an underperforming episode
$\mathcal{Z}$	$\{\mathrm{HYP}, \mathrm{HNS}, \mathrm{CPT}, \mathrm{MSN}\}$
d_i	Execution-log evidence detector
u_j	Outcome of counterfactual probe j
c_j	Computation cost of probe j
$P_t(z)$	Cause probability assigned to z
$\mathcal{E}_t$	Cause-specific evidence stored with the hypothesis
s_t	Validity confidence of the stored hypothesis
$L(z', z)$	Loss of applying write operation $W_{z'}$ when the true cause is z
$R(P)$	Bayes risk of stopping under cause distribution P
γ	Probe-price coefficient
ν	Minimum number of qualifying unsuccessful examinations before discard
B	Number of memory items retrieved before the next episode

A Appendix Overview

This appendix provides additional material supporting the formulation, method, and empirical evaluation of DEC-MEM. We first discuss limitations of our work in Appendix B. We then provide further details of causal-aware evidence attribution, cause-specific memory writing, and frontier-aware retrieval in Appendix C, followed by the theoretical analysis in Appendix D. The construction of the controlled R&D benchmark is described in Appendix E; the experimental protocol and implementation details are reported in Appendix F. Finally, we provide additional information about the baselines, evaluation metrics, supplementary results, and case study in Appendices G–J. Appendix L then presents end-to-end memory traces from recorded R&D streams, showing the attribution, writing and retrieval steps of a single false-negative rescue.

B Limitations

Our study has several limitations. First, DEC-MEM organizes the causes of an underperforming R&D episode into four categories: an invalid hypothesis (HYP), a harness or implementation problem (HNS), an insufficient compute budget (CPT), and an unsuitable performance measurement protocol (MSN). This taxonomy follows the major stages of an R&D episode. Although it covers every failure mode encountered in a real research environment, the granularity of the taxonomy may be limited. Moreover, multiple causes may contribute to the same failure. DEC-MEM partially accommodates this case by maintaining a posterior distribution over all four causes rather than recording only a single label. Second, exact computation of the false-negative lock-in rate (FNLR) requires ground-truth knowledge of whether a hypothesis is valid under a correct implementation and an appropriate experimental configuration. We therefore construct an R&D environment from pre-executed experiments and restrict the agent to a broad but finite candidate set. This design makes hypothesis validity and failure causes observable for evaluation, but it does not completely represent open-ended scientific discovery, in which the agent may generate a previously unseen hypothesis and ground-truth validity is unavailable. Third, to isolate memory behavior under matched conditions, we use a controlled benchmark whose outcome table is constructed from real model fits. This design enables reproducible comparison across many streams while preserving variation across tasks, hypotheses, compute allocations, metrics, and injected failures. The results therefore can more cleanly isolate the effects of memory mechanisms in ML engineering processes.

C Additional Method Details

C.1 Notation

Table 3 summarizes notation used in the method and the appendix.

Table 4: Likelihood $P(d_i = 1 \mid z)$ assigned to each execution-log detector.

Detector	HYP	HNS	CPT	MSN
Out-of-memory abort	0.02	0.35	0.55	0.02
Timeout	0.03	0.20	0.60	0.03
Non-finite loss	0.02	0.70	0.05	0.02
Shape mismatch	0.01	0.75	0.02	0.02
Empty feature	0.02	0.65	0.03	0.03
Degenerate prediction	0.08	0.45	0.20	0.30
Split anomaly	0.02	0.10	0.03	0.70
Identical artifact hash	0.01	0.90	0.01	0.01

C.2 Execution-Log Evidence

The execution log provides evidence without requiring an additional model training run. DEC-MEM applies a series of detectors (in the experiment section, there are eight) covering a comprehensive range of possible issues, including, but not limited to, out-of-memory termination, timeout, non-finite loss, shape mismatch, empty features, degenerate predictions, anomalous data splitting, and identical output artifacts. Only detectors that alarm contribute a likelihood factor. This choice avoids treating the simultaneous silence of several correlated detectors as independent evidence for HYP.

The initial prior used in the experiments is

$$P_0(\mathrm{HYP}) = 0.40, \quad P_0(\mathrm{HNS}) = 0.30, \quad P_0(\mathrm{CPT}) = 0.20, \quad P_0(\mathrm{MSN}) = 0.10. \tag{2}$$

This deliberately non-uniform prior gives HYP substantial prior plausibility while assigning meaningful mass to each confounding cause based on our prior knowledge about their likelihood, allowing the controlled evaluation to test whether DEC-MEM can revise an initially uncertain attribution using execution-log and counterfactual evidence. For a set $\mathcal{D}_t^+$ of detectors that alarm in episode t, the log-derived distribution is

$$P_0(z \mid l_t) = \frac{P_0(z) \prod_{i \in \mathcal{D}_t^+} P(d_i = 1 \mid z)}{\sum_{z' \in \mathcal{Z}} P_0(z') \prod_{i \in \mathcal{D}_t^+} P(d_i = 1 \mid z')}. \tag{3}$$

When no detector alarms, DEC-MEM retains the initial prior.

Table 4 reports the detector alarm likelihoods used in the evaluation given a cause z. We note that this predefined set of likelihoods reflects our prior knowledge about the detectors' behavior and can be dynamically updated by users when more R&D stream experiences accumulate over time.

C.3 Counterfactual Probes and Posterior Update

DEC-MEM uses a causal-aware probing technique to collect evidence that is not available from the original log. The HNS probe reruns the experiment after replacing the original implementation with a verified implementation of the same hypothesis. The CPT probe changes the compute allocation and examines the performance trend. The MSN probe re-scores the stored predictions under an alternative metric aligned with the task objective. A positive change after an intervention provides evidence that the intervened cause contributed to the original failure.

The controlled implementation separates the CPT probe into a less expensive trend probe and a more expensive direct probe. The trend probe uses the tested allocation together with lower allocations, fits a least-squares slope against log allocation, and extrapolates toward the largest reachable allocation. It fires only when the slope exceeds a noise band and the extrapolated effect is positive. The direct probe evaluates the intervention at the next larger allocation and fires when the measured effect becomes positive. The direct probe is undefined when the original episode already uses the largest allocation. Similarly, a skipped or undefined probe contributes neither a likelihood factor nor computation cost.

Given the current distribution $P(z)$ and an observed probe outcome u_j, DEC-MEM computes

$$P^{u_j}(z) = \frac{P(z)P(u_j \mid z)}{\sum_{z' \in \mathcal{Z}} P(z')P(u_j \mid z')}. \tag{4}$$

The outcomes of different probes are treated as conditionally independent given the cause with $P(u_j = 1 \mid z)$ designed and dynamically adjusted based on prior knowledge and accumulated evidence.

C.4 Cost-Aware Sequential Evidence Selection

For a cause distribution P, the Bayes risk of stopping and applying a write operation immediately is

$$R(P) = \min_{z' \in \mathcal{Z}} \sum_{z \in \mathcal{Z}} P(z) L(z', z). \tag{5}$$

The asymmetric loss used in our experiments is

$$L(z', z) = \begin{cases} 0, & z' = z, \\ 1.0, & z' = \mathrm{HYP},\ z \neq \mathrm{HYP}, \\ 0.3, & z' \neq \mathrm{HYP},\ z = \mathrm{HYP}, \\ 0.1, & \text{otherwise.} \end{cases} \tag{6}$$

The largest penalty is assigned to incorrectly demoting a valid hypothesis for a confounded failure, since that error can remove the hypothesis from future consideration. Retaining an invalid hypothesis is assigned a smaller loss because it can be corrected by a later examination. Confusing two non-hypothesis causes receives the smallest nonzero loss because both decisions preserve the hypothesis, although they may attach different procedural guidance.

For each available probe j, define its expected risk reduction as

$$\Delta_j(P) = R(P) - \mathbb{E}_{u_j}\left[R(P^{u_j})\right]. \tag{7}$$

DEC-MEM selects the probe with the largest $\Delta_j(P)/c_j$ and stops when $\max_j \frac{\Delta_j(P)}{c_j} \leq \gamma$. Therefore, at termination, no remaining individual probe has an expected risk reduction greater than its price γc_j. After each observed outcome, DEC-MEM updates P and recomputes the acquisition scores of the remaining probes.

C.5 Cause-Specific Memory Writing

After evidence acquisition, DEC-MEM stores the examined hypothesis together with the complete cause distribution and the evidence associated with each cause. As defined in the main paper, a memory item takes the form

$$m_t = (h_t, \mathcal{E}_t, s_t), \tag{8}$$

where $s_t \in [0, 1]$ is the current validity confidence of h_t. We represent this confidence explicitly for each memory item and initialize a newly created memory item at $s_0 = 0.5$. For an R&D episode, we compute the reliability r_t (constrained to $[0.5, 1)$) of its attribution from the maximum cause posterior $a_t = \max_{z \in \mathcal{Z}} P_t(z \mid l, \mathcal{J})$ as

$$r_t = \min(0.95, \max(0.5,\ 0.5 + 0.6(a_t - 0.25))). \tag{9}$$

When evidence from the episode is written to the corresponding memory item, its reliability determines the strength of the update to that item's validity confidence. Thus, we use a reliability-weighted Bayesian rule to update the memory-item confidence from s_{t-1} to s_t. Writing $p = s_{t-1}$, the update is

$$s_t = \begin{cases} \dfrac{p r_t}{p r_t + (1-p)(1-r_t)}, & \text{positive evidence,} \\ \dfrac{p(1-r_t)}{p(1-r_t) + (1-p) r_t}, & \text{negative evidence.} \end{cases} \tag{10}$$

As shown by Equ. 10, positive evidence raises confidence, whereas an HYP-attributed negative lowers it. HNS evidence leaves the existing memory item's confidence unchanged because it is recorded as procedural evidence. For CPT or MSN evidence, we create a child memory item for the refuted budget or metric with initial confidence $\max(0.05, 0.35 s_{t-1})$ and then apply the negative update above; the parent memory item survives for the complementary scopes. This representation retains uncertainty when several causes remain plausible. The most probable cause determines the

immediate write operation, while the full distribution and all available evidence remain attached to the memory item for future retrieval and re-examination. For HNS, the evidence includes relevant log alarms, the diagnosed implementation problem, and the performance change obtained after correcting the implementation. For CPT, it records the allocations examined and the corresponding performance trend. For MSN, it stores the evaluation results under the original and alternative metrics. Evidence associated with HYP records the negative outcome that remains after the available non-hypothesis explanations have been examined. These records allow a future episode to avoid repeating the same confounder when the hypothesis is revisited. DEC-MEM discards a hypothesis conservatively. A hypothesis is removed only when its validity confidence is below the threshold, it has accumulated at least ν qualifying unsuccessful examinations, and HYP is the dominant cause in each of those examinations. Appendix D.1 shows that this repeated-evidence requirement geometrically decreases the probability of incorrectly discarding a valid hypothesis.

C.6 Frontier-Aware Retrieval

The retrieval policy allocates half of its budget to exploitation and half to exploration. The exploitation branch retrieves hypotheses that are relevant to the incoming problem, have sufficiently high validity confidence, and are feasible under the available experimental resources. The exploration branch operates on the remaining candidates and prioritizes hypotheses that are both uncertain and underexplored. In the controlled implementation, the frontier score has the form

$$S_{\text{frontier}}(m) = H(s_m)\, d(m)^{\lambda}\, \rho(m), \tag{11}$$

where $H(s_m)$ is the binary entropy of the validity confidence,

$$d(m) = \frac{1}{1 + n_m} \tag{12}$$

is a coverage-distance term based on the number n_m of pieces of evidence already collected for the hypothesis and task, and $\rho(m)$ is a feasibility term. Specifically,

$$\rho(m) = \frac{1}{1 + 0.25\ \text{spec}(m)}, \tag{13}$$

where $\text{spec}(m)$ counts the number of constrained fields in the experimental scope. The retriever also exposes unexplored hypothesis–allocation–metric cells for which no evidence has yet been recorded.

D Theoretical Analysis

D.1 False-Negative Retention

We next formalize how repeated examinations protect a valid hypothesis from being discarded in the memory mechanism. Consider a valid hypothesis h that produces negative outcomes in an R&D episode because of non-hypothesis factors. Let E_i denote the event that DEC-MEM nevertheless assigns HYP as the dominant cause in its i-th qualifying unsuccessful examination.

Assumption 1 (Bounded conditional attribution error). There exists $\epsilon < 1$ such that, for every i and every history of earlier attribution decisions,

$$\Pr(E_i \mid E_1, \ldots, E_{i-1},\ h \text{ is valid}) \le \epsilon. \tag{14}$$

This condition does not require attribution errors to be independent; it only requires their conditional probability to remain bounded by ϵ.

Theorem 1 (Conditional geometric retention bound). Suppose DEC-MEM discards a hypothesis only after at least ν qualifying unsuccessful examinations in which HYP is the dominant attributed cause. Thus, if the conditional probability of misattributing each qualifying examination is bounded by $\epsilon < 1$, the probability of erroneously discarding a valid hypothesis is at most ϵ^{ν}.

Table 5: Kaggle task datasets used in the controlled R&D evaluation. They are selected from the MLE-bench Lite benchmark and other comparable tabular and text task families.

Task	Modality	Rows	Features	Classes	Minority rate	MLE-bench Lite
adult	Tabular	32,561	14	2	0.241	–
telco_churn	Tabular	7,043	19	2	0.265	–
bank_marketing	Tabular	11,162	16	2	0.474	–
german_credit	Tabular	1,000	8	2	0.333	–
imdb	Text	12,000	Raw text	2	0.498	–
agnews	Text	12,000	Raw text	4	0.247	–
sms_spam	Text	5,572	Raw text	2	0.134	–
jigsaw_toxic	Text	14,000	Raw text	2	0.101	✓

Proof. Erroneously discarding a valid hypothesis requires HYP to be selected as the dominant cause in all ν qualifying examinations. The discard event is thus a subset of $\cap_{i=1}^{\nu} E_i$. Applying the probability chain rule and Equation 14 gives

$$\Pr\left(\bigcap_{i=1}^{\nu} E_i \mid h \text{ is valid}\right) = \prod_{i=1}^{\nu} \Pr(E_i \mid E_1, \ldots, E_{i-1},\ h \text{ is valid}) \tag{15}$$

$$\leq \prod_{i=1}^{\nu} \epsilon = \epsilon^{\nu}. \tag{16}$$

The additional requirement that the validity confidence fall below its threshold can only reduce the probability of discard, so the same upper bound continues to hold. □

Corollary 1. For a collection of N valid hypotheses satisfying Assumption 1, the probability that at least one is erroneously discarded is at most $N\epsilon^{\nu}$ by the union bound.

The bound exposes the protection–patience trade-off introduced by ν. Increasing ν geometrically reduces erroneous removal, but also delays the removal of genuinely invalid hypotheses. This motivates treating ν as a conservative memory parameter rather than setting it as large as possible.

E Controlled R&D Benchmark Construction

E.1 Motivation and Evaluation Design

FNLR requires identifying whether an initially unsuccessful hypothesis is in fact valid under a correct harness and an appropriate experimental configuration. Such counterfactual ground truth is generally unavailable in an open-ended R&D stream. We therefore construct a controlled benchmark in which the candidate hypotheses are fixed and their outcomes are measured in advance under multiple compute allocations and evaluation metrics. During an R&D stream, the agent selects a candidate hypothesis and experimental configuration, and the environment retrieves the corresponding measured outcome from the cache. Ground-truth fields are used only for evaluation and are never exposed to the agent for effective experimental evaluations.

E.2 Datasets

The controlled evaluation contains eight public classification datasets: four tabular datasets and four text datasets. Dataset statistics after loading and preprocessing are shown in Table 5. The minority rate is the smallest class's share of the training split. We construct a stratified 70/30 train/test split for each of the eight tasks using `train_test_split`. Before splitting, we uniformly sample 12,000 rows from each of `imdb` and `agnews`, and 14,000 rows from `jigsaw_toxic`; the other datasets are used in full. The measured outcome table uses seed 0 for both subsampling and splitting. We perform ML experiments on all eight tasks, 12 candidate hypotheses per task, four compute allocations, and two evaluation metrics and collect the results. For each setting of the ML experiments, we perform an ML experiment with a baseline pipeline and a treatment with the hypothesis

Table 6: Source paths for the Kaggle datasets used in the controlled R&D evaluation. Each path is appended to the common prefix `https://www.kaggle.com`.

Dataset	Kaggle path
adult	/datasets/uciml/adult-census-income
telco_churn	/datasets/blastchar/telco-customer-churn
bank_marketing	/datasets/janiobachmann/bank-marketing-dataset
german_credit	/datasets/uciml/german-credit
imdb	/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews
agnews	/datasets/amanandrai/ag-news-classification-dataset
sms_spam	/datasets/team-ai/spam-text-message-classification
jigsaw_toxic	/competitions/jigsaw-toxic-comment-classification-challenge

applied to the baseline pipeline. The ten seeds used to evaluate R&D streams vary the streams, not the measured outcome table. The four allocations use 3%, 10%, 35%, and 100% of the training split.

For tabular tasks, we encode categorical features with an `OrdinalEncoder` fitted on the training split, map unseen categories to -1, and convert unparseable numeric values to zero. The base classifier is a `HistGradientBoostingClassifier` with early stopping disabled. For text tasks, we fit a word-unigram `TfidfVectorizer` on the training split and use `LogisticRegression` as the base classifier. The compute allocation also controls fitting iterations and model capacity. A candidate hypothesis modifies the corresponding fixed base pipeline.

The targets are `income` for `adult`, `Churn` for `telco_churn`, `deposit` for `bank_marketing`, `sentiment` for `imdb`, `Class Index` for `agnews`, and `Category` for `sms_spam`. For `jigsaw_toxic`, a comment is labeled positive if any of its six toxicity labels is set. The public `german_credit` mirror used in our experiments lacks the original `Risk` label. We therefore define its target as whether `Credit amount` falls in the top tercile and remove that column from the features; this task does not measure credit-risk prediction. All tasks are binary except the four-class `agnews` task. The primary performance metric is macro-averaged F_1, and the secondary metric is accuracy, both computed on the held-out test split. Their difference provides evidence for MSN when a hypothesis underperforms under one metric but improves under the other.

E.3 Measured Outcome Table and Ground Truth

The shared controlled environment contains 96 task–hypothesis pairs, with 12 candidate interventions for each of the eight tasks. Each intervention is measured at four compute allocations, producing 384 cached experimental cells. Every cell records the baseline and treated performance under two metrics (we name them primary and secondary metrics for ease of distinction). Constructing these cells requires 768 model fits: one baseline fit and one treated fit for each cell. For task q, hypothesis h, and allocation b, let $\delta^{\mathrm{pri}}_{q,h,b}$ and $\delta^{\mathrm{sec}}_{q,h,b}$ denote the measured changes relative to the corresponding base pipeline. Therefore, we have

$$\delta^{\star}_{q,h} = \max_{b} \delta^{\mathrm{pri}}_{q,h,b} \tag{17}$$

and treat hypothesis h as valid on task q when $\delta^{\star}_{q,h} > 0.002$. For evaluation, we assign each negative benchmark outcome a benchmark cause using the following deterministic annotation protocol. The protocol first determines if a hypothesis is compute-limited, then identifies a metric-specific cause, and assigns HYP only when neither condition applies; injected harness faults override these labels:

$$z(q,h,b) = \begin{cases} \mathrm{CPT}, & \delta^{\star}_{q,h} > 0.002, \\ \mathrm{MSN}, & \delta^{\mathrm{sec}}_{q,h,b} > 0.002, \\ \mathrm{HYP}, & \text{otherwise.} \end{cases} \tag{18}$$

Therefore, our ML experiment results pool includes both hypotheses that improve a task under some allocation and hypotheses that do not meet the improvement threshold. Intended negative controls are evaluated by the same measured criterion as every other hypothesis; in particular, adding random feature noise meets the threshold on three tabular tasks, whereas aggressive document truncation meets it on none.

Table 7: Candidate hypotheses for the four tabular tasks. Each hypothesis modifies the ordinal-encoded `HistGradientBoostingClassifier` base pipeline. Entries show the best change in macro-averaged F_1 across the four compute allocations; ✓ denotes a valid task–hypothesis pair.

Hypothesis	Intended implementation	adult	bank_marketing	german_credit	telco_churn	Valid tasks
pca_compression	Project features onto leading principal components	−0.0440	+0.0057 ✓	+0.0560 ✓	−0.0009	2/4
smote_like_upsample	Resample the minority class up to parity before fitting	+0.0016	+0.0023 ✓	+0.0477 ✓	+0.0083 ✓	3/4
early_stopping	Enable internal early stopping on a held-out fraction	+0.0004	+0.0078 ✓	+0.0408 ✓	+0.0246 ✓	3/4
bagging_ensemble	Average predictions from models fitted on bootstrap samples	+0.0024 ✓	+0.0096 ✓	+0.0404 ✓	+0.0171 ✓	4/4
class_weight_balanced	Weight training examples by inverse class frequency	−0.0024	+0.0052 ✓	+0.0258 ✓	+0.0117 ✓	3/4
random_feature_noise	Add Gaussian noise to numeric features	−0.0064	+0.0034 ✓	+0.0172 ✓	+0.0048 ✓	3/4
low_lr_long	Set the learning rate to 0.03 and triple fitting iterations	+0.0006	+0.0092 ✓	+0.0000	+0.0140 ✓	2/4
deeper_trees	Set maximum tree depth to 10	−0.0017	+0.0093 ✓	+0.0000	+0.0090 ✓	2/4
feature_interactions	Add pairwise products of selected numeric features	−0.0015	+0.0031 ✓	+0.0053 ✓	+0.0017	2/4
more_estimators	Increase fitting iterations to 2.5 times the allocation default	+0.0000	+0.0000	+0.0010	+0.0016	0/4
wide_leaves	Increase the maximum number of leaf nodes to 96	+0.0007	+0.0000	+0.0000	+0.0000	0/4
poly_numeric	Add degree-two polynomial terms over numeric features	+0.0000	+0.0000	+0.0000	+0.0000	0/4

Table 8: Candidate hypotheses for the four text tasks. Each hypothesis modifies the TF-IDF and `LogisticRegression` base pipeline. Entries show the best change in macro-averaged F_1 across the four compute allocations; ✓ denotes a valid task–hypothesis pair.

Hypothesis	Intended implementation	agnews	imdb	jigsaw_toxic	sms_spam	Valid tasks
class_weight_balanced	Weight training examples by inverse class frequency	+0.0421 ✓	+0.0046 ✓	+0.2966 ✓	+0.4337 ✓	4/4
min_df_raise	Set `min_df` to 3 to exclude rare terms	+0.0004	+0.0130 ✓	+0.0416 ✓	+0.1726 ✓	3/4
char_ngram_features	Use `char_wb` character n-grams of lengths 3–5	+0.0226 ✓	+0.0075 ✓	+0.0458 ✓	+0.0037 ✓	4/4
ensemble_seeds	Average probabilities across seeds with varied C	+0.0088 ✓	+0.0031 ✓	+0.0248 ✓	+0.0429 ✓	4/4
tfidf_sublinear	Enable sublinear term-frequency scaling	+0.0101 ✓	+0.0262 ✓	+0.0000	+0.0016	2/4
stopword_removal	Remove English stop words during vectorization	+0.0112 ✓	+0.0111 ✓	+0.0000	+0.0000	2/4
word_bigrams	Extend the word n-gram range to $(1, 2)$	−0.0006	+0.0056 ✓	+0.0023 ✓	+0.0000	2/4
svd_compression	Apply truncated SVD to the TF-IDF features	−0.0221	−0.0122	+0.0045 ✓	+0.0046 ✓	2/4
larger_vocab	Increase maximum vocabulary size from 20,000 to 200,000	+0.0000	+0.0031 ✓	+0.0000	+0.0000	1/4
word_trigrams	Extend the word n-gram range to $(1, 3)$ with a smaller feature cap	−0.0049	+0.0020	+0.0000	+0.0000	0/4
aggressive_truncation	Truncate each document to its first 120 characters	−0.0383	−0.1463	+0.0000	+0.0000	0/4
lower_C_regularise	Strengthen regularization by setting $C = 0.25$	−0.0144	−0.0184	+0.0000	+0.0000	0/4

E.4 Harness-Failure Injection

Natural variation in the measured table provides CPT and MSN failures, whereas HNS failures are introduced through controlled fault injection. Each episode independently receives a harness fault with probability 0.30 under the default configuration. The fault decision and subtype are determined by the stream seed, episode index, task, and hypothesis, ensuring matched and reproducible streams. The fault injector samples from six subtypes: label shuffling, zero learning rate, evaluation on an incorrect split, removal of an informative feature, metric-sign inversion, and a degenerate seed collision. A sampled fault changes the observed effect to a negative value and sets the episode's ground-truth cause to HNS. Different fault subtypes leave different observable signatures in the execution log; some are deliberately silent so that the log alone cannot solve the attribution problem.

E.5 Controlled Measured-Outcome Protocol

At each episode, the Researcher Agent selects a candidate hypothesis after observing the memory context, and the Engineer Agent selects its compute allocation and evaluation metric. The environment then retrieves the corresponding measured effect from the table. It adds a carefully curated Gaussian perturbation determined by the stream seed, episode index, task, hypothesis, and allocation. This scheme makes repeated runs reproducible while avoiding bit-identical observations. Each evaluated R&D stream contains 30 episodes. Tasks are visited in a fixed cycle, while hypothesis and configuration selection depend on the backbone LLM and the memory context. Ground-truth validity, ground-truth failure causes, and validity indicators are retained only for metric computation and are not shown to the Researcher or Engineer Agent.

E.6 End-to-End Case-Study

We utilize `jigsaw-toxic-comment-classification-challenge`, a genuine MLE-bench Lite competition. We formulate it as binary classification: an online comment is positive when any of the six original labels, i.e., `toxic`, `severe_toxic`, `obscene`, `threat`, `insult`, or `identity_hate`, is set. The resulting target is named `toxic_any`, with values `clean` and `toxic`. We sample 20,000 of the 159,571 labeled rows and use a 70/30 train/test split. The positive class comprises approximately 10% of the data, and performance is measured by macro-averaged F1

Table 9: Compute allocations used in the end-to-end case study.

Allocation	Fraction of training split	Training rows
`xs`	0.03	420
`sm`	0.10	1,400
`md`	0.35	4,900
`lg`	1.00	14,000

Table 10: Default controlled-evaluation configuration in Table 1.

Quantity	Symbol/configuration	Value
Episodes per stream	T	30
Number of seeds	–	10
Harness-fault probability	–	0.30
Probe-cost safety ceiling	β	3.0
Probe-price coefficient	γ	0.02
Validity-confidence threshold	θ	0.20
Required refuting scopes	ν	2
Frontier coverage exponent	λ	1.0
Retrieved memory items	B	10
LLM temperature	–	0.7
Context window	–	8,192

on the held-out split. The compute allocation variable subsamples the training split as shown in Table 9. This design creates a controlled setting in which a valid hypothesis may appear unsuccessful under insufficient compute and become successful after additional training data are allocated.

F Experimental and Implementation Details

F.1 Shared Evaluation Protocol

All memory mechanisms are evaluated with the same task sequence, curated hypothesis set, precomputed measured outcomes, episode budget, Researcher Agent, Engineer Agent, and backbone LLM within a matched stream. The conditions differ only in the memory context presented to the agents, the attribution procedure available after a negative episode, and the resulting memory update. Each main condition is evaluated over 10 seeds, with 30 episodes per seed. The backbone is prompted through an OpenAI-compatible Ollama interface. LLM responses are cached using the model, prompt, temperature, seed, response format, and context-window size. Under the shared configuration, the sampling temperature is 0.7 and the context window is 8,192 tokens.

F.2 Default Hyperparameters

Table 10 reports the agent calling information for our experiments in Table 1. Probe computation is normalized in episode-equivalent units. The safety ceiling β limits the maximum probe cost for one negative episode, whereas γ determines whether an additional probe is worth its cost.

F.3 Probe-Cost Normalization

The four normalized compute allocation costs are 0.25, 0.5, 1.0, and 2.0 episode-equivalent units. The MSN probe reuses stored predictions and has cost 0.05. The HNS control probe has cost 0.25. The CPT trend probe charges only for additional lower-allocation runs, while the direct CPT probe charges for the next larger allocation. The original tested allocation is not charged a second time.

The probe costs used above are normalized compute costs, rather than wall-clock measurements. To test whether our conclusions depend on their relative values, we rerun complete R&D streams after multiplying the cost of one probe family by 0.5 or 2. We vary the HNS harness probes, the CPT compute-scaling probes, and the MSN measurement check separately. Each setting uses DeepSeek-v4-pro and the same ten seeds, measured effect table, and fault schedule as the reference setting.

Table 11: Sensitivity to relative probe costs, mean ± standard deviation over ten matched streams. *Charged cost* uses the prices applied during each run. *Reference-price cost* values the probes selected in that run at the original prices, allowing probe expenditure to be compared across settings. Costs are in normalized episode-equivalent units.

Probe-price setting	FNLR ↓	CPG ↑	Charged cost	Reference-price cost
Original prices	0.392 ± 0.190	0.396 ± 0.230	12.68 ± 4.08	12.68 ± 4.08
HNS $\times 0.5$	0.397 ± 0.188	0.364 ± 0.241	10.47 ± 3.35	12.41 ± 3.63
HNS $\times 2$	0.447 ± 0.133	0.338 ± 0.188	11.44 ± 4.32	9.49 ± 3.79
CPT $\times 0.5$	0.359 ± 0.160	0.356 ± 0.259	8.07 ± 2.11	12.25 ± 3.49
CPT $\times 2$	0.415 ± 0.171	0.400 ± 0.210	4.56 ± 0.77	4.56 ± 0.77
MSN $\times 0.5$	0.392 ± 0.190	0.396 ± 0.230	12.29 ± 4.02	12.68 ± 4.08
MSN $\times 2$	0.392 ± 0.190	0.396 ± 0.230	13.46 ± 4.20	12.68 ± 4.08

All seven settings use the default probe-price coefficient $\gamma = 0.02$. Across these price changes, FNLR ranges from 0.359 to 0.447 and CPG from 0.338 to 0.400, compared with 0.392 and 0.396 at the original prices. Probe selection nevertheless responds to the assigned costs. Doubling the HNS probe price reduces reference cost from 15.6 to 7.8 per stream. Halving the CPT probe price increases direct compute-scaling checks from 4.4 to 7.7, whereas doubling it eliminates those checks and reduces the attribution score to 0.415. Changing the MSN probe price leaves the selected probes and outcome metrics unchanged in this range. The cheaper CPT trend check is never selected in these runs. Thus, FNLR and CPG remain within the ranges shown above, while attribution, particularly when direct compute checks become too expensive, is more sensitive to relative probe costs. We can observe from Table 11 that our proposed DEC-MEM outperforms the baselines in Table 1 stably with varied normalized probe cost assignments.

F.4 Hardware and Software

The released controlled-evaluation code uses Python and depends on `PyYAML`, `requests`, `pandas`, and `tqdm`. Model calls are issued through Ollama, and experiments can be resumed from cached LLM responses and completed stream files. All experiments were run on a shared SLURM cluster under Ubuntu 22.04.5 LTS. Compute nodes had Intel Xeon Silver 4208 CPUs (16 logical cores) and 125 GB of RAM; individual jobs requested 4–6 CPUs and 16–32 GB of memory. The controlled R&D benchmark and the end-to-end case study both executed their ML experiments on CPUs. In our experiment on open-world R&D research streams that do not provide hypothesis choices to agents, demonstrated in Fig. 3, we allowed each episode to use one NVIDIA A100 GPU. Case-study jobs reserved a GPU, although most agent-generated pipelines used scikit-learn and did not perform GPU computation. Backbone LLMs were served through the Ollama Cloud HTTP API rather than run on the cluster.

The software environment used Python 3.10.21, NumPy 1.26.4, pandas 2.3.3, scikit-learn 1.7.2, SciPy 1.15.3, LightGBM 4.7.0, XGBoost 3.2.0, CatBoost 1.2.10, openai 1.75.0, OmegaConf 2.3.1, Hydra 1.3.6, and FAISS 1.15.0. We ran six R&D streams concurrently for the controlled evaluation and four to six for the case study, while limiting concurrent requests to the LLM service. Table 12 reports the recorded LLM calls and tokens for each controlled-evaluation experiment family. In total, these families comprise 870 R&D streams of 30 episodes each, or 26,100 benchmark episodes driven by measured outcomes. A stream typically required about 60 LLM calls and 1.5 hours of elapsed time. With six concurrent streams, a 60-stream experiment took approximately 12–15 hours, and a 90-stream experiment approximately 22 hours. These times include LLM response latency and contention on the shared cluster. The end-to-end case study comprised 24 R&D streams of 20 episodes each. It accumulated 121.7 hours of stream execution, with a median of 305 minutes per stream; running four to six streams concurrently required approximately one day of elapsed time. Each episode executed agent-generated code, subject to a 420–600-second limit per code execution and a 25-minute episode limit. Approximately 38% of episodes reached the episode limit. Constructing the measured outcome table was a separate one-time cost of 768 model fits, corresponding to a baseline and a treated fit for each of its 384 cells. We report LLM calls and

Table 12: LLM usage in the controlled R&D evaluation. Token counts include the recorded calls made during each experiment family.

Experiment family	Streams	LLM calls	LLM tokens
Main comparison, `gpt-oss:20b`	90	5,400	3,467,510
Main comparison, `deepseek-v4-pro`	90	5,400	4,356,894
Main comparison, `qwen3.5:397b`	90	5,400	20,973,682
ML engineer agent memory baselines, three backbones	90	5,400	9,551,997
Component ablations, `deepseek-v4-pro`	80	4,979	7,624,534
Probe-price coefficient γ, three backbones	180	10,804	26,883,912
Memory discard threshold, three backbones	120	7,201	17,886,342
Validity-confidence threshold, `deepseek-v4-pro`	40	2,400	4,047,348
Total	870	52,386	99,031,778

tokens rather than a monetary API cost because the backbone models were accessed through a flat-rate account. A dollar cost cannot be assigned to the individual experiment families.

G Baseline Implementation Details

We compare DEC-MEM with a no-memory control, general-purpose agent-memory mechanisms, and memory components adapted from ML engineer agent systems. For all baselines, we preserve the common episode loop, candidate hypotheses, experimental environment, backbone LLM, and episode budget. This isolates the effect of memory writing and retrieval. For a negative episode, baselines without an explicit cause-attribution module receive the episode outcome according to their original memory interface; they do not receive DEC-MEM's privileged ground-truth fields. Ground-truth validity and failure causes are accessed only by the evaluation code. DEC-MEM strictly does not observe these fields and must infer the cause from the execution log and probes.

Reproduction scope. We evaluate mechanism-level reproductions of the baselines within a common R&D loop, keeping the task sequence, candidate set, backbone model, and episode budget fixed. This isolates differences in memory writing and retrieval, while avoiding claims that the shared-loop implementations reproduce every detail of the original agent systems. We preserve each method's defining write and retrieval mechanism where the available paper or source code establishes it; changes required by the shared loop are stated below.

G.1 No Memory

The no-memory control retains no episode information. It has the same agent and R&D setup as the other baselines, but we do not write to or retrieve from memory storage across episodes.

G.2 Reflexion

We reproduce the bounded verbal-reflection mechanism of Reflexion (Shinn et al., 2023). Memory retains the three most recent free-text reflections. After a positive episode, it appends a reflection that the intervention helped; after a negative episode, it appends one that the intervention did not help and should be avoided. Each reflection includes the observed effect. Retrieval injects the entire buffer without similarity ranking. The shared Engineer performs the evaluation, replacing Reflexion's separate actor and evaluator roles.

G.3 ExpeL

We reproduce ExpeL's (Zhao et al., 2024) vote-weighted insight pool. Each insight has text and an integer importance count. A new insight starts at count two; a positive episode adds or upvotes the corresponding insight, whereas a negative episode downvotes it and removes it when the count reaches zero. Retrieval returns up to 12 task-matching insights in descending importance order. The shared R&D loop provides one trajectory per episode, so ExpeL's pairing of successful and failed trajectories for the same task is not reproduced.

G.4 A-MEM

We preserve A-MEM's (Xu et al., 2026b) outcome-agnostic note construction. A note stores its content, keywords, contextual description, and links to other notes; it has no success or failure label. Positive and negative episodes follow the same write rule: create a note recording the observed result, link it to the five nearest notes by keyword overlap, and update those neighbours' context and keywords. Retrieval supplies up to 12 notes whose keywords match the task. This is a mechanism-level adaptation of retrieval: keyword matching replaces the original embedding retrieval, and note text is generated within the shared interface rather than by A-MEM's original note-writing pipeline.

G.5 MEM0

Our Mem0 adaptation follows its ADD, NOOP, and DELETE operations for conflicting facts (Chhikara et al., 2025). Memory stores short facts keyed by task and intervention. A positive episode writes that the intervention helps; a negative episode writes that it does not help. An absent fact is added, an identical fact is left unchanged, and a contradictory stored fact is deleted before the new fact is added. Retrieval supplies up to 14 facts for the current task. Interpreting an experimental outcome as a conversational fact is our mechanism-level adaptation: Mem0 itself defines contradiction between facts, not a rule for interpreting R&D outcomes.

G.6 REASONINGBANK

Our implementation follows the success- and failure-derived strategy items described for ReasoningBank (Ouyang et al., 2026). An item retains a title, strategy text, outcome kind, task, and intervention. Positive episodes produce a strategy encouraging reuse of the intervention; negative episodes produce a failure item advising against it. Retrieval supplies the 12 most recent items for the current task, including their outcome kinds. We omit ReasoningBank's memory-aware test-time scaling to keep the episode budget fixed. This is a mechanism-level adaptation: the available source description establishes that both successes and failures yield memory items, but does not establish our particular avoidance wording for failure items.

G.7 MEMP

We represent procedural memory as scripts keyed by intervention, each with steps and an integer score (Fang et al., 2026). A positive episode creates or updates a script, revises its steps for the current allocation and metric, and increases its score. A negative episode creates no script and decreases the score of an existing one. Retrieval returns up to ten positive-score scripts in score order. The published Build–Retrieve–Update lifecycle motivates this implementation.

We implement the published Build–Retrieve–Update abstraction using outcome-labelled intervention scripts. The exact score update and script-creation rules are adapted to the shared R&D interface and are thus interpreted as a mechanism-level baseline.

G.8 AIDE

Because the controlled benchmark exposes a sequential memory interface rather than AIDE's original code tree, we implement the Journal's metric-based node retention and best-result selection within the shared loop. Each retained node contains a plan, result narrative, validation metric, and buggy flag. We therefore interpret this condition as the memory mechanism extracted from AIDE rather than a full reproduction of the original AIDE R&D system.

G.9 AIRA

We adapt AIRA's memory operator. Memory retains episode nodes and whether they arose from debugging. Both positive and negative episodes append a node. The simple-memory operator summarizes usable nodes and truncates the resulting context from the front when it exceeds the length limit; its node formatting distinguishes debugging from ordinary proposals. AIRA also defines an ancestral operator that follows a failure chain to a successful ancestor. Tree-dependent sibling

memory and crossover are omitted because the shared loop has no tree branches. Thus, operator formatting and truncation are reproduced, while the underlying episode structure is adapted.

G.10 MLEVOLVE

We reproduce the outcome-labelled write and opposed retrieval rules of MLEvolve's (Du et al., 2026) `GlobalMemoryLayer` (`MLEvolve`, `agents/memory/global_memory.py`). A record retains the development stage, plan, outcome label in $\{-1, 0, +1\}$, and parent and current validation metrics. A node is stored only when it is non-buggy and has a metric. Draft nodes receive label $+1$; subsequent changes receive $+1$, 0, or -1 according to whether their metric improves, matches, or worsens relative to the parent. Retrieval separately returns the two highest-ranked positive and two highest-ranked negative records, showing the metric transition for each. The write gate, labels, and opposed presentation are preserved; the original FAISS/BM25 retrieval is replaced with the shared lexical retrieval substrate.

H EVALUATION METRICS

H.1 FALSE-NEGATIVE LOCK-IN RATE

For each stream, the at-risk population contains every task–hypothesis pair that is valid under at least one reachable experimental configuration and is observed at least once with a negative outcome caused by HNS, CPT, or MSN. A pair is locked in when it is never proposed again after its first such confounded negative outcome. FNLR is the fraction of at-risk pairs that are locked in. If a stream contains no at-risk pair, its FNLR is undefined and the stream is excluded from the FNLR aggregate rather than assigned a value of zero.

H.2 FALSE-NEGATIVE RECALL RATE

FNF1 is macro-averaged over the four failure causes. For each $z \in \mathcal{Z}$, we compute precision and recall from negative episodes for which z is respectively the attributed or ground-truth cause, and then compute the classwise F1. FNF1 is the unweighted mean of the four classwise scores. Positive episodes are excluded because they have no failure cause.

H.3 CUMULATIVE PERFORMANCE GAIN AND PROBE COST

The cumulative performance gain (CPG) is the sum of positive episode effects, as in Section 3:

$$\mathrm{CPG} = \sum_{t=1}^{T} \max(y_t, 0).$$

Probe cost is the sum of the normalized costs of all counterfactual probes executed in the stream.

H.4 MEDAL SCORE FOR FIG. 3

In Fig. 3, we evaluate the performance of an R&D stream using its results across five runs. Each run is assigned one of four outcome levels: medal, above median, valid but below median, or invalid. We assign these levels scores of 3, 2, 1, and 0, respectively. The medal score is defined as

$$\mathrm{MS} = \frac{3n_{\mathrm{medal}} + 2n_{\mathrm{above}} + n_{\mathrm{valid}} + 0n_{\mathrm{invalid}}}{n_{\mathrm{medal}} + n_{\mathrm{above}} + n_{\mathrm{valid}} + n_{\mathrm{invalid}}}, \tag{19}$$

where n_{medal}, n_{above}, n_{valid}, and n_{invalid} denote the numbers of runs in the corresponding outcome categories. Since each R&D stream contains five runs, the denominator is always 5, and the medal score ranges from 0 to 3.

Table 13: DEC-MEM results by ML problem, pooled across four backbones (40 streams per problem). Values are mean ± standard deviation across contributing streams. CPG and probe cost include only the episodes for the indicated problem within each stream; probe cost is in the compute units used for the probing experiments.

ML problem	FNLR ↓	n_{FNLR}	FNF1 ↑	CPG ↑	Probe cost ↓
Adult	0.167 ± 0.381	24	0.242 ± 0.163	0.0000 ± 0.0001	1.12 ± 0.60
AG News	0.562 ± 0.512	16	0.696 ± 0.362	0.0105 ± 0.0149	2.01 ± 1.76
Bank Marketing	0.218 ± 0.299	39	0.738 ± 0.333	0.0074 ± 0.0057	1.51 ± 1.36
German Credit	0.259 ± 0.371	38	0.618 ± 0.338	0.0196 ± 0.0218	3.14 ± 1.72
IMDb	0.162 ± 0.342	34	0.531 ± 0.365	0.0132 ± 0.0251	1.34 ± 1.36
Jigsaw Toxic	0.322 ± 0.448	30	0.692 ± 0.355	0.1990 ± 0.1466	2.01 ± 1.71
SMS Spam	0.433 ± 0.458	15	0.816 ± 0.378	0.1462 ± 0.1860	1.13 ± 1.34
Telco Churn	0.425 ± 0.446	40	0.635 ± 0.393	0.0079 ± 0.0090	2.11 ± 1.29

Table 14: Failure-cause attribution across three backbones and ten seeds per backbone. Rows are ground-truth causes and columns are causes attributed by DEC-MEM. The *none* row contains negative observations produced by measurement jitter in cells with a positive underlying effect; these episodes have no ground-truth label among the four modelled causes.

	Attributed cause				
Ground truth	HYP	HNS	CPT	MSN	Episodes
HYP	68	54	28	10	160
HNS	2	345	2	20	369
CPT	55	61	91	12	219
MSN	0	1	0	0	1
None	3	2	4	24	33

I Supplementary Experimental Results

I.1 Per-Task Results

Table 13 reports DEC-MEM's results separately for each ML problem, pooled across the backbone models and ten matched seeds per backbone. Each metric is computed from the episodes for that problem within a stream. FNLR is undefined when a stream contains no eligible confounded false-negative pair for that problem; such streams are excluded from its mean rather than assigned a value of zero. The column n_{FNLR} gives the number of streams contributing to that mean. The per-problem FNLR estimates have small denominators: each stream contains only three or four episodes for a given problem, and most eligible stream–problem pairs contain one or two at-risk hypotheses. This explains the large FNLR standard deviations. FNF1 measures attribution across the failure causes.

I.2 Attribution by Failure Cause

We evaluate failure-cause attribution on negative R&D episodes. Across three backbones and ten seeds per backbone, DEC-MEM encountered 782 negative episodes out of 1,200 episodes. Table 14 shows their ground-truth and attributed causes; Table 15 reports precision, recall, and F1 for each of the four causes. The cause distribution reveals how causes of negative outcomes in real-world R&D processes are unbalanced. Among the 384 measured task–intervention–allocation cells, only three have a natural MSN failure, whereas HNS failures arise more commonly. Consequently, only one evaluated negative episode has MSN as its ground-truth cause. Its reported F1 is zero. HNS is identified more consistently ($\mathrm{F1} = 0.829$); CPT is sometimes missed, with 61 CPT episodes attributed to HNS and 55 attributed to HYP. We retain MSN in the four-cause macro average rather than silently removing the least-represented cause; we interpret the macro average together with the per-cause support and confusion matrix rather than as a stable estimate of four-cause attribution performance.

Table 15: Per-cause attribution on the same negative episodes. The 33 episodes labelled *none* contribute false positives to the predicted causes, but do not contribute to per-cause recall.

Cause	Episodes	Precision	Recall	F1
HYP	160	0.531	0.425	0.472
HNS	369	0.745	0.935	0.829
CPT	219	0.728	0.416	0.529
MSN	1	0.000	0.000	0.000
Macro average	749			0.458

Table 16: DEC-MEM ablations on DeepSeek-v4-pro. Entries are mean $\pm$ standard deviation over ten matched seeds. Probe cost is measured in the compute units used for counterfactual probing.

Variant	FNLR $\downarrow$	CPG $\uparrow$	FNF1 $\uparrow$	Probe cost $\downarrow$
DEC-MEM	0.392 ± 0.190	0.396 ± 0.230	0.375 ± 0.148	12.68 ± 4.08
Oracle attribution	0.232 ± 0.138	0.520 ± 0.303	0.712 ± 0.027	0.00 ± 0.00
Execution log only	0.441 ± 0.103	0.408 ± 0.283	0.290 ± 0.087	0.00 ± 0.00
Probes without execution log	0.497 ± 0.129	0.324 ± 0.270	0.396 ± 0.125	14.66 ± 4.21
Harness control replay only	0.335 ± 0.115	0.321 ± 0.261	0.333 ± 0.064	4.35 ± 0.56
Compute-scaling probe only	0.411 ± 0.099	0.345 ± 0.235	0.359 ± 0.089	29.77 ± 7.68
Measurement check only	0.456 ± 0.161	0.274 ± 0.240	0.334 ± 0.073	0.80 ± 0.11
Without cause-specific evidence routing	0.520 ± 0.129	0.335 ± 0.263	0.391 ± 0.081	9.88 ± 3.09
Without frontier-aware retrieval	0.790 ± 0.107	0.131 ± 0.114	0.395 ± 0.056	16.63 ± 4.17

Remark on MSN coverage. Please note that the low MSN frequency reflects the empirical distribution of causes for negative outcomes of real-world ML R&D processes, rather than our controlled benchmark neglecting this failure mode. In fact, MSN is included as a first-class cause in our memory mechanism: we conduct a measurement probe to re-evaluate the same outputs under alternative metrics to carefully identify if an R&D episode has a negative outcome due to this cause.

I.3 Additional Ablations

In this section, we further expand our ablation study on DEC-MEM. Table 16 removes individual components of DEC-MEM while keeping the DeepSeek-v4-pro backbone and ten matched seeds fixed. The oracle-attribution variant supplies the ground-truth failure cause and serves as an upper bound on attribution quality. The log-only variant performs no counterfactual probes, whereas the probe-without-log variant does not use execution-log evidence to initialize cause attribution. Each single-probe variant restricts evidence acquisition to the indicated counterfactual check. Removing frontier-aware retrieval produces the largest increase in FNLR, from 0.392 to 0.790, and lowers CPG from 0.396 to 0.131. Removing cause-specific evidence routing also increases FNLR, even though its FNF1 remains high: recognizing the attributed cause does not, by itself, ensure that the resulting evidence is written to the appropriate memory component.

I.4 Sensitivity Analyses

We vary one setting at a time on DeepSeek-v4-pro Xu et al. (2026a) using ten matched seeds. Table 17 reports FNLR and CPG alongside FNF1, which directly reflects the parameter's effect on attribution. The γ analysis appears in the main paper. The tested discard thresholds produce identical stream-level results. Likewise, increasing ν beyond two and changing λ between one and two do not alter these results. Raising the fault-injection rate substantially reduces CPG, as a larger share of failures is caused by the more readily detected harness faults.

J Additional Case-Study Details

We report complete node-level trajectories for two illustrative streams shown in Figure 6. Each setting was run for eight streams, with up to 60 R&D episodes per stream under a three-hour compute

Table 17: Sensitivity of DEC-MEM on DeepSeek-v4-pro Xu et al. (2026a), reported as mean ± standard deviation over ten seeds. The fault rate is the probability of an injected harness failure.

Parameter	Value	FNLR ↓	CPG ↑	FNF1 ↑
Discard threshold θ	0.10	0.392 ± 0.190	0.396 ± 0.230	
	0.20	0.392 ± 0.190	0.396 ± 0.230	
	0.50	0.392 ± 0.190	0.396 ± 0.230	
Refuting scopes ν	1	0.355 ± 0.083	0.318 ± 0.218	
	2	0.392 ± 0.190	0.396 ± 0.230	
	4	0.392 ± 0.190	0.396 ± 0.230	
Frontier exponent λ	0	0.392 ± 0.190	0.409 ± 0.229	
	1	0.392 ± 0.190	0.396 ± 0.230	
	2	0.392 ± 0.190	0.396 ± 0.230	
Retrieval budget B	5	0.385 ± 0.214	0.518 ± 0.252	
	10	0.392 ± 0.190	0.396 ± 0.230	
	20	0.334 ± 0.158	0.332 ± 0.225	
Fault-injection rate	0.15	0.445 ± 0.216	0.415 ± 0.254	
	0.30	0.392 ± 0.190	0.396 ± 0.230	
	0.60	0.377 ± 0.140	0.170 ± 0.132	

budget. The task is toxic comment classification. In Tables 18 and 19, "parent" identifies the node from which AIDE developed the next solution; a dash marks the initial node. The reported score is macro-F_1. A BUGGY node has no usable score. The first node in each stream was affected by a harness fault that permuted 85% of the training labels. The trajectories show how the final solutions were developed. AIDE with DEC-MEM reaches 0.9099 at node 41 through class-weight rebalancing, which is directly derived by revisiting the initial false-negative episode thanks to our proposed DEC-MEM. Based on that node, AIDE w/ DEC-MEM improves that solution with sublinear term-frequency scaling at node 42 and word n-grams at node 45, reaching optimal performance at 0.9192. Meanwhile, vanilla AIDE quits on the node with a confounded negative outcome and continues exploring based on suboptimal hypotheses and reaches its best score of 0.8842 at node 33, which is lower than 0.9192 produced by AIDE with DEC-MEM.

K PROMPTS AND OUTPUT SCHEMAS

In the evaluated implementation, the Researcher Agent and Engineer Agent receive language-model prompts. Execution-log detection, counterfactual-probe interpretation, memory writing, and retrieval are implemented as rules rather than additional language-model calls. We specify the two agent prompts below and then describe the rules and recorded outputs. Boxed material in this appendix is the implemented format itself: prompt templates, output schemas and example records. Boxes in Appendix L instead reproduce text recovered from a recorded run.

K.1 RESEARCHER AGENT

The system prompt is:

Researcher Agent system prompt (template)

```
You are the Researcher Agent in an autonomous ML research system.  You
choose which single intervention to test next in order to maximise
cumulative discovery across many experiments.  Prefer directions
whose outcome is genuinely uncertain over directions already settled.
Always answer with a single JSON object and nothing else.
```

The user prompt substitutes the task, episode number, retrieved memory context, and available candidate interventions:

Table 18: Complete AIDE with DEC-MEM trajectory for the selected stream. Node 45 obtains the best macro-F_1 of 0.9192. Nodes 4 and 5 are debug descendants inferred from the supplied tree trace.

Node	Parent	Hypothesis or research action	Macro-F_1
0	–	char_ngrams	0.5216
1	0	cross_validation	0.7746
2	0	class_weight_balanced	0.7978
3	0	cross_validation	BUGGY
4	3	cross_validation	BUGGY
5	4	early_stopping	BUGGY
6	0	large_vocab	0.7835
7	0	max_df_lower	0.8124
8	0	min_df_raise	0.7776
9	7	sublinear_tf	0.8124
10	9	threshold_tuning	0.8124
11	9	word_ngrams	0.8124
12	9	max_df_lower	0.8124
13	0	min_df_raise	0.8124
14	0	word_ngrams	0.8124
15	0	char_ngrams	0.8124
16	0	threshold_tuning	BUGGY
17	0	large_vocab	0.8163
18	17	class_weight_balanced	0.8186
19	18	weak_regularisation	0.8219
20	19	weak_regularisation	0.8453
21	20	sublinear_tf	0.8111
22	0	max_df_lower	0.8072
23	0	min_df_raise	0.8124
24	0	word_ngrams	0.8124
25	0	char_ngrams	0.8124
26	0	threshold_tuning	0.8395
27	0	large_vocab	0.8468
28	0	weak_regularisation	0.8446
29	0	class_weight_balanced	0.8591
30	0	naive_bayes	0.8276
31	0	sublinear_tf	0.7909
32	0	max_df_lower	0.8121
33	0	min_df_raise	0.8529
34	0	weak_regularisation	0.8367
35	0	word_ngrams	0.8448
36	0	naive_bayes	BUGGY
37	0	char_ngrams	0.7897
38	0	cross_validation	0.8417
39	0	threshold_tuning	0.8535
40	0	large_vocab	0.8840
41	0	class_weight_balanced	0.9099
42	41	sublinear_tf	0.9156
43	42	max_df_lower	BUGGY
44	42	min_df_raise	0.9088
45	42	word_ngrams	**0.9192**
46	45	char_ngrams	0.8690

Table 19: Complete vanilla AIDE trajectory for the selected stream. Node 33 first reaches its best macro-F_1 of 0.8842; nodes 35 and 37 match that score.

Node	Parent	Hypothesis or research action	Macro-F_1
0	–	`char_ngrams`	0.4979
1	0	`cross_validation`	0.8094
2	1	`early_stopping`	0.8262
3	2	`class_weight_balanced`	BUGGY
4	2	`large_vocab`	0.8322
5	4	`max_df_lower`	0.8515
6	5	`min_df_raise`	0.8613
7	6	`pca`	0.8432
8	6	`naive_bayes`	0.8658
9	8	`scale_pos_weight`	0.8698
10	9	`standard_scaling`	0.8754
11	10	`calibration`	0.8714
12	10	`sublinear_tf`	0.8754
13	10	`linear_svm`	0.8723
14	10	`threshold_tuning`	BUGGY
15	10	`word_ngrams`	0.8715
16	10	`early_stopping`	0.8604
17	10	`max_df_lower`	0.8803
18	17	`standard_scaling`	0.8645
19	17	`min_df_raise`	0.8714
20	17	`scale_pos_weight`	0.8728
21	17	`naive_bayes`	0.8738
22	17	`char_ngrams`	0.8770
23	17	`large_vocab`	0.8746
24	17	`threshold_tuning`	0.8671
25	17	`class_weight_balanced`	0.8769
26	17	`sublinear_tf`	0.8785
27	17	`pca`	0.8786
28	17	`word_ngrams`	0.8729
29	17	`cross_validation`	0.8795
30	17	`linear_svm`	0.8828
31	30	`early_stopping`	0.8730
32	30	`max_df_lower`	0.8803
33	30	`standard_scaling`	**0.8842**
34	33	`min_df_raise`	BUGGY
35	33	`scale_pos_weight`	0.8842
36	33	`naive_bayes`	0.8823
37	33	`char_ngrams`	0.8842

Researcher Agent user prompt (template)

```
TASK: {task}
EPISODE: {episode_id}

MEMORY OF PRIOR EPISODES:
{memory_context}

CANDIDATES: {list of candidate interventions}

Choose exactly one intervention from CANDIDATES to test next.
Do not choose something the memory shows is already settled
unless you have a specific reason to re-test it under
different conditions.
Answer JSON: {intervention: <one of CANDIDATES>,
              rationale: <one sentence>}
```

The response contains an `intervention` string and a `rationale` string. An out-of-list intervention is first matched without regard to letter case; if matching fails, the runner selects a candidate uniformly at random.

K.2 Engineer Agent

The system prompt is:

Engineer Agent system prompt (template)

```
You are the Engineer Agent in an autonomous ML research system.  You
choose the compute budget and evaluation metric for one experiment.
Always answer with a single JSON object and nothing else.
```

The user prompt is:

Engineer Agent user prompt (template)

```
EXPERIMENT: test '{intervention}' on task '{task}'.
RATIONALE: {rationale}

RELEVANT MEMORY:
{memory_context}

Choose a compute budget and an evaluation metric.
budget: one of xs (0.25x), sm (0.5x), md (1x), lg (2x).
Larger budgets cost proportionally more; you have a finite total.
metric: 'primary' or 'secondary'.
Answer JSON: {budget: ..., metric: ...}
```

The response contains a `budget` in {`xs`, `sm`, `md`, `lg`} and a `metric` in {`primary`, `secondary`}. Out-of-vocabulary values fall back to `sm` and `primary`, respectively. Please note that the budget multipliers in the Engineer Agent prompt describe *normalized episode cost*, not the fraction of training data used. For `xs`, `sm`, `md`, and `lg`, the assigned costs are respectively 0.25, 0.5, 1.0, and 2.0 times that of a standard run. The corresponding fractions of the training split used by the experimental pipeline are 0.03, 0.10, 0.35, and 1.00. The prompt presents costs because the Engineer Agent must choose an allocation under a finite compute budget. The wider differences in training-data fractions also make effects that emerge at larger allocations detectable by the compute-scaling probe.

Table 20: Specified likelihood that an execution-log detector fires under each failure cause.

Log signal	HYP	HNS	CPT	MSN
Out-of-memory abort	0.02	0.35	0.55	0.02
Timeout	0.03	0.20	0.60	0.03
Non-finite loss	0.02	0.70	0.05	0.02
Shape mismatch	0.01	0.75	0.02	0.02
Empty features	0.02	0.65	0.03	0.03
Degenerate predictions	0.08	0.45	0.20	0.30
Data-split anomaly	0.02	0.10	0.03	0.70
Identical artifact hash	0.01	0.90	0.01	0.01

Table 21: Specified likelihood of a positive counterfactual-probe result under each failure cause. The initial cause prior is $(0.40, 0.30, 0.20, 0.10)$ in HYP–HNS–CPT–MSN order.

Probe	HYP	HNS	CPT	MSN
Harness control replay	0.05	0.85	0.06	0.05
Compute-budget trend check	0.14	0.15	0.55	0.15
Direct compute-scaling check	0.05	0.08	0.90	0.08
Measurement check	0.07	0.06	0.10	0.85

K.3 Execution-Log Evidence and Counterfactual Probes

A firing execution-log detector updates the prior belief over HYP, HNS, CPT, and MSN according to its likelihoods in Table 20. A detector that does not fire contributes no likelihood update. These likelihoods are specified in advance, rather than fitted from the evaluation episodes.

Executed probes produce Boolean evidence, which updates the same belief by likelihood multiplication; no language model interprets their results. Table 21 gives the likelihood of a positive probe result. A probe that is skipped supplies no observation and does not update the belief.

K.4 Episode Record and Memory Context

The following record illustrates the fields saved for a positive R&D episode. The keys `true_cause`, `recoverable`, and `fault_kind` are evaluation annotations: neither agent nor the memory mechanism receives them as privileged inputs.

Episode record: fields saved for a positive episode

```
{
  episode_id: 13,
  task: jigsaw_toxic,
  intervention: class_weight_balanced,
  budget: sm,
  metric: primary,
  observed_effect: 0.296414,
  positive: true,
  true_cause: none,
  attributed_cause: none,
  recoverable: true,
  fault_kind: ,
  posterior: {},
  probes: [],
  probe_cost: 0.0,
  memory_writes: [support:c0015],
  llm_calls: 0,
  llm_tokens: 0
}
```

Here, `posterior` maps failure causes to probabilities when attribution is performed; `probes` records the probes executed in that episode; and `memory_writes` identifies the resulting memory updates. Both `posterior` and `probes` are empty for the positive example above. Memory writing and frontier-aware retrieval are implemented deterministically. The retrieved context, rather than the internal memory store, is inserted into the {`memory_context`} slot of both agent prompts. The following MSN-attributed episode illustrates the cause-specific write policy. Episode 17 tests whether `tfidf_sublinear` improves the primary metric on AG News at allocation `sm`. Its observed effect is -0.012742, and the estimated cause distribution is

$$P_{17}(z) = (0.1854,\ 0.1192,\ 0.1325,\ 0.5629)$$

in HYP–HNS–CPT–MSN order. Since MSN is the most probable cause, DEC-MEM retains the general hypothesis at validity confidence $s_t = 0.5$ and writes the negative evidence to a claim restricted to the primary metric. The resulting metric-scoped memory item (i.e., the evidence may not apply to evaluations using other metrics) is:

Metric-scoped memory item written after episode 17

```
{
  claim_id: c0021,
  h_t: tfidf_sublinear improves primary on agnews,
  scope: agnews|*|*|primary,
  s_t: 0.0878,
  retracted: false,
  n_refuting_scopes: 1,
  E_t: {
    positive: [],
    negative: [{
      episode_id: 17,
      scope: agnews|*|sm|primary,
      positive: false,
      effect: -0.012742,
      cause_posterior: {
        HYP: 0.1854,
        HNS: 0.1192,
        CPT: 0.1325,
        MSN: 0.5629
      },
      attributed_cause: MSN,
      reliability: 0.6877,
      artifact_hash: a1d9e9792552
    }]
  },
  provenance: [split-from:c0020:measure]
}
```

Here, a wildcard in the scope means that the claim is not restricted along that dimension. The negative result applies to the tested allocation and primary metric. It lowers confidence in the metric-scoped claim, while the parent claim, `c0020`, remains unretracted at $s_t = 0.5$ with no negative evidence attached. Thus, the result does not rule out the hypothesis under other evaluation settings.

HNS follows a different route. In episode 11, the negative outcome for `early_stopping` on German Credit has HNS as its most probable cause, with $P_{11}(\text{HNS}) = 0.5472$. The hypothesis remains at $s_t = 0.5$ without negative evidence. DEC-MEM instead writes a procedural note:

Procedural note written at episode 11

```
[ep11] harness failure while testing 'early_stopping' on
german_credit:  verify the control before trusting a negative result
here.
```

This note preserves the diagnosed harness problem for a later R&D episode without treating the failed execution as a refutation of the hypothesis.

L Qualitative Evidence and End-to-End Memory Traces

We now follow one R&D stream through cause attribution, memory writing, and retrieval. The example shows how a negative outcome can be attributed to the experimental harness without demoting the hypothesis, and how the hypothesis can be re-examined. We also compare the same negative episode across memory mechanisms and give examples of compute-budget and performance-measurement causes. The episode comes from the controlled benchmark in Appendix E.

L.1 How Each Cause Changes Memory

DEC-MEM distinguishes four possible causes of a negative episode, as defined in Remark 1. Table 22 summarizes the evidence and the corresponding cause-specific writing policy. The frequency of each attributed cause is reported separately in Appendix I.2.

Table 22: Evidence and memory update for each possible cause of a negative outcome. Execution-log detectors d_i require no additional experiment; counterfactual probes u_j incur compute costs.

Cause	Interpretation	Evidence used in attribution	Cause-specific write
HYP	Hypothesis does not improve performance	Log and probe evidence favors HYP	Attach negative evidence; lower s_t
HNS	Harness affected the observed outcome	Control replay supports HNS	Record a procedural note; leave s_t unchanged
CPT	Compute budget was insufficient	Budget trend or direct scaling probe	Refute only a budget-scoped child claim
MSN	Evaluation metric obscured the effect	Re-score stored outputs under another metric	Refute only a metric-scoped child claim

An HYP attribution lowers the validity confidence of the hypothesis. An HNS attribution records a harness problem without changing that confidence. For CPT and MSN, the negative evidence applies to a child memory item restricted to the tested budget or metric; the more general parent hypothesis remains available for re-examination under other conditions.

L.2 A False-Negative Hypothesis Re-Examined

Consider a DEC-MEM stream using the `deepseek-v4-pro` (Xu et al., 2026a) backbone (6 seeds), and the default hyperparameters. Each memory item receives an identifier such as `c0009`; the allocation labels `xs`, `sm`, and `md` denote compute-budget settings. In episode 7, the agent tested the hypothesis represented by `class_weight_balanced` on Telco Churn. The Engineer Agent chose allocation `xs` and the primary metric. The observed effect was $y_7 = -0.044129$, so the episode had a negative outcome. No execution-log detector fired. Consequently, the initial distribution over causes remained at its specified prior, $P_0(z) = (0.40,\ 0.30,\ 0.20,\ 0.10)$, in HYP–HNS–CPT–MSN order. HYP has the largest prior probability, but the log provides no evidence for attributing this particular failure to the hypothesis. DEC-MEM therefore considered proceeding to perform counterfactual probing as described in Section 4.2.

Table 23: Counterfactual probes selected in episode 7, in execution order. After the second probe, no remaining probe exceeded the stopping threshold in Eq. 1.

Order	Probe	Cost c_j	Outcome u_j
1	`P3_metric_swap`	0.05	did not fire
2	`P1_control_replay`	0.25	fired
	Total	0.30	

The first probe did not support MSN. However, the second probe generated supportive evidence for HNS. Updating the cause distribution with both outcomes yielded

$$P_7(z) = (0.0689,\ 0.8883,\ 0.0400,\ 0.0028).$$

HNS is therefore the most probable cause. An HYP write would treat this confounded negative outcome as evidence against the hypothesis. Instead, DEC-MEM recorded `procedural_only:c0009`: the harness problem was stored as a procedural note, while claim `c0009` kept its initial validity confidence $s_t = 0.50$ and had no supporting or refuting evidence attached to its hypothesis. This follows the cause-specific writing policy in Section 4.3.

Before episode 15, frontier-aware retrieval made c0009 available to the Researcher Agent. The following excerpt reproduces the retrieved memory context; the omitted scope cells are marked explicitly. Here, kappa=0.50 corresponds to the validity confidence $s_t = 0.50$.

Retrieved memory context before episode 15 (verbatim, truncated where marked)

```
ESTABLISHED CLAIMS (high or low confidence):
  - [c0009] class_weight_balanced improves primary on telco_churn | scope=telco_churn
    |*|*|* | kappa=0.50 | support=0 refute=0
  - [c0001] class_weight_balanced improves primary on adult | scope=adult|*|*|* | kappa
    =0.50 | support=0 refute=0
  - [c0003] class_weight_balanced improves primary on agnews | scope=agnews|*|*|primary |
    kappa=0.07 | support=0 refute=1
OPEN QUESTIONS (maximally uncertain, least explored):
  - [c0009] class_weight_balanced improves primary on telco_churn | scope=telco_churn
    |*|*|* | kappa=0.50 | support=0 refute=0
UNEXPLORED SCOPE CELLS (no evidence at all):
  - bagging_ensemble @ budget=xs, metric=primary
  [... 95 further unexplored cells omitted ...]
PROCEDURAL NOTES (harness failures, not refutations):
  - [ep8] harness failure while testing 'class_weight_balanced' on adult: verify the
    control before trusting a negative result here.
  - [ep13] harness failure while testing 'class_weight_balanced' on jigsaw_toxic: verify
    the control before trusting a negative result here.
```

The same memory item appears in the general memory storage and the items for exploration (termed "open questions"). Its zero evidence counts and $s_t = 0.50$ show that DEC-MEM did not refute the hypothesis; the exploration branch can still prioritize it because its validity is uncertain and it has not been examined successfully. The separate procedural-notes block illustrates how harness failures are passed to future episodes without demoting the memory item. The metric-scoped claim c0003, by comparison, has one negative observation attached with confidence 0.07. The Researcher Agent's recorded response referred to the open question:

Researcher Agent response at episode 15 (verbatim)

```
{intervention: class_weight_balanced,
 rationale: The open question c0009 for telco_churn has no supporting or
  refuting evidence yet, making its outcome genuinely uncertain and directly
  relevant to the current task.}
```

The Engineer Agent again chose allocation xs and the primary metric. This time the measured effect was $y_{15} = +0.008927$, and DEC-MEM wrote support:c0009. Thus, the hypothesis received another examination after a harness-confounded negative outcome and produced a positive episode.

L.3 Comparison at the Same Negative Episode

We compare memory mechanisms at episode 7 in streams sharing backbones, seeds, and harness-faults. Each tests the class_weight_balanced hypothesis on the task of Telco Churn when the injected zero_lr fault makes the outcome negative. The Engineer Agent's allocation varies across streams, so the measured effects are not identical.

Table 24: Memory updates at the confounded negative outcome in episode 7 (deepseek-v4-pro, seed 6; evaluator-known cause HNS). The last column reports if the same task–hypothesis pair is re-examined within the R&D stream for each memory mechanism.

Memory	Compute Allo.	Effect y_7	Write at episode 7	Re-examined later
DEC-MEM	xs	−0.044129	procedural note only	yes (ep. 15, +0.008927)
Memp	sm	−0.046175	decay	no
ExpeL	sm	−0.046175	ADD	no
A-MEM	sm	−0.046175	atom, 5 links	no
ReasoningBank	md	−0.035889	failure	no
Reflexion	sm	−0.046175	reflection	yes (ep. 15, 23)

Four of the five baseline streams did not re-examine this pair after episode 7. Reflexion is the only one that re-examined the hypothesis in episodes 15 and 23. However, it cannot attribute the real cause of the false-negative outcome. This example illustrates how a negative outcome can be followed by false-negative lock-in, which is measured by FNLR in Table 1.

L.4 Other Cause-Specific Writes

Two other negative episodes from the same DEC-MEM stream illustrate why a cause-specific write preserves the scope of its evidence. Appendix K gives the episode-record format and another example of an HNS procedural note.

CPT: insufficient compute budget. In episode 9, the hypothesis `tfidf_sublinear` on the AG News task received a measured effect $y_9 = -0.001169$ at compute allocation `md`. DEC-MEM ran the direct compute-scaling probe `P2b_budget_direct` at cost 2.3. The resulting cause distribution was $P_9(z) = (0.1013,\ 0.0194,\ 0.8728,\ 0.0065)$, in HYP–HNS–CPT–MSN order. Because CPT was most probable, the write `scope_split:c0010->c0011:budget=md` attached the negative observation to child memory item `c0011`, restricted to budget `md`. Parent memory item `c0010` remained at confidence $s_t = 0.50$, available for examination at other compute allocations.

MSN: performance measurement. In episode 1, the hypothesis `class_weight_balanced` on AG News had effect $y_1 = -0.000171$ under the primary metric. After the measurement probe, $P_1(z) = (0.2064,\ 0.0209,\ 0.1459,\ 0.6267)$; MSN was most probable. The write `scope_split:c0002->c0003:metric=primary` attached the refutation to child memory item `c0003`, restricted to the primary metric. In the episode 15 context above, that child has one refutation and $s_t = 0.07$, whereas parent `c0002` retains $s_t = 0.50$ for other metric scopes.

In both examples, the negative observation narrows what is known about the tested conditions; it does not establish that the general hypothesis is invalid. The HNS write in episode 7 similarly leaves validity confidence unchanged and records the harness failure as procedural evidence. A hypothesis can be removed only after it is re-examined multiple times with the cause attributed as "hypothesis."

L.5 Information Available During the Stream

The distinction between attributed causes and ground truth matters in this example. At runtime, DEC-MEM uses the task, selected hypothesis, compute allocation, metric, measured effect y_t, execution-log detector outputs d_i, and the outcomes and costs of selected probes u_j, c_j. The agents receive retrieved memory context from earlier episodes. The ground-truth fields `true_cause`, `recoverable`, and `fault_kind` in the episode records are reserved for evaluation; they are not inputs to cause attribution, memory writing, or retrieval. The oracle-attribution ablation in Fig. 4 is the only exception: it uses the ground-truth cause to estimate the benefit of perfect attribution. The evaluator records that episode 7 had an injected `zero_lr` harness fault. DEC-MEM did not observe that label. It inferred HNS from the probe outcomes after the log detectors provided no additional evidence. Thus the agreement between its attribution and the evaluator's HNS label is a result of our DEC-MEM memory mechanism, not information supplied to it.

L.6 Summary of the R&D streams

In this R&D stream, episode 7 had a confounded negative outcome. Our counterfactual probes indicated a harness problem (HNS). DEC-MEM recorded that problem without lowering the validity confidence of memory item `c0009`. Frontier-aware retrieval later returned the item to the Researcher Agent, which selected the hypothesis for re-examination in episode 15. That episode had a positive effect of 0.008927, contributing that amount to the stream's cumulative performance gain (CPG). The hypothesis was therefore not locked in after its confounded negative outcome. This R&D stream shows one successful HNS attribution and one later re-examination; it cannot show how often either occurs. Across ten matched seeds, removing frontier-aware retrieval raised mean FNLR from 0.392 to 0.790 and lowered mean CPG from 0.396 to 0.131 (Table 16). The aggregate results in Section 5 and the per-cause results in Appendix I.2 provide the broader comparisons.