Small Models Think Big: Toward Effective Memory Distillation for Small Co-Scientists

Stanford CS224N Custom Project

Jaanak Prashar*
Stanford University
jaanak@stanford.edu

Renn Su*
Stanford University
rrsu@stanford.edu

Summer Royal*
Stanford University
sroyal@stanford.edu

Abstract

Small language models hold potential for scientific reasoning, but their stateless nature prevents them from accumulating knowledge and strategies across queries. Throughout our study, we explore a pipeline of supervised fine-tuning (SFT) and Curriculum Learning on LAB-Bench multiple choice questions. We study the effects of training small models on when to use and distrust external memory through SFT on teacher-generated curriculum distillations on a retrieval corpus. Our best strategy (retrieval-synthesis) improves ProtocolQA by 31 points and LitQA2 by 5 points over the 7B zero-shot baseline. Further, we show that distilled memory can also be manipulated: targeted surface-form attacks resist both similarity-based and semantic write gate defenses, revealing a fundamental limit of retrieval-based filtering when the attack exploits the signal the retrieval system relies on. Overall, these results show that effective memory distillation for small co-scientists requires learning when to use memory and when to distrust it.

1 Key Information to include

Our TA mentor is Mirac Suzgun. We have no external collaborators and no external mentors. We are not sharing this project with any other class. **We are using 3 late days for this report**. Jaanak has 0 late days remaining, Renn has 5, and Summer has 6 for a total of 11 pooled late days.

2 Introduction

Large language models (LLMs) have recently shown a strong performance across a wide range of reasoning and scientific question-answering tasks. Co-scientist language models have been used in many cases to solve problems in bioinformatics and biology research, showing promise when applied to protein prediction [1], gene expression analysis, and drug discovery [2]. However, modern LLMs are "stateless," meaning that each input is processed in isolation [3]. As such, these models have no memory of previous successes or previous failures and continue to make the same mistakes and fail to reuse previous discoveries. This is particularly inefficient in domains that require cumulative reasoning or reusable strategies.

One way we can address this limitation of LLMs is to augment them with external memory. Dynamic Cheatsheet (DC) introduced a test-time memory mechanism that retrieves relevant prior examples using fixed embedding-similarity rules and a predetermined top-k retrieval strategy [4]. These static mechanisms assume that memory should always be consulted and that the same retrieval policy is appropriate for every query. However, irrelevant or weakly related memory entries can degrade performance, especially for tasks that require thorough reasoning or structured knowledge.

We ask the following research question: can small language models learn when to use external memory via supervised distillation (rather than relying on static heuristics)? Ideally, a model would selectively retrieve relevant prior knowledge, avoid unnecessary context, and decide when new information should be written to memory. We expected that this adaptive memory would be useful

for smaller "co-scientist" language models that have limited context budgets and model capacity. In this work, our contributions include the following:

1. A curriculum-guided SFT pipeline that teaches small co-scientist models to reason with external memory, achieving up to $+31$ points on ProtocolQA using a 7B model trained against a 32B teacher.
2. An empirical analysis showing that demonstration strategy determines whether memory helps or hurts: SFT + curriculum outperforms both curriculum-only and SFT-only conditions, and gains are concentrated on tasks grounded in fixed reference corpora.
3. A memory poisoning evaluation with two attack types and two write gate defenses, showing that surface-form attacks resist similarity-based filtering while semantic verification provides partial but incomplete recovery.

3 Related Work

Retrieval-Augmented Generation and Curriculum Learning. Retrieval-augmented generation augments language model outputs with documents retrieved at inference time [5], but applies fixed retrieval policies with no mechanism for deciding when retrieval is helpful or harmful. Curriculum learning improves generalization by ordering training examples from simple to complex [6]. Our work combines both ideas, constructing task-specific curriculum memory banks and training small models via SFT on teacher-generated demonstrations drawn from those banks.

Knowledge Distillation. Distilling reasoning capabilities from large to small language models via chain-of-thought supervision has shown promise on reasoning benchmarks [7, 8]. These approaches distill reasoning traces but do not address memory interaction. We extend the distillation paradigm to memory use, training a small model on teacher-generated demonstrations of when to use, ignore, or distrust retrieved entries. On the other hand, several proposed methods have focused on solving the "statelessness" limitation of language models using inference-time memory solutions. Suzgun et al. (2025) proposed a self-maintained memory architecture to accumulate learning strategies, though improvement was more limited on smaller language models [4]. Frameworks such as Reflexion allow for verbal memory and learning from past experiences and learning [9]. However, they do not use distillation or finetuning processes, so our work fills the gap of the distillation of memory and curriculum on smaller models.

4 Approach

Our approach addresses a central question: does the source and structure of the external memory bank affect what a student model learns to do with that memory under SFT? We study this by holding the student model and training procedure fixed while exploring 1) the content of the memory bank and 2) the strategy used to generate teacher demonstrations from it 3) the effects of memory poisoning.

For our project specifically, we use curriculum-based memory architecture to perform supervised finetuning (SFT) using a "teacher" model (`Qwen2.5-32B`) to augment the performance of the "student" small language model (`Qwen2.5-7B`) performance. Namely, we evaluate how curriculum-based approaches affect memory use and strategy curation.

4.1 Memory Construction

Curriculum Curation. For our curriculum selection, we curated domain-relevant reference materials aligned with the knowledge requirements of each LAB-Bench subtask. Sources were chosen to cover the core concepts appearing in benchmark questions and to reflect standard biological references used in the literature. Two group members independently reviewed and cross-verified the selected sources to ensure relevance and coverage. See Appendix for additional details.

Retrieval Corpus. Rather than using a generic retrieval corpus, we construct task-specific curriculum memory banks for each LAB-Bench subtask. For each domain, we curate a set of authoritative reference documents (detailed in the Appendix), chunk them into overlapping 400-word segments with a 50-word stride, and embed each chunk using `sentence-transformers/all-MiniLM-L6-v2`.

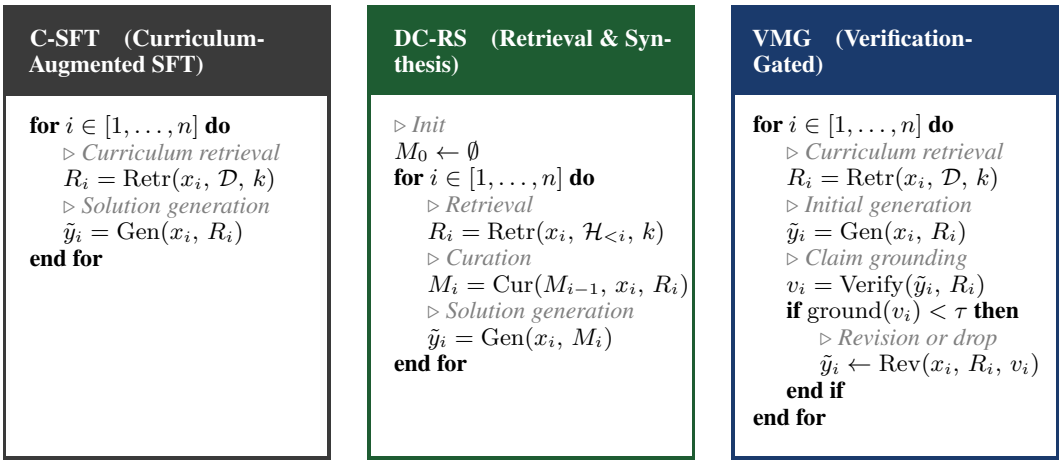

Figure 1: Teacher data generation strategies (`Qwen2.5-32B` teacher, `Qwen2.5-7B` student). All three share the same output schema, enabling controlled comparison.

At retrieval time, a query is embedded with the same model and the top-k chunks are retrieved by cosine similarity, with the query excluded from its own results.

Formally, if we let $\mathcal{D} = \{d_1, \ldots, d_m\}$ be the set of curriculum chunks for a given domain, each embedded as $\mathbf{e}_j \in \mathbb{R}^d$. For a query x_i with embedding $\mathbf{q}_i$, the retrieved memory is:

$$R_i = \text{top-}k\left(\left\{\frac{\mathbf{q}_i \cdot \mathbf{e}_j}{\|\mathbf{q}_i\|\|\mathbf{e}_j\|} : d_j \in \mathcal{D}\right\}\right) \tag{1}$$

Since irrelevant reasoning traces can degrade retrieval performance, we apply two filtering steps before writing entries to the memory bank: 1) schema validation to ensure outputs are not malformed, and 2) semantic similarity filtering for 'sufficiently similar' content (cosine ≥ 0.7) [10] to make sure that the generated reasoning remains relevant to the source question and requested subtask.

4.2 Teacher-Generated Demonstration Construction

We implement three teacher demonstration strategies, illustrated in Figure 1:

1. **Curriculum-Augmented SFT (C-SFT).** A single teacher call per example: the teacher receives the question and top-k curriculum passages R_i and produces a chain-of-thought trace and final answer letter. No memory of prior examples is maintained.
2. **Retrieval & Synthesis (DC-RS).** Adapted from the Dynamic Cheatsheet paper[4], following the same retrieve, curate, generate pattern with two modifications: 1) the running cheatsheet M_0 is bootstrapped from curriculum passages rather than initialized empty, and 2) the pipeline targets SFT data generation rather than inference. Per example, the top-k most similar prior solved pairs from $\mathcal{H}_{<i}$ are retrieved and synthesized with M_{i-1} into an updated cheatsheet M_i, which is passed to the generator alongside x_i.
3. **Verification Memory-Gated (VMG).** Extends (C-SFT) with a post-generation verification step: atomic claims in $\tilde{y}_i$ are checked for grounding against R_i, and traces below a specified threshold τ (default $\tau = 0.8$) are revised once or dropped.

All three strategies use `Qwen2.5-32B` as the teacher and share the same structured JSON output schema (entries used/ignored, chain-of-thought, final answer letter), enabling controlled comparison.

In DC-RS, the curator and generator roles use distinct system prompts on the same underlying model. The student never generates its own training traces. More information about for each strategy is provided in the Appendix.

4.3 Student Model Finetuning

We fine-tune Qwen2.5-7B [11] using causal language modeling on (system, user, assistant) triples. Loss is masked to the assistant turn only, so the model is trained purely on the reasoning and answer, not on the prompt or retrieved entries. To prevent "teaching to the test" and to promote learning generalizable reasoning patterns, teacher demonstrations were generated without conditioning on the multiple-choice options and the correct answer. This design discourages shortcut strategies that exploit the constrained answer space and instead encourages open-ended reasoning about the problem. During evaluation, models receive the answer options and produce a final answer letter, which is compared against the gold label from LAB-Bench. More specifically, our steps are as follows:

1. **Teacher data generation.** `Qwen2.5-32B` generates structured demonstrations under one of three strategies (C-SFT, DC-RS, VMG); outputs are filtered and written to a training corpus.
2. **Supervised Finetuning.** `Qwen2.5-7B` is finetuned on the filtered corpus via causal LM loss on the assistant turn only, with checkpoints saved per epoch.
3. **Inference with memory bank.** The SFT checkpoint is loaded alongside a DC-style curriculum memory bank; for each evaluation question the top-k passages are retrieved, the model generates a response, and we report accuracy, average retrieved k, and memory selectivity.

4.4 Baseline Conditions

We provide the zero-shot performance metrics of the student model (`Qwen2.5-7B`) and the teacher model (`Qwen2.5-32B`) to establish the raw performance of the base models. We additionally evaluate two baseline conditions that isolate the effects of retrieval and fine-tuning. Namely, in the *Curriculum Only* baseline, the base student model receives task-relevant curriculum passages at inference time but is not finetuned. In the *SFT Only* baseline, the student model is finetuned on teacher-generated reasoning traces where the teacher has no access to curriculum information. These latter two baselines allow us to measure the contributions of curriculum retrieval and supervised fine-tuning separately.

4.5 Memory Poisoning and Write Gate

Memory that can be learned can also be manipulated. As a result, we explore memory robustness against poisoning and the effectiveness of write gate guardrails. In particular, we ask: a) which lab-bench domains seem to have the most sensitivity to memory poisoning applied globally and why, and b) which memory poisoning techniques (i.e., factual corruption and cross-domain injection) result in the largest accuracy drops and are hardest to recover by the write gate.

Attack types. *P1 (targeted answer flip)* injects five adversarial entries at fixed positions $\mathcal{I} = \{3, 7, 11, 15, 19\}$ (16.7% of the 30-example evaluation set). Each entry reuses exact keywords from the target question, achieving BoW cosine ≈ 0.85, and corrupts the answer via one of three sub-strategies: direct answer-letter flip, numerical value corruption, or experimental method swap. *P2 (cross-domain injection)* inserts five DbQA entries (gene–disease vocabulary) into the LitQA2 memory; their sentence-transformer cosine to LitQA2 questions is ≈ 0.05–0.12.

Write gate defenses. *WG-Sim* filters retrieved entries by sentence-transformer cosine $\geq \tau = 0.20$, blocking P2 entries but not P1 (cosine ≈ 0.85). *WG-Teacher* prompts the model to reject entries that are off-domain or clearly factually wrong. A combined defense (WG-Sim followed by WG-Teacher on passing entries) is evaluated in the Appendix. More details on memory poisoning techniques can be found in the Appendix.

5 Experiments

5.1 Data

Benchmark. Our primary dataset is LAB-Bench [12], which contains 2,400+ multiple-choice biology questions across 8 task categories. We restrict our evaluation to the following text-based tasks: LitQA2, DbQA, ProtocolQA, SeqQA, and CloningScenarios. We exclude FigQA and TableQA (as they test visual reasoning), along with SuppQA (which require large PDFs).

Task-Specific Curriculum. Curriculum documents were curated per domain to serve as the external memory bank. Curriculum documents are chunked into overlapping 400-word segments (50-word stride) and embedded using `sentence-transformers/all-MiniLM-L6-v2` for retrieval. Details of the selected sources are listed in the Appendix.

5.2 Evaluation method

Memory Distillation. We measure subtask accuracy on LAB-Bench as our main quantitative metric (i.e., multiple choice matches to the "gold answer" of the benchmark). To assess the effect of our memory-based training strategies, we report the change in accuracy (Δ) relative to a zero-shot baseline using the same model without memory augmentation. We additionally report average retrieved memory size (k) to evaluate selectivity in memory use for each subtask.

Memory Poisoning. To evaluate robustness to corrupted memory entries, we report diagnostics including the a) number of injected entries ($n_{\text{injections}}$), b) the number of retrieved entries removed by the write gate prior to generation (gate_blocked), and c) the average cosine similarity between the question and retrieved entries when similarity-based filtering is used. These metrics help quantify both the impact of poisoning and the effectiveness of the write gate in filtering corrupted memories.

5.3 Experimental details

Our training step uses the HuggingFace `Trainer` with batch size 4, gradient accumulation over 8 steps (effective batch size 32), learning rate 2×10^{-5}, 3 epochs, and maximum sequence length 2048. We apply gradient checkpointing to reduce GPU memory usage and run training on Modal GPU infrastructure. The dataset is split 85%/15% into train and validation sets, with an explicit leakage check verifying zero overlap between splits. For subtasks with over 100 items, we deterministically subsample 100 items using the random seed 42 [1].

5.4 Results

5.4.1 Memory Distillation

Our SFT condition using DC-RS improves procedural reasoning tasks such as ProtocolQA (Δ =+31 points) and LitQA2 (Δ =+5 points), while verification-gated training (VMG) degrades performance on most tasks.

Task	32B Base	7B Base	C-SFT	DC-RS	VMG
LitQA2	38	32	33 (+1)	**37 (+5)**	22 (-10)
DbQA	25	14	7 (-7)	8 (-6)	**15 (+1)**
SeqQA	9	13	**16 (+3)**	9(-4)	3 (-10)
ProtocolQA	56	16	20 (+4)	**47 (+31)**	15 (-1)
CloningScenarios	3	3	0 (-3)	**3 (+0)**	3 (+0)

Table 1: Accuracy (%) on five LAB-Bench subtasks. Parenthetical values indicate percentage-point improvement over the Qwen2.5-7B no-memory baseline on the same task.

[1] The Answer to the Ultimate Question of Life, the Universe, and Everything is 42 - *The Hitchhiker's Guide to the Galaxy*

Across all three conditions, DbQA and CloningScenarios show no meaningful improvement: the best result on DbQA is VMG at 15% versus a 14% baseline (+1), and no condition exceeds the 3% CloningScenarios baseline. SeqQA is the only task where the finetuned 7B model outperforms the 32B teacher: C-SFT reaches 16% against the teacher's 9%. On the remaining tasks, no SFT condition exceeds the 32B teacher.

Ablation Studies. Can we achieve comparable improvements with curriculum without SFT or SFT without curriculum? We performed ablation studies to evaluate performance on tasks that isolate the independent effects of curriculum and SFT. We compare the *Curriculum Only* and *SFT Only* baselines against the `Qwen2.5-7B` base model zero-shot performance.

Task	n	7B Base	Curriculum Only	SFT Only
LitQA2	100	32.0	0.0 (-32.0)	34.0 (+2.0)
DbQA	100	14.0	0.0 (-14.0)	9.0 (-5.0)
ProtocolQA	100	16.0	1.0 (-15.0)	7.0 (-9.0)
SeqQA	100	13.0	1.0 (-12.0)	1.0 (-12.0)
CloningScenarios	33	3.0	0.0 (-3.0)	0.0 (-3.0)

Table 2: Ablation study isolating the contributions of curriculum retrieval and supervised fine-tuning (SFT). Values in parentheses denote the change in accuracy relative to the 7B zero-shot baseline.

The results in Table 2 suggest that neither component alone is sufficient. Providing curriculum passages at inference time without SFT (*Curriculum Only*) scores near zero or below baseline across all tasks. Finetuning without curriculum (*SFT Only*) produces a small gain on LitQA2 (+2.0) but hurts performance on DbQA (-5.0), ProtocolQA (-9.0), SeqQA (-12.0), and CloningScenarios (-3.0). Previous literature also suggests that smaller models experience limited benefits when DC-RS is applied at inference time [4, 13]. These ablation results suggest that curriculum, SFT, and DC-RS components alone do not cause the improvements we saw in Table 1.

5.4.2 Memory Poisoning and Write Gate

Experiment 1: Granularity sensitivity. Our results suggest that write gate recovery depends on the level of granularity involved in the poisoned memory. Namely, well-known biological facts that are poisoned are easily detectable compared to more niche, grounded literature claims due to model uncertainty. Across LitQA2, DbQA, and ProtocolQA, factual corruption caused accuracy drops of 20.0, 21.0, and 29.5 points respectively, with write gate recovery of 16.7, 16.3, and 23.6 points. These results suggest consistent partial recovery across domains, but with a persistent residual gap. For brevity, the full experimental details and figures from this can be found under the memory poisoning experiment #1 section of the appendix.

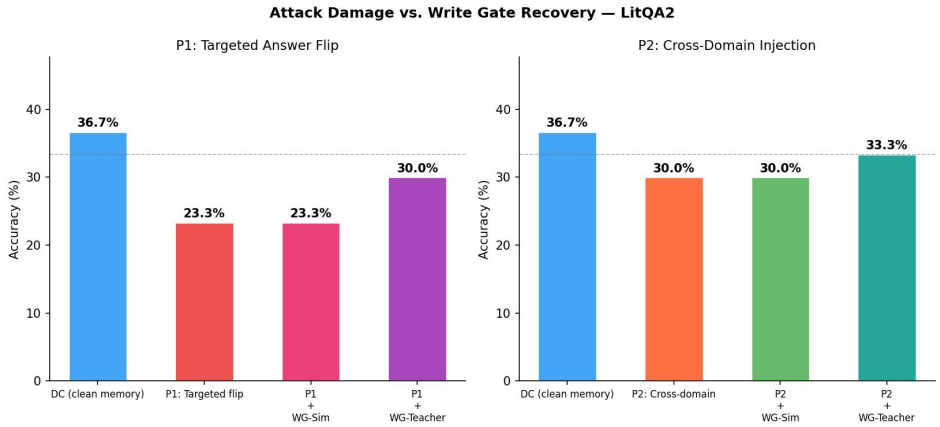

Figure 2: Memory poisoning performance for targeted answer flip and cross domain injection on DC inference on LitQA2

Experiment 2: Attack type and write gate comparison. We evaluated two attack types (P1: targeted answer flip, P2: cross-domain injection) against two defenses (WG-Sim, WG-Teacher) on LitQA2 using DC inference memory. We found that P1 is more damaging than P2 as it drops accuracy from 36.7% to 23.3%, compared to 30.0% by P2. WG-Sim cannot recover the P1 gap since injected entries achieve BoW cosine ≈ 0.85 and pass the similarity threshold by design. WG-Teacher provides partial recovery on both attacks, raising accuracy from 23.3% to 30.0% under P1 and from 30.0% to 33.3% under P2. WG-Sim fails to achieve the same recovery because cosine filtering blocks semantically related but embedding-distant entries alongside poisoned ones. WG-Teacher avoids this by reasoning about entry relevance directly, rather than relying on similarity scores. For a full outline on the technical specifications behind the target type and the write gate types, please see the Appendix.

6 Analysis

Output Pathologies. Manual inspection of incorrect outputs across LitQA2, DbQA, and CloningScenarios reveals that degradation under SFT is driven less by reasoning failure than by two structural issues: retrieval distribution mismatch and format breakdown (Table 3).

Type	Task / ID	Example output	Description
Empty response	LitQA2 (`litqa2_0002`)	*(no letter)*	No answer or blank space provided by the model.
Train/eval mismatch	LitQA2 (`litqa2_0021`)	`[ANSWER]\nA`	Trained with reasoning + tags; eval expects single token.
Cross-domain retrieval	DbQA (`dbqa_0002`)	`in Turtle format...`	DbQA eval uses LitQA2 bank (`litqa2_mem_*`); model continues off-domain RDF passage.
Sequence continuation	CloningScenarios (`0028`)	`ATG`	Model continues with DNA bases instead of a choice letter.
Context continuation	CloningScenarios (`0032`)	`igation-indep...`	Model outputs small token cloning fragment instead of choice letter.

Table 3: Qualitative failure taxonomy across LitQA2, DbQA, and CloningScenarios. Format failures and cross-domain retrieval account for the majority of degradation under SFT on DbQA and CloningScenarios.

Inheritance of Model-Level Failures. Failure modes such as Empty Responses and Sequence/Context Continuation are observable between both the zero-shot model and the post-finetuned models, suggesting that they may occur due to base model performance rather than downstream memory pipelines. [2] Inheriting thought processes for scientific heuristics does not result in learning the correct response format, such as for CloningScenarios, where a model-level tendency toward sequence continuation led to very low performance for the teacher and student model alike (0% - 3% accuracy).

Task Structure Determines Distillation Benefit. Our results suggest that sequential memory synthesis is beneficial when reasoning patterns can be reused, but explicit verification may remove useful demonstrations when grounding information is incomplete. Gains in ProtocolQA and LitQA2 may be explained by the extent to which broad reasoning patterns that transfer across questions. On the other hand, the tasks requiring precise lookup or algorithmic execution show less improvement. DbQA, SeqQA, and CloningScenarios all require precise, condition-specific knowledge such as facts from long DNA sequences or multi-step combinatorial reasoning. Notably, DbQA also requires live lookup for specific databases, which our SFT could not teach. With evaluation sets of 33 to 100 items, SFT provides insufficient exposure to learn these fine-grained rules reliably, and often does not act as a replacement to the tool-use needed for these tasks. SeqQA is a notable case where DC-RS specifically hurts (-4 points versus C-SFT's $+3$), as its codon translation tasks are deterministic procedures where accumulating prior examples adds no transferable signal, and the synthesized cheatsheet instead introduces irrelevant context.

[2] We did not use the Instruct variants of the Qwen 2.5 models due to compute limitations.

Verification Gating Penalizes Incomplete Curricula. VMG degrades performance on four of five tasks, most severely on LitQA2 (-10) and SeqQA (-10). The mechanism is not that verification is wrong in principle. We find that that verification at $\tau = 0.8$ may assume the curriculum is sufficiently complete to ground all relevant reasoning. When it is not, the verifier cannot distinguish between a demonstration that is factually correct but ungrounded in the retrieved passages and one that is genuinely wrong, possibly explaining the performance limitations. This same dynamic appears in our memory poisoning experiments, since WG-Teacher partially recovers both P1 and P2 attacks by reasoning about entry plausibility rather than surface similarity, but fails on niche domain-specific facts where model uncertainty causes it to pass poisoned entries through. Across our two experiments, we uncover a shared limitation where verification-based gating is bounded by base model confidence.

Knowledge Without Guidance is Insufficient. Throughout most of our experiments, the larger Qwen2.5-32B teacher baseline appears to be an upper bound for student performance after finetuning. For example, the weak teacher model performance across CloningScenarios did not result in any meaningful change. The exception is SeqQA, where SFT shows a small improvement over the teacher baseline (16% vs. 9%), suggesting that curriculum-guided supervision may allow the smaller model to internalize task-specific heuristics not reliably expressed by the teacher model's zero-shot responses. Our results suggest that providing a knowledge base is not enough for significant improvement on smaller models. The Curriculum Only condition makes this especially clear, as providing the student model with the same reference passages as SFT yields near-zero accuracy across all five tasks ($\Delta = -32.0$ on LitQA2, -14.0 on DbQA, -15.0 on ProtocolQA).

7 Conclusion

For our project, we investigated whether small language models can learn to use external memory effectively and safely through supervised fine-tuning on teacher-generated demonstrations. Our results suggest that the choice of demonstration strategy is the primary driver of performance, with the largest gains on tasks where answers are grounded in a fixed reference corpus rather than precise database lookup or algorithmic execution. Notably, curriculum distillation with DC-RS improves ProtocolQA by 31 points and LitQA2 by 5 points over the 7B baseline, closing 58% of the gap to the 32B teacher on ProtocolQA. On the safety side, we find that attack recovery depends on the retrieval mechanism an attack exploits. Cross-domain injection (semantic dissimilarity) is greatly recovered by WG-Teacher (+3.3%). Targeted surface-form attacks (high bag-of-words similarity) resist both defenses and retain a 6.7 point accuracy gap relative to clean memory.

In terms of limitations, compute constraints limited our teacher to a single teacher model (`Qwen2.5-32B`) and resulted in running smaller samples. Limited curriculum coverage due to file size, selection, and accessibility may have capped retrieval fidelity. Poisoning experiments were also conducted on LitQA2 alone, leaving domain/format generalization of attack sensitivity an open question. For future work, researchers could explore adaptive retrieval policies that learn when not to consult memory, explore alternative memory verification architectures, and generalize the pipeline to other model suites or stronger teacher models.

Team contributions

Summer: Reviewed questions in evaluation subtasks to find task-specific curriculum for ProtocolQA, CloningScenarios, and SeqQA, filtered curriculum for relevant content, built end-to-end SFT pipeline (chunk and embed curriculum texts into the memory bank, generate structured teacher demos, finetuned Qwen2.5-7B on training data), wrote the Intro, Data, Evaluation Method, Experimental Details, and Curriculum Selection sections of write-up, and made the poster.
Renn: Built the three SFT variant pipelines (C-SFT, DC-RS, VMG) for teacher memory, reporting and analyzing all relevant results and fine-tuning models on these results downstream. Constructed and analyzed all post-milestone updated baselines and ablations including SFT-Only, Curriculum-Only, and zero-shot baselines. Constructed teacher answer parsing and grading logic across tasks. Evaluation, debugging, and analysis of teacher model outputs. Task-specific curriculum documents for LitQA2 and DbQA.
Jaanak: Developed DC memory retrieval training pipeline, developed memory poisoning pipeline (memory poisoning verifier script, figures, write gate pipeline), contributed to methods, results, and appendix section.

References

[1] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *nature*, 596(7873):583–589, 2021.

[2] Oluwafemi A. Sarumi and Dominik Heider. Large language models and their applications in bioinformatics. In *Computational and Structural Biotechnology Journal*, 2024.

[3] Ahmed Salem, Andrew Paverd, and Sahar Abdelnabi. Stateless yet not forgetful: Implicit memory as a hidden channel in llms. *arXiv preprint arXiv:2602.08563*, 2026. Accepted at IEEE SaTML 2026.

[4] Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. *arXiv preprint arXiv:2504.07952*, 2025.

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

[6] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[7] Namgyu Ho, Laura Schmid, and Se-Young Yun. Large language models are reasoning teachers. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pages 14852–14882, 2023.

[8] Lucie Charlotte Magister, Jonathan Mallinson, Jakub Adamek, Eric Malmi, and Aliaksei Severyn. Teaching small language models to reason. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1773–1781, 2023.

[9] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

[10] Tristan JB Cann, Ben Dennes, Travis Coan, Saffron O'Neill, and Hywel TP Williams. Using semantic similarity to measure the echo of strategic communications. *EPJ Data Science*, 14(1):20, 2025.

[11] Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2025.

[12] Jon M. Laurent, Joseph D. Janizek, Michael Ruzo, Michaela M. Hinks, Michael J. Hammerling, Siddharth Narayanan, Manvitha Ponnapati, Andrew D. White, and Samuel G. Rodriques. Lab-bench: Measuring capabilities of language models for biology research. *arXiv preprint arXiv:2407.10362*, 2024.

[13] Mirac Suzgun. Personal communication, 2026. Email correspondence, February 2026.

[14] James D. Watson, Tania A. Baker, Stephen P. Bell, Alexander Gann, Michael Levine, and Richard Losick. *Molecular Biology of the Gene*. Pearson, 7th edition, 2014.

[15] Janet Piñero, Àlex Bravo, Núria Queralt-Rosinach, Alba Gutiérrez-Sacristán, Jordi Deu-Pons, Emilio Centeno, Javier García-García, Ferran Sanz, and Laura I Furlong. Disgenet: a comprehensive platform integrating information on human disease-associated genes and variants. *Nucleic acids research*, page gkw943, 2016.

[16] Online Mendelian Inheritance in Man. OMIM search help. `https://www.omim.org/help/search`. Accessed 2026.

[17] Ensembl. Ensembl training coursebook. `https://www.ensembl.org/info/website/tutorials/coursebook.pdf`. Accessed 2026.

[18] Yuhao Chen and Xiaowei Wang. mirdb: an online database for prediction of functional microrna targets. *Nucleic acids research*, 48(D1):D127–D131, 2020.

[19] Anthony S Castanza, Jill M Recla, David Eby, Helga Thorvaldsdóttir, Carol J Bult, and Jill P Mesirov. Extending support for mouse data in the molecular signatures database (msigdb). *Nature methods*, 20(11):1619–1620, 2023.

[20] Mouse Genome Informatics. Mouse genome informatics general searching help. `https://www.informatics.jax.org/userhelp/MISC_general_searching_help.shtml`. Accessed 2026.

[21] Broad Institute. MSigDB and GSEA user guide. `https://docs.gsea-msigdb.org/#GSEA/GSEA_User_Guide/`. Accessed 2025.

[22] Ivan Yevshin, Ruslan Sharipov, Semyon Kolmykov, Yury Kondrakhin, and Fedor Kolpakov. Gtrd: a database on gene transcription regulation—2019 update. *Nucleic acids research*, 47(D1):D100–D105, 2019.

[23] National Center for Biotechnology Information. ClinVar help documentation. `https://www.ncbi.nlm.nih.gov/clinvar/docs/help/`, 2024. Accessed 2026.

[24] Melissa J Landrum, Jennifer M Lee, Mark Benson, Garth Brown, Chen Chao, Shanmuga Chitipiralla, Baoshan Gu, Jennifer Hart, Douglas Hoffman, Jeffrey Hoover, et al. Clinvar: public archive of interpretations of clinically relevant variants. *Nucleic acids research*, 44(D1):D862–D868, 2016.

[25] Gorka Lasso, Sandra V Mayer, Evandro R Winkelmann, Tim Chu, Oliver Elliot, Juan Angel Patino-Galindo, Kernyu Park, Raul Rabadan, Barry Honig, and Sagi D Shapira. A structure-informed atlas of human-virus interactions. *Cell*, 178(6):1526–1541, 2019.

[26] Joseph Sambrook and David W. Russell. *Molecular Cloning: A Laboratory Manual*. Cold Spring Harbor Laboratory Press, 3rd edition, 2001.

[27] Carola Engler, Romy Kandzia, and Sylvestre Marillonnet. A one pot, one step, precision cloning method with high throughput capability. *PLoS ONE*, 3(11):e3647, 2008.

[28] Daniel G. Gibson, Lei Young, Ray-Yuan Chuang, J. Craig Venter, Clyde A. Hutchison, and Hamilton O. Smith. Enzymatic assembly of dna molecules up to several hundred kilobases. *Nature Methods*, 6(5):343–345, 2009.

[29] Addgene. Addgene: CRISPR Guide. `https://www.addgene.org/guides/crispr/`, 2015. Accessed: 2026-03-09.

A Appendix

Curriculum Selection: To train the model for LAB-Bench evaluation, we selected curriculum documents that aligned with all of the subtasks that we used to evaluate our model. We fine-tuned our model for each subtask separately. In other words, we found separate curriculum documents that are specific to each subtask, and we finetuned our model on each separate set of corpus documents before evaluating our model on each subtask. Table 4 below shows the subtasks on which we evaluated our model and the associated curricula that we used to finetune our model for that subtask.

From "Molecular cloning: A laboratory manual" [26], we included chapters that provide foundational knowledge for common molecular biology workflows. These chapters collectively cover major topics such as RNA isolation, bacterial transformation, PCR troubleshooting, transfection, and protein expression, which correspond to several of the wet-lab domains represented in ProtocolQA. For CloningScenarios, the following Sambrook chapters were included: Plasmid Vectors, Restriction Enzymes and DNA Modification, PCR, Introduction of Recombinant DNA into E. coli, Gel Electrophoresis and DNA Analysis, and DNA Sequencing. For the ProtocolQA subset of questions relevant to Sambrook (approximately 30% of the benchmark), we included chapters covering: RNA Isolation, cDNA Synthesis and Library Preparation, Introduction of Cloned Genes into Mammalian Cells, and Bacterial Transformation. We excluded Sambrook chapters focused on topics not

Subtask	Questions	Knowledge Type	Associated Curriculum
SeqQA	600	Algorithmic: find ORFs, translate codons	"Molecular Biology of the Gene" [14]
DbQA	520	Bioinformatics databases (OMIM, DisGeNet, UniProt)	Papers and documentation about the following databases related to example questions in the LAB-Bench DbQA subtask: DisGeNET [15] Online Mendelian Inheritance in Man [16] Ensembl [17], miRDB [18] Mouse Genome Informatics [19, 20] MSigD [21] GTRD [22] ClinVar [23, 24] P-HIPSTer [25]
LitQA2	199	Recent research papers (2023–2024)	LitQA2 Source Papers
ProtocolQA	108	Troubleshooting wet lab protocols	"Molecular cloning: A laboratory manual" [26]
CloningScenarios	33	Plasmid design, restriction enzymes, assembly	"Molecular cloning: A laboratory manual" [26], "A One Pot, One Step, Precision Cloning Method with High Throughput Capability" [27], "Enzymatic assembly of DNA molecules up to several hundred kilobases" [28], Addgene: CRISPR Guide [29]

Table 4: Task Breakdown (text-only subtasks) and the corresponding curricula.

represented in the benchmark tasks, including bacteriophage λ and M13 vectors, phagemids and single-stranded DNA systems, Southern/Northern blotting, phage-based library construction, and yeast two-hybrid methods.

Several widely used cloning technologies that appear frequently in LAB-Bench are not covered in depth in Sambrook. To address this gap, we added the following references:

1. Golden Gate Assembly: Engler et al., 2008 (PLoS ONE) [27], describing one-pot Type IIS restriction enzyme cloning, including BsaI/BsmBI overhang design, color-based screening strategies, and thermocycler protocols.
2. Gibson Assembly: Gibson et al., 2009 (Nature Methods) [28], explaining the exonuclease–polymerase–ligase assembly mechanism and overlap design rules used for primer construction.
3. CRISPR Vector Design: Addgene's CRISPR cloning guides [29], covering sgRNA scaffold structure, spacer cloning strategies, and BsmBI-based guide insertion into gRNA vectors.

Furthermore, we used the materials included in the LitQA2 subsection of LAB-Bench. There are 103 excluded articles due to lack of public access, meaning that we successfully retrieved the full text of 45.8% of papers used. For the articles that we included, we included the full text in a JSON file.

Additional Information about Supervised Fine-Tuning Table 5 reports the average number of curriculum passages retrieved per question (k) across subtasks and conditions.

Subtask	Base (default)	DC-RS	VMG
LitQA2	0.40	0.40	0.50
DbQA	1.80	2.47	2.47
ProtocolQA	2.67	2.87	3.00
SeqQA	2.87	2.87	3.00
CloningScenarios	0.60	0.60	0.60

Table 5: Average retrieved memory size (k) across LAB-Bench subtasks under different retrieval strategies.

We see that retrieval size is largely stable across strategies here. DC-RS and VMG retrieve slightly more passages than the base condition on DbQA (1.80 vs. 2.47) and ProtocolQA (2.67 vs. 2.87–3.00), suggesting that finetuned models draw on more context when the cheatsheet synthesis step identifies relevant prior examples. LitQA2 shows no change in average k across conditions (0.40), consistent with the finding that top-k retrieval frequently returns off-domain passages that the teacher learns to

ignore. CloningScenarios similarly shows no variation (0.60), reflecting both the small evaluation set and the model-level sequence continuation failures that prevent meaningful retrieval use.

Memory Poisoning Experiment 1: Write Gates Benefit on Well Known Facts
In an early iteration of memory poisoning work, we conducted an experiment where an adversary injects five fake entries (out of 30 - i.e., a particular biomolecule has no effect as opposed to a deleterious effect, etc.) into the DC memory, where the entries were enforced by the model to use the same format as the ground truth memory entries in addition to a fair balance of words that would have high embedding similarity upon retrieval. We then implement a mechanism in the write gate in that before the model uses any retrieved memory, it is asked: "Is this memory entry state an answer that is clearly factually wrong?" Answer yes (block it) or no (keep it). In doing so, we particularly noticed that confidence in rejecting memories is inversely proportional to how niche a biological fact in memory is; for example, it is quite easy for a model to reject a well known biological fact, but on the other hand, rejecting a niche research claim makes the model uncertain and in such cases, the model generally didn't reject retrieving the memory. This could be a limitation with model choice, and we expect perhaps stronger write gate abilities in stronger reasoning models.

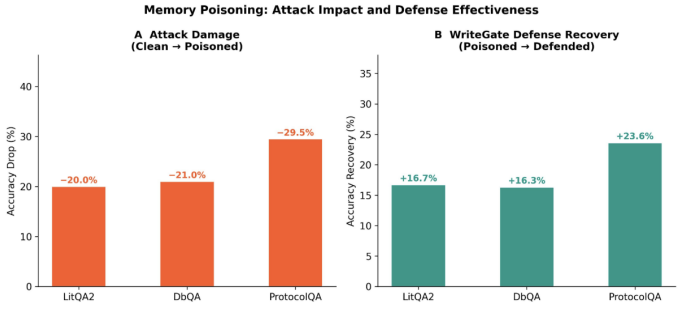

Figure 3: Model accuracy drop on factual corruption

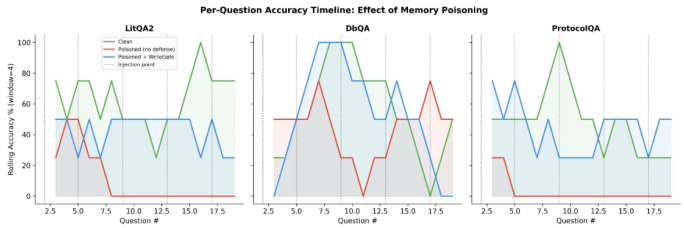

Figure 4: Model accuracy on memory poisoned entries over entries

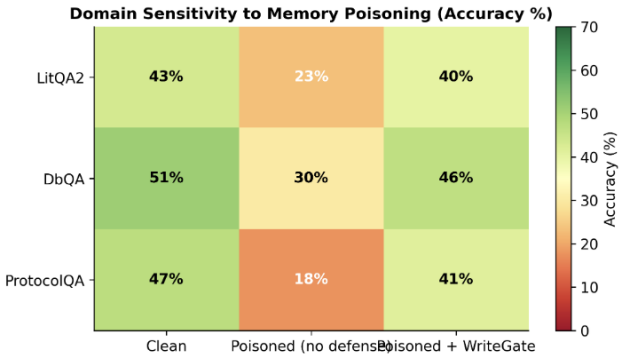

Figure 5: Domain sensitivity to memory poisoning

Memory Poisoning Empirical Experiment 1 Specifications

Component	**Details**
Memory System (DC inference)	Sequential bag-of-words **DCMemory**, not embedding-based.
Retrieval Method	Cosine similarity on word bags: $\frac{\lvert words_A \cap words_B \rvert}{\lvert words_A \cup words_B \rvert}$
Top-k Retrieval	$k = 3$: Top-3 most similar past Q→A pairs retrieved per question.
Memory Update	After each question, the correct answer is added to memory.
Poisoning Attack (P1 — Targeted Answer Flip)	Adversary injects 5 fake Q→A entries into DC memory.
Entry Format	Fake entries match legitimate entries exactly, making them indistinguishable from real memory.
Attack Strategy	Entries reuse exact question keywords from the evaluation set, producing high bag-of-words similarity during retrieval.
Attack Type	All 5 entries flip the correct answer letter to an incorrect choice.

Table 6: Memory System and Targeted Poisoning Attack Setup

Component	Details
Memory System (DC inference)	Sequential **DCMemory**: accumulates correct Q→A pairs at inference time.
Retrieval Method	Bag-of-words cosine similarity: $\frac{\lvert\text{words}_A \cap \text{words}_B\rvert}{\lvert\text{words}_A \cup \text{words}_B\rvert}$
Top-k Retrieval	$k = 3$: top-3 most similar past Q→A pairs per question.
Memory Update	After each question, the correct answer is appended to memory.
Attack Types	
P1 — Targeted Answer Flip	5 adversarial entries injected at fixed indices $\mathcal{I} = \{3, 7, 11, 15, 19\}$. Entries reuse exact question keywords (BoW cosine ≈ 0.85) but state the wrong answer letter. Format-indistinguishable from legitimate memory.
P2 — Cross-Domain Injection	DbQA entries (gene–disease vocabulary) injected into LitQA2 memory. Sentence-transformer cosine to LitQA2 questions ≈ 0.05–0.12. Pollutes context with semantically irrelevant content.
Write Gate Defenses	
WG-Sim	Filters retrieved entries by sentence-transformer cosine $\geq \tau = 0.20$. Local, zero API calls. Blocks P2 (cosine $< \tau$); *cannot* block P1 (cosine ≈ 0.85).
WG-Teacher	LLM verifies each retrieved entry: rejects if off-domain or factually wrong. One API call per entry. Blocks both P1 (when error is model-recognizable) and P2 (off-domain rejection). May fail on domain-specific P1 facts.
WG-Combined (proposed)	WG-Sim first (fast semantic filter), then WG-Teacher on passing entries. Full-spectrum defense against both attack types at minimum API cost.

Table 7: Memory Poisoning Experiment 2 Specifications: attack types and write gate defenses.

A.1 System Prompts

VMG Verifier System Prompt

```
You are a verification assistant.
Your job is to check whether the model's reasoning is grounded
in the retrieved passages.

Question:
{question_text}

Answer choices:
{choices_str}

Retrieved passages (memory entries):
{passages_str}

Candidate reasoning and answer from the model:
Reasoning:
{reasoning}
Final answer: {final_answer}

Instructions:
1. Break the reasoning into a list of short, atomic factual claims.
2. For each claim, determine whether it is directly supported by
   at least one retrieved passage.
3. If supported, copy the minimal supporting span verbatim and
   record the passage rank.
4. If not supported, set supporting_span = null and
   source_entry_rank = null, and grounded = false.
5. Decide passes_gate based on whether the reasoning is
   sufficiently grounded in the passages.
```

```
Output your analysis as a single JSON object in EXACTLY this
format (no markdown fences, no extra text):
{
  "passes_gate": true,
  "claims": [
    {
      "claim": "<short atomic factual claim>",
      "supporting_span": "<verbatim quote or null>",
      "source_entry_rank": 1,
      "grounded": true
    }
  ],
  "verifier_notes": "<free-form comments>"
}
```

DC-RS Generator System Prompt

```
# GENERATOR (PROBLEM SOLVER)
Instruction: You are an expert problem-solving assistant tasked
with analyzing and solving various questions using a combination
of your expertise and provided reference materials. Each task
will include:
1. A specific question or problem to solve
2. A cheatsheet containing relevant strategies, patterns, and
   examples from similar problems

---
## 1. ANALYSIS & STRATEGY
- Carefully analyze both the question and cheatsheet before
  starting
- Search for and identify any applicable patterns, strategies,
  or examples within the cheatsheet
- Create a structured approach to solving the problem at hand
- Review and document any limitations in the provided reference
  materials

## 2. SOLUTION DEVELOPMENT
- State your answer first, then explain your reasoning
- Use clear, logical steps that others can follow and review
- Provide concise methodology -- do not copy the cheatsheet
  verbatim
- Check and verify assumptions; focus on answering the question

## 3. PROGRAMMING TASKS
When coding is required:
- Write clean, efficient Python code
- Use a Python code block for all code
- After the code block, append: EXECUTE CODE!
\end{lstlisting}
\textcolor{red}{FOR LATER: Full
system prompts, memory templates, and representative SFT examples for
each strategy are provided in the Appendix.}
```

SFT Training
System Prompt:

```
You are a biology research assistant. Given a question, answer choices
(when provided), and retrieved memory entries, use only relevant entries to
answer. Output your final answer in <ANSWER>...</ANSWER>. Use the letter
```

```
(A/B/C/D) or exact answer text as appropriate.
```

Answer Inference Prompt Structure:

```
{question}

[TASK CURRICULUM]
{curriculum_text}

A. ...
B. ...

Answer:
```