

TRACE: Tool-augmented Reasoning via Atomic Cheatsheet Editing

Stanford CS224N Custom Project

Mentor: Mirac Suzgun

Key Information: No External Collaborators, No Sharing Project Between Classes

Late Days: Kyleen Liao has 4 late days remaining, Roshen Nair has 4 late days remaining, and Arnold Yang has 3 late days remaining. In total, the team has 11 late days available. For this report, we are applying 3 late days as a group (with a team size of 3), extending the deadline to Sunday, March 15, 2026 at 4:30 PM.

Kyleen Liao

Stanford University

kliao07@stanford.edu

Roshen Nair

Stanford University

roshen03@stanford.edu

Arnold Yang

Stanford University

arnoldya@stanford.edu

Abstract

Test-time learning promises language models that improve with experience, yet existing systems either discard prior context, update memory too slowly, or maintain memory as free-form text that is repeatedly rewritten. In approaches such as Dynamic Cheatsheet (DC), this makes it difficult to track individual strategies or distinguish those that help from those that harm. We introduce Tool-augmented Reasoning via Atomic Cheatsheet Editing (**TRACE**), a structured, tool-augmented memory system that incrementally edits a persistent store of reasoning strategies. Memory is represented as a metadata-annotated bullet store whose entries are modified through explicit operations (add, update, delete, consolidate). An oracle reflector evaluates cited entries using ground-truth feedback and tags them as helpful or harmful, while a curator performs targeted edits based on these signals. A citation judge filters generator citations to ensure that feedback is only applied to strategies genuinely used in reasoning, and a deduplication pass merges semantically similar bullets to control redundancy. Together, these mechanisms allow memory to evolve through incremental, entry-level updates rather than full rewrites. TRACE achieves **86.8%** accuracy on the Multistep Arithmetic BigBench Hard task (vs. **10.0%** without memory and **38.8%** for the strongest DC baseline) and **46.4%** on the Tracking 7 Objects task (vs. **8.4%** and **15.7%**, respectively). Across four BigBench Hard tasks and five evaluation-order seeds, TRACE reaches **65.7%** macro accuracy, outperforming all baselines (no-memory, DC-Cu, and DC-RS) by more than **20 percentage points** while exhibiting substantially lower variance.

1 Introduction

Language models (LMs) are increasingly able to solve complex multi-step reasoning tests. However, there is still a gap between the maximum potential of a language model's performance on certain tasks and its normal performance. When solving such tasks, models typically solve each instance independently. As a result, they repeatedly rediscover similar reasoning patterns and pitfalls instead of building an understanding of the task.

Test-time learning attempts to address this limitation, for example, by maintaining an external memory that evolves during evaluation. Instead of updating model weights, the system records useful strategies discovered while solving earlier problems and provides them as context for future ones. This approach has several practical advantages: it works with black-box models, avoids expensive retraining, and allows adaptation to new tasks entirely during inference. However, designing such memory systems is challenging. Memory must remain compact enough to fit in the prompt (context limits), accurate enough to avoid reinforcing incorrect strategies, and stable enough to evolve reliably across long sequences of problems.

Recent work has shown that maintaining a persistent memory of distilled strategies can substantially improve reasoning performance. In particular, Dynamic Cheatsheet (DC) [1] introduces a framework in which a language model maintains a running cheatsheet of strategies that evolves across problems. However, this cheatsheet is one block of text fully rewritten every run, while making the cheatsheet more cohesive, losing granularity and the ability to pinpoint unreliable strategies. Consequently, Agent Context Engineering (ACE) [2] was built on DC by treating cheatsheet strategies as individual modules, rather than curating a singular cheatsheet overall. Thus, instead of rewriting the entire memory after each problem, it represents it as a collection of discrete strategy bulletpoints that can be modified incrementally through the addition of a reflector module. The reflector separately analyzes the generator’s trajectories and feedback to identify useful strategies, mistakes, and failure patterns that the curator then converts into memory modules.

However, ACE only provided the curator with an add bullet function without the ability to remove bullets, and added a lot of context overhead that only fit larger models. The incremental updates also made it such that ACE accumulated its learnings more slowly than DC, thus only suitable for larger (500+ tasks) datasets. Both also noticeably struggled with Small Language Models (SLMs), where better strategies were rarer to come by, and bad or overly specific strategies dominated the cheatsheet.

Thus, we attempt to address these issues by taking the modular memory further by specializing its ability on smaller models. We propose **TRACE** (Tool-augmented Reasoning via Atomic Cheatsheet Editing), a memory curation model that updates memory using a larger set of tool calls (add, update, delete, and consolidate), each targeting individual entries as seen in Figure 1. To guide these edits further, we improve upon ACE’s oracle reflector to make it better analyze reasoning traces using the ground-truth correctness signal and label referenced memory entries as helpful or harmful. These annotations provide a direct signal for deciding which strategies should be preserved, refined, or removed.

We evaluate TRACE on four more complicated BigBench Hard reasoning tasks covering logical reasoning, spatial reasoning, procedural computation, and state tracking. Across tasks, our structured memory system achieves a macro-average accuracy of **65.7%**, compared to **43.5%** for the strongest free-text (Dynamic Cheatsheet) memory baseline and **29.3%** without memory. The largest improvements occur on tasks where reusable procedures can be expressed as discrete strategies and bullet points, including multistep arithmetic and object tracking. In addition to higher accuracy, TRACE shows substantially lower variance than free-text approaches, suggesting that structured memory updates lead to more stable test-time learning. These improvements come with increased token usage and modest runtime overhead due to reflection and tool calls, which we analyze in Appendix A.

Finally, we explore how structured memory editing can be learned rather than hard-coded. We introduce a supervised fine-tuning pipeline that trains a student model on teacher-generated generator, reflector, and curator traces. This allows the system to learn structured memory editing behavior without requiring ground-truth signals during inference.

2 Related Work

To improve performance on complex tasks, a growing line of work augments language models with stronger test-time capabilities, especially better reasoning and better use of context or memory. Early approaches include Chain-of-Thought prompting, which encourages models to decompose problems into intermediate reasoning steps [3], and few-shot in-context learning, which conditions behavior on demonstrations placed directly in the prompt [4]. Retrieval-augmented generation (RAG) extends this idea by retrieving relevant information from an external corpus during inference rather than relying only on knowledge stored in the model’s parameters [5]. More recent work, such as GEPA, explores prompt optimization and reflective prompting to improve reasoning without modifying model weights [6]. While effective, these approaches typically operate within a single inference or retrieve from static knowledge bases, leaving the open question of how models should accumulate and refine task-specific knowledge across sequences of problems.

Dynamic Cheatsheet (DC) addresses this by introducing test-time learning through a persistent cheatsheet of strategies that evolves across problems [1]. A generator solves each problem using the current cheatsheet, and a curator rewrites the cheatsheet to incorporate new strategies. The DC-CU variant performs cumulative rewriting, while DC-RS retrieves similar past traces and synthesizes a task-specific cheatsheet before generation. These results show that small stores of distilled strategies can substantially improve reasoning without parameter updates. However, because memory is

repeatedly rewritten as free-form text, it is difficult to track the history of individual strategies, and memory quality can degrade as useful and harmful strategies become entangled over long sequences.

Agentic Context Engineering (ACE) extends this idea with a generator-reflector-curator pipeline that separates self-reflection from incremental delta memory updates [2]. The reflector identifies helpful or harmful strategies from reasoning traces, and the curator updates memory accordingly. This improves efficiency and highlights challenges such as context collapse, strategy overgeneralization, and maintaining useful context over long sequences. However, ACE is specifically optimized for LLMs running on a large (500+) series of sequential problems. Our work follows the same intuition that memory should evolve incrementally, and we replace free-form rewriting with structured edit operations (add, update, delete, consolidate) that let localized updates and preserve an auditable history of changes. TRACE increases the tooling available for editing, allowing smaller models to learn faster on a smaller (30-100) set of sequential problems.

Finally, our tool-mediated memory editing connects to a broader trend of enabling language models to interact with external systems through structured interfaces. Prior work shows that models can learn to call tools for reasoning or retrieval during inference [7]. A strong parallel is incorporating human problem-solving strategies through built tool-based episodic memory, where "generative simulacra" use simplified human cognitive processes (planning, memory, and reflection modules) [8]. In TRACE, the tools operate directly on the memory store, allowing the model to treat memory as an editable data structure. We will then refine a specific SLM to specialize in memory curation tool-calling, which relies on supervised fine-tuning and distillation from stronger models [9], to create better memory edits during test time.

Overall, TRACE builds on Dynamic Cheatsheet and ACE by replacing free-form memory rewrites with structured, tool-mediated memory edits, allowing for more stable memory evolution of task-specific knowledge during test-time learning.

3 Approach

Let x_i denote the i -th problem and y_i the ground-truth answer. TRACE maintains a structured bulleted memory system M_i that evolves across the problem sequence through six sequential stages. First, a retriever selects a relevant subset $B_i \subseteq M_i$. Second, the generator produces an answer $\tilde{y}_i$ and a set of cited bullet IDs C_i . Then, an optional citation judge filters C_i to the subset $C'_i \subseteq C_i$ of bullets that it decides have been genuinely and appropriately used in reasoning. Fourth, the reflector updates the helpful and harmful metadata of bullets in C'_i in place, auto-deactivates any bullet whose harmful count reaches a threshold, and outputs textual feedback F_i for the curator (we write M'_i for the store after these in-place updates). Next, the curator applies structural tool-based edits guided by both M'_i and F_i to produce an intermediate store M''_i . Finally, a deduplication pass produces the final updated store M_{i+1} :

$$B_i = \text{Retr}(x_i, M_i), \quad (1)$$

$$(\tilde{y}_i, C_i) = \text{Gen}(x_i, B_i), \quad (2)$$

$$C'_i = \text{CitationJudge}(x_i, \tilde{y}_i, C_i, M_i), \quad (3)$$

$$(M'_i, F_i) = \text{Refl}(M_i, x_i, \tilde{y}_i, C'_i, y_i), \quad (4)$$

$$M''_i = \text{Cur}(M'_i, F_i), \quad (5)$$

$$M_{i+1} = \text{Dedup}(M''_i). \quad (6)$$

Equation (3) corresponds to an optional component: when the citation judge is disabled, $C'_i = C_i$. We evaluate the impact of the citation judge in the ablation studies reported in Appendix B. It is important to emphasize that the reflector is an oracle component (Equation (4) receives y_i), which we discuss further below. The key architectural departure from DC is the separation of feedback-signal reflection and the structural, tool-based content editing, which DC conflates in a single free-text rewrite. A high-level diagram of this is illustrated in Figure 1.

Memory format. Memory is stored as a list of bullet points, each with an immutable five-digit identifier and structured metadata:

```
[ctx-00001] section=Solutions usage=2 helpful=3 harmful=1 active=1 created=0
updated=3 :: content
```

Each field serves a specific role in the pipeline. `usage` is incremented each time the generator includes the bullet in C_i , providing a signal of how frequently a strategy has been consulted. `helpful` and `harmful` are incremented by the reflector when a cited bullet is judged to have contributed positively or negatively to the answer. `active` is set to 0 when the curator deactivates a bullet, and then it is excluded from retrieval and display but retained in history for auditing. `created` and `updated` record the problem indices at which the bullet was first added and last revised, enabling the curator to identify stale entries. `section` is a categorical label (e.g., "Solutions", "Strategies") assigned by the curator to organize the store. Individual bullet content is capped at 600 characters and passed through a sanitization step that detects and removes repeated phrases before storage.

Bullet Retriever. Before each generation step, the retriever selects the bullets most relevant to the current problem from the active store M_i . Given the problem text x_i , the retriever encodes it and each active bullet’s content using a pretrained sentence-transformer model and computes the cosine similarity between the query embedding and each bullet embedding. Each similarity score is then scaled by a quality weight derived from the bullet’s `helpful` and `harmful` counts, so that strategies with a consistent track record of usefulness rank above untested or harmful ones. The retriever returns the top- k bullets by this weighted score and always includes any bullet whose `helpful` count meets or exceeds a minimum threshold, regardless of its raw similarity to the current query. The generator then receives only this retrieved subset B_i rather than the full memory store, reducing the risk of low-quality or irrelevant strategies occupying limited context.

Generator. The generator receives the current problem x_i and the retrieved cheatsheet B_i formatted as a structured bullet list. It must produce a response by calling the `submit_answer` tool with three fields: `reasoning` (a step-by-step reasoning trace), `bullet_ids` (a list of IDs for any bullets consulted in producing the answer), and `final` (the extracted answer). We set `tool_choice="required"` to enforce structured output at the provider level, but allow the system to fall back to text extraction when the provider does not support this setting. The `bullet_ids` field is how the generator’s reliance on specific memory entries becomes explicit and feeds into downstream feedback (from the reflector).

Reflector. The reflector receives the problem x_i , the generator’s reasoning trace and answer $\tilde{y}_i$, a binary ground-truth correctness signal (whether $\tilde{y}_i = y_i$), the cited bullet IDs C_i , and the active cheatsheet M_i . It operates through two sequential tool calls. First, it calls `update_bullet_feedback(bullet_id, tag, reason)` for each bullet it judges to be relevant, where `tag` $\in$ {`helpful`, `harmful`, `neutral`} and `reason` is a brief textual justification. Each bullet may be tagged at most once per reflection step to prevent double-counting. Second, it calls `submit_reflection_feedback(summary, cheatsheet_vision)`, where `summary` contains an overall assessment of the current memory system and `cheatsheet_vision` provides restructuring guidance passed directly to the curator. Together, these two calls implement Equation (3) as the metadata fields `helpful` and `harmful` are updated to produce M'_i , while the cheatsheet vision provides helpful notes for the curator’s editing decisions in Equation (4).

The reflector is an oracle component: it receives the true correctness label y_i at each step. This is appropriate for analyzing what a structured memory system can achieve when feedback is reliable, but it requires an external correctness signal that is not always available in all tasks.

Citation Judge. An optional citation judge acts as a filter before the reflector applies any tag from `update_bullet_feedback`. Given the problem x_i , the generator’s full reasoning trace, and the content of a specific bullet, the judge is prompted to determine whether the bullet was meaningfully used in reaching the answer, as opposed to simply appearing in the context. If the judge concludes it was not meaningfully used, the `update_bullet_feedback` call is rejected, and the bullet’s `helpful` and `harmful` counts are left unchanged. This prevents strategies from accumulating fake feedback through citation gaming, where the generator lists a bullet in C_i without genuinely relying on it (so essentially it aims to circumvent fake citations from the generator). In later sections, we present detailed ablations and analyses when running our experiments with and without this citation judge.

Curator. The curator receives the reflection summary and cheatsheet vision from `submit_reflection_feedback`, along with the current memory state M'_i (which now re-

flects updated helpful/harmful metadata from the reflector). It stages memory edits through five tool calls before finalizing:

- `add_bullet(section, content)`: creates a new bullet initialized with `usage=0`, `helpful=0`, `harmful=0`,
- `update_bullet(bullet_id, content)`: revises an existing bullet’s content and advances its updated index,
- `delete_bullet(bullet_id)`: sets `active=0`, deactivating the bullet without erasing its history (remains its log files for manual auditing),
- `consolidate_bullets(bullet_ids, content)`: merges two or more bullets into a single, more general entry. Note that the new bullet inherits the sum of source usage counts and the maximum of their helpful and harmful counts;
- `submit_curator_edits()`: commits all staged edits, producing M_{i+1} .

The curator is instructed to maintain 10 to 20 active bullets, to prioritize consolidation and deletion before adding new entries (early experiments showed that an unconstrained store grew large enough to degrade performance on later problems), and to remove bullets with `usage=0` that were created three or more problems prior.

Deduplication. Inspired by ACE [2], an optional deduplication pass runs after curation. All active bullets in M_{i+1} are encoded with a sentence-transformer model, and FAISS is used to efficiently identify all pairs whose cosine similarity exceeds 0.90. Each cluster of near-duplicate bullets is merged via an additional LLM call that produces a single consolidated entry, with metadata inherited from its sources. This step runs independently of the main curator loop and serves as a safety net against semantic redundancy that the curator did not catch in early experiments.

Baselines. We compare against three baselines. **(1) No-memory**: the model solves each problem independently with no cheatsheet or retrieval. **(2) DC-Cumulative (DC-Cu)** [1]: after each problem, a curator rewrites the cheatsheet as free-form text conditioned on the previous cheatsheet and the generator’s response, following the update $M_{i+1} = \text{Cur}(M_i, x_i, \tilde{y}_i)$. **(3) DC with Retrieval and Synthesis (DC-RS)** [1]: the most similar past input-output traces are retrieved using pre-computed `all-mpnet-base-v2` embeddings, an updated cheatsheet is synthesized conditioned on those traces and the current problem, and the answer is generated from that cheatsheet, following the DC-RS update equations. DC baselines use exact prompt templates and code setup from the official DC repository. The no-memory baseline is run with a minimal generator prompt that presents only the current problem and instructs the model to respond with the final answer only (no cheatsheet, reflector, retrieval, etc.).

Implementation. Our implementation builds on the Dynamic Cheatsheet codebase [1]. We wrote `generator.py`, `reflector.py`, `curator.py`, `retriever.py`, and `cheatsheet.py` as new modules and modified `language_model.py` (and prompts) to support the full tool-calling pipeline. This included building all the new tool-call functions the LLM could call, refining the system for SLMs, and implementing the deduplicator and citation judge. The `BulletpointAnalyzer` deduplication module is adapted from the ACE repository [2]. We also set up supervised fine-tuning in `our_approach/sft`, including a dedicated subfolder for collecting teacher-model traces, which was the main part of the SFT pipeline we experimented with and discuss in the analysis section.

Supervised Fine-tuning. The weakest and most error-prone point in the cheatsheet process is still the curator for SLMs. Thus, we proposed to finetune the curator function for Qwen3-8B with traces from Claude 4.6 Sonnet as the teacher model. As seen in Figure 1, we would have Qwen3-8B perform all other parts of the pipeline except the curator step. For each dataset, we marked off the first 50 problems as evaluation and collected traces and full logs for the remaining 200 problems for evaluation. Each trace would be formatted such that the rest of the step would be identical to the trace logs, and then Qwen3-8B would substitute Claude 4.6 Sonnet at the Curator step. Then we apply cross-entropy loss only on the curator’s assistant tokens, masking all system, user, and tool-result turns using TRL’s `DataCollatorForCompletionOnlyLM` [10].

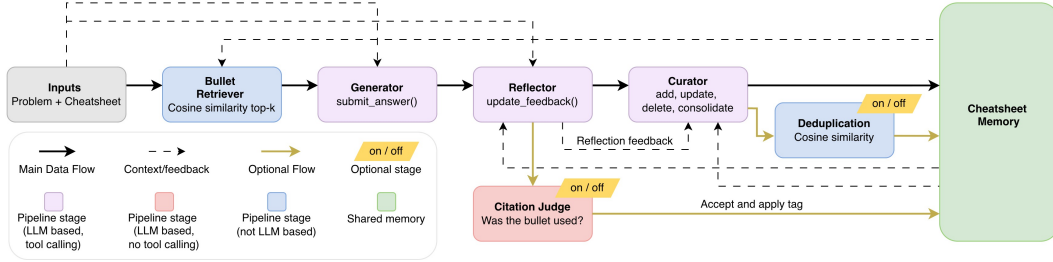


Figure 1: Flow of TRACE for one problem.

4 Experiments

4.1 Data

We evaluate on four reasoning tasks from the BigBench Hard (BBH) benchmark [11]: **Formal Fallacies** (classifying whether a logical argument is formally valid), **Geometric Shapes** (identifying a geometric shape from an SVG path description), **Multistep Arithmetic** (evaluating nested multi-level arithmetic expressions), and **Tracking Shuffled Objects** (tracking the final positions of seven objects through a sequence of pairwise swaps). These tasks span logical, spatial, procedural, and state-tracking reasoning and were selected because Qwen3-8B has a meaningful opportunity for progress on each. The questions are divided into problem-solution pairs; thus, we input the problem as part of our input prompt and a loose answer extractor to extract the answer output, which could be in the form of an integer, valid/invalid, or a multiple-choice answer.

For each task, we sample 250 examples from the official BBH dataset using a fixed random seed. The first 50 examples (indices 0-49) form the held-out test set, while the remaining 200 (indices 50-249) are reserved exclusively for SFT training, with no overlap between training and evaluation. To measure stability under different problem orderings (an important concern for memory-based methods since the order problems are solved determines which strategies are available when), we evaluate each configuration across five evaluation-order seeds, yielding 250 total evaluation episodes per task per approach.

4.2 Evaluation Method

We report exact-match accuracy as the primary metric, following standard BBH evaluation practice. Task-specific evaluation handles format differences: Formal Fallacies and Geometric Shapes use multiple-choice letter matching, while Tracking Shuffled Objects uses string matching over the target object’s final position, and finally Multistep Arithmetic compares parsed integers. The answer extraction pipeline checks the generator’s structured final tool-call field first, falls back to JSON parsing, then searches for an explicit "Final answer:" phrase, and finally looks for a trailing standalone integer in the model’s output. All results are reported as mean accuracy $\pm$ standard deviation across the five evaluation-order random seeds (for experimental reproducibility), quantifying how reliably each approach performs across different problem orderings. In the appendices, we also present result tables tracking other metrics such as token usage and runtime.

4.3 Experimental Details

All approaches use Qwen3-8B as the base language model, served via vLLM on a single NVIDIA A100 80GB GPU using Modal cloud infrastructure. We enable vLLM’s native tool-call support with `-enable-auto-tool-choice` and `-tool-call-parser qwen3_coder`, and set a maximum context length of 32,768 tokens. To mitigate output degeneration in the multi-turn tool-loop settings used by the reflector and curator, we apply a frequency penalty of 0.35 and a presence penalty of 0.20 during these calls. DC-Cu and DC-RS use the exact prompt templates and extraction logic from the official DC repository. DC-RS additionally requires pre-computing `all-mpnet-base-v2` sentence embeddings for all task examples before evaluation - these are computed once per task and reused across all seeds (consistent with the paper’s repo). In total, for the main experiments (not counting ablations), we run 80 evaluation configurations (4 tasks $\times$ 4 approaches $\times$ 5 random seeds), with each configuration processing 50 problems in a fixed order.

Table 1: **Performance on 4 BBH tasks** - This table displays the mean accuracy (%) with the accuracy-range from 5 different seeds on 4 different BBH tasks with different memory frameworks for Qwen3-8B. TRACE achieves **65.7%** macro-average accuracy, outperforming the strongest baseline by over 22 percentage points, with consistently lower variance than Dynamic Cheatsheet baselines.

Task	Baselines			TRACE
	No-Memory	DC-RS	DC-Cu	
Formal Fallacies	80.0 $\pm$ 1.3	75.5 $\pm$ 2.4	83.7 $\pm$ 3.5	83.6 $\pm$ 2.0
Geometric Shapes	18.8 $\pm$ 1.0	50.1 $\pm$ 7.5	54.9 $\pm$ 16.1	46.0 $\pm$ 5.1
Multistep Arithmetic	10.0 $\pm$ 2.3	38.8 $\pm$ 7.7	18.4 $\pm$ 6.6	86.8 $\pm$ 3.4
Tracking (7 obj.)	8.4 $\pm$ 1.5	9.7 $\pm$ 13.7	15.7 $\pm$ 8.9	46.4 $\pm$ 4.6
Macro-Avg	29.3	43.5	43.2	65.7

5 Results & Analysis

Table 1 compares TRACE against the no-memory baseline and two Dynamic Cheatsheet variants. TRACE achieves **65.7% macro-average accuracy**, improving over the strongest baseline (DC-RS) by **22.2 percentage points**, with consistently lower variance across evaluation-order seeds. The pattern of results splits cleanly by task type: tasks with reusable procedural structure benefit substantially, while tasks requiring perceptual recognition or where the base model is already strong show smaller gains.

A key qualitative difference is how strategies persist over time. Free-text cheatsheets rewrite the entire memory after each problem, causing strategies to drift or be accidentally overwritten. TRACE instead applies localized edits to persistent bullet entries, allowing useful procedures to remain stable while harmful ones are explicitly removed.

5.1 Procedural Reasoning: Multistep Arithmetic and Tracking Shuffled Objects

The largest gains appear on tasks with transferable procedural structure. On **Multistep Arithmetic**, TRACE achieves **86.8%** versus **38.8%** for DC-RS and **18.4%** for DC-Cu. This task reduces to a single reusable rule (evaluate innermost parentheses first), which, once captured by the curator, transfers to nearly every subsequent problem. DC-Cu degrades this strategy through repeated full rewrites; DC-RS recovers it only when similar traces happen to appear early in the sequence.

Tracking Shuffled Objects follows the same pattern: **46.4%** versus **15.7%** and **9.7%**, respectively. Effective strategies (e.g., maintain an explicit position table, update after each swap) generalize across all problems once learned. DC-RS exhibits extreme variance (± 13.7) because retrieval quality depends heavily on problem-phrasing similarity, while TRACE converges to stable strategies with much lower variance (± 4.6). Concrete examples of the memory evolution process on this task are shown in Appendix C.

5.2 Limited-Headroom and Perceptual Tasks

On **Formal Fallacies**, the no-memory baseline already reaches **80.0%**, leaving little room for memory-based improvement. TRACE (83.6%) and DC-Cu (83.7%) are essentially tied, confirming that memory systems are most useful when tasks require discovering new procedures rather than applying pretrained knowledge.

Geometric Shapes is the one task where DC-Cu outperforms us (54.9% vs. 46.0%), though with dramatically higher variance (± 16.1 vs. ± 5.1). We attribute this to the perceptual nature of the task: recognizing shapes from SVG path commands depends on subtle spatial heuristics that are difficult to encode as concise reusable bullets. Free-text cheatsheets can capture these loosely structured patterns more naturally, but at the cost of instability across orderings.

5.3 Ablations

We analyzed the citation faithfulness judge, deduplication pass, and memory deletion thresholds. These modifications produced only modest and inconsistent performance differences, suggesting that the primary gains arise from the structured memory architecture itself rather than any single auxiliary component. Full results are in Appendix B.

5.4 Supervised Fine-tuning

Due to Claude API errors, Claude Sonnet 4.6 did not consistently outperform Qwen3-8B on the BigBench Hard tasks during teacher trace collection. Combined with limited remaining API credits, this prevented us from completing the full supervised fine-tuning (SFT) pipeline. Investigating stronger or more reliable teacher models for trace generation remains an important direction for future work.

6 Conclusion

We presented TRACE, a novel, structured, tool-augmented memory system for test-time learning in language models. Rather than rewriting memory as free-form text after each problem, TRACE maintains a metadata-annotated bullet store that evolves through explicit operations (add, update, delete, consolidate). An oracle reflector labels cited entries as helpful or harmful using ground-truth correctness signals, and a curator performs targeted edits guided by these annotations. Together, these components allow the memory store to accumulate, refine, and self-correct over a sequence of problems.

Structured memory editing yields substantial and stable improvements on tasks with reusable procedural structure. TRACE achieves 86.8% on Multistep Arithmetic (a 48-point gain over the strongest Dynamic Cheatsheet baseline) and 46.4% on Tracking Shuffled Objects, where baselines exhibit high variance. Gains are modest where pretrained model capabilities are already strong (Formal Fallacies) or strategies resist procedural encoding (Geometric Shapes). Crucially, TRACE exhibits substantially lower variance across evaluation-order seeds than all baselines, likely a consequence of localized updates, explicit harmful-entry deactivation, and metadata-guided prioritization. These properties make structured memory editing a more stable and auditable foundation for test-time learning than free-text approaches.

Limitations. Several limitations constrain the current approach. The reflector is an oracle component, meaning it receives the ground-truth correctness label at each step, which limits its use to settings where such signals are available. We did not have the chance to meaningfully test the viability of not having a ground truth (but the results from the Agentic Context Engineering paper [2] suggest it causes poor performance with small language models). Our evaluation is also restricted to four BBH tasks using a single model (Qwen3-8B), and it remains to be seen how the system generalizes to open-ended or non-classificatory tasks as well as other small language models. The structured bullet format, while effective for procedural reasoning, may be too rigid to capture perceptual heuristics, as suggested by the Geometric Shapes results. Finally, the pipeline introduces additional token usage and latency relative to no-memory and free-text baselines, which we characterize in detail in Appendix B.

Future work. The most direct extension is removing the oracle dependency by replacing the ground-truth correctness signal with a model-based judge or a self-consistency signal. Our analysis suggests that the curator was still the weakest point in the system, and fine-tuning a stronger curator would be the goal of a better system. A further direction is extending the memory format to support richer structures, such as conditional rules or hierarchical strategy trees, that may better accommodate tasks where flat bullet lists are insufficient. Finally, scaling the evaluation to longer problem sequences and a broader set of tasks would clarify how the system’s performance ceiling depends on memory budget, task diversity, and the quality of early-sequence strategies.

Team contributions

All three team members contributed equally overall to the project. Everyone contributed to building the core pipeline components (generator, curator, reflector). Kyleen additionally implemented deduplication and the citation judge. Arnold implemented the bullet retriever. Roshen integrated the baseline comparisons and built the foundation for SFT testing. All members ran experiments and/or ablations.

References

- [1] Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory, 2025.
- [2] Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models, 2026.
- [3] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.
- [4] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.
- [5] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey, 2024.
- [6] Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziem, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. Gepa: Reflective prompt evolution can outperform reinforcement learning, 2025.
- [7] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023.
- [8] 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, 2023.
- [9] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.
- [10] Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. TRL: Transformers Reinforcement Learning, 2020.
- [11] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, , and Jason Wei. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.
- [12] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Prompt repetition improves non-reasoning llms, 2025.

A Runtime and Token Use

As seen in Table 2, TRACE uses around twice the number of tokens as both free-form text baselines. This is due to the extra steps of the reflector tool call and citation judge tool call, where the whole process is re-inputted utilizing the extra tokens. We hope to optimize this further in the future, but note that the increased performance of SLMs means that these cheatsheet tasks could be handed off to cheaper specialized models.

Table 2: Average tokens consumed per question across different memory architectures.

Task	No-memory	DC-CU	DC-RS	TRACE
Formal Fallacies	200	6,368	7,678	12,704
Geometric Shapes	231	6,530	6,760	11,794
Multistep Arithmetic	127	4,933	4,800	10,173
Tracking Objects	289	6,277	6,609	13,350
Overall	212	6,027	6,462	12,005

In Table 3 below, we can see that TRACE spends similar time to other cheatsheet memory structures across the different tasks, fluctuating in terms of which tasks use the most time.

Table 3: Average time spent (seconds) per question.

Task	No-memory (s)	DC-CU (s)	DC-RS (s)	TRACE (s)
Formal Fallacies	0.1	17.8	20.5	9.8
Geometric Shapes	0.1	14.5	15.2	13.9
Multistep Arithmetic	0.3	5.8	10.5	8.2
Tracking Objects	0.1	12.3	13.8	18.6
Overall	0.2	12.6	15.0	12.6

B Ablation Studies

We chose our ablation studies according to the new features we added to the pipeline, as well as where we suspected the biggest change factors in performance may be. We note that our model had a 15 bullet point limit throughout the testing due to Qwen3-8B’s limited context and the observance of degeneration occurring for much larger cheatsheets. We also note that TRACE never utilized more than 10 bullet points at its max across various run configurations; therefore, we decided that it would show little difference to run ablation testing on our max bulletpoint counts.

According to our results from Table 4, the citation faithfulness judge did not seem to have a large enough effect on TRACE that it affected its performance in any significant way. We suspect that this was due to the maintenance of a low bulletpoint count cheatsheet.

Table 4: Citation Judge ablation (With/Without Judge)

Task	TRACE (cj1, with Judge)	TRACE (cj0, without Judge)	Δ
Formal Fallacies	83.6 ± 2.0	82.4 ± 2.3	-1.2
Geometric Shapes	46.0 ± 4.9	47.6 ± 5.4	+1.6
Multistep Arithmetic	86.8 ± 3.2	87.6 ± 6.1	+0.8
Tracking (7 obj.)	46.4 ± 4.5	43.2 ± 7.7	-3.2
Macro Avg	65.7	65.2	-0.5

The limiting factor for our pipeline’s performance on Geometric Shapes against DC-CU and DC-RS seemed to be our choice to keep the cheatsheet helpful by pruning harmful strategies as soon as they hurt the performance on a single question. Geometric shapes is a harder dataset, where the accumulation of various small domain-specific knowledge is very helpful. However, in general, the pruning was because for short datasets and runs like 50 questions on BigBench Hard it was hard

Table 5: Ablation of Bulletpoint Harmful Deletion Threshold (τ)

Task	HDT=1 (Baseline)	HDT=2	HDT=5
Formal Fallacies	83.6 $\pm$ 2.2	81.2 $\pm$ 4.8	83.2 $\pm$ 4.8
Geometric Shapes	46.0 $\pm$ 5.5	53.5 $\pm$ 4.4	50.8 $\pm$ 8.8
Multistep Arithmetic	86.8 $\pm$ 3.6	85.6 $\pm$ 1.7	87.2 $\pm$ 3.6
Tracking Shuffled	46.4 $\pm$ 5.0	45.0 $\pm$ 8.4	39.2 $\pm$ 4.4
Average	65.7	66.3	65.1
Max Bullets Observed	4	6	4

Table 6: Deduplication ablation (With/without deduplication pass)

Task	Baseline (TRACE)	Dedup=0	Δ
Formal Fallacies	83.6 $\pm$ 2.2	86.8 $\pm$ 4.5	+3.2
Geometric Shapes	46.0 $\pm$ 5.5	51.6 $\pm$ 6.4	+5.6
Multistep Arithmetic	86.8 $\pm$ 3.6	84.8 $\pm$ 5.6	-2.0
Tracking Shuffled	46.4 $\pm$ 5.0	42.0 $\pm$ 5.5	-4.4
Average	65.7	66.3	+0.6

for an SLM to accumulate good strategies, and more likely than not, a strategy that it found would hurt it more than help. Therefore, we wanted to do an ablation study to see if changing the Deletion Threshold would increase or decrease performance.

As seen in Table 5 and reviewing the logs, the slight increase in the threshold seemed to have allowed for greater strategy accumulation that actually increased performance for Geometric shapes. However, it seems that increasing it further did not have further performance gains, which seems to be the result of increasing the population of bad strategies in the cheatsheet that are removed by the curator, but decrease context for creating better strategies.

Another interesting mechanism studied was deduplication, which was recommended by ACE and would hopefully remove extra redundant context. In Table 6, the results show that there were some large increases and decreases in performance on various tasks when the deduplication module was removed. However, there was not a statistically significant enough change in performance across 5 seeds to say that this change was not due to random chance. The analysis of the task logs does show that there may be some tendency for the prompt repetition to increase good strategy use on some tasks and bad strategy use on others [12].

C Cheatsheet Examples

Below are two traces from the Tracking Shuffled Objects task, chosen to illustrate different aspects of the cheatsheet system in action.

Question #4 (Partner Swap, Square Dance) is an early-run example where the cheatsheet already holds two bullets, one for partner swaps and one for position swaps, both encoding the same procedural strategy: track sequentially, do not skip steps. The generator cited both bullets and methodically tracked Bob’s partner through all seven swaps. This is the clearest example of the cheatsheet establishing a reusable procedural strategy, rather than a simple factual hint, that the model follows step-by-step.

Question #18 (Position Swap, Soccer) is a more complex example involving seven sequential position swaps and only a single cheatsheet bullet; that bullet includes a detailed worked example from a prior question. The generator followed the exact same pattern as before: list initial positions, apply each swap, record intermediate results. Notably, the curator updated this bullet after the question, actively refining it to generalize from partner-tracking to position-tracking. This trace illustrates the system’s refinement loop: bullets improve with each question they are exercised on.

Question #4 — Partner Swap (Square Dance)

QUESTION #4

PROBLEM

Alice, Bob, Claire, Dave, Eve, Fred, and Gertrude are dancers at a square dance. At the start of a song, their partners are:

Alice	→ Sam	Bob	→ Rodrigo	Claire	→ Jamie
Dave	→ Lola	Eve	→ Helga	Fred	→ Melissa
Gertrude	→ Ophelia				

SWAP SEQUENCE

1. Bob ↔ Dave
2. Dave ↔ Claire
3. Claire ↔ Eve
4. Alice ↔ Gertrude
5. Alice ↔ Bob
6. Bob ↔ Fred
7. Fred ↔ Eve

OPTIONS

(A) Sam (B) Rodrigo (C) Jamie (D) Lola (E) Helga (F) Melissa (G) Ophelia

Target: (F) Melissa

CHEATSHEET MEMORY M_i

ACTIVE BULLETS

[ctx-00001] section=Solutions usage=1 helpful=0 harmful=0 active=1 created=0 updated=2 :: Track partner swaps sequentially to avoid errors in multi-step exchanges. Start with the initial partner of the target individual and update their partner after each swap, ensuring no assumptions are made about intermediate steps.

[ctx-00002] section=General Meta-Reasoning Strategies usage=0 helpful=0 harmful=0 active=1 created=1 updated=1 :: Track position swaps sequentially to avoid errors in multi-step exchanges. Start with the initial position of the target individual and update their position after each swap, ensuring no assumptions are made about intermediate steps.

GENERATOR submit_answer()

1. ANALYSIS & STRATEGY

Relevant cheatsheet bullets: ctx-00001, ctx-00002 — both advise tracking the target individual's partner sequentially through each swap without skipping intermediate steps.

2. SOLUTION DEVELOPMENT

Swap	Operation	Bob's partner	Note
—	Initial state	Rodrigo	
1	Bob ↔ Dave	Lola	Bob takes Dave's partner
2	Dave ↔ Claire	Lola	Bob uninvolved
3	Claire ↔ Eve	Lola	Bob uninvolved
4	Alice ↔ Gertrude	Lola	Bob uninvolved
5	Alice ↔ Bob	Ophelia	Bob takes Alice's partner
6	Bob ↔ Fred	Melissa	Bob takes Fred's partner
7	Fred ↔ Eve	Melissa	Bob uninvolved

3. FINAL ANSWER FORMAT

reasoning: "Track Bob's partner swaps sequentially. Bob starts with Rodrigo. Swap 1 (Bob/Dave): Bob gets Lola. Swaps 2-4: Bob uninvolved, stays with Lola. Swap 5 (Alice/Bob): Bob gets Ophelia. Swap 6 (Bob/Fred): Bob gets Melissa. Swap 7: Bob uninvolved. Final: Melissa."
`bulletids : ["ctx - 00001", "ctx - 00002"] final : "(F)"`

Answer: (F) ✓ **Correct**

CITATION JUDGE [optional stage]

WAS THE BULLET MEANINGFULLY USED?

Bullet	Verdict	Reason
ctx-00001	Rejected	Sequential tracking applied but not attributable to this bullet.
ctx-00002	Rejected	Same rationale; metadata unchanged.

$C_i = \{\text{ctx-00001}, \text{ctx-00002}\} \xrightarrow{\text{citation judge}} C'_i =$ (no feedback tags applied)

REFLECTOR update_feedback()

REFLECTION SUMMARY

The model correctly tracked Bob's partner swaps using a sequential strategy, leading to the correct final answer. Sequential tracking effectively avoided errors in this multi-step exchange.

BULLET FEEDBACK CALLS

Bullet	Proposed tag	Reason
ctx-00001	helpful	Provides a clear strategy for tracking partner swaps; directly used in reasoning.
ctx-00002	helpful	Reinforces the sequential tracking strategy; applied correctly.

Both tags suppressed by citation judge ($C'_i =$); helpful counts not incremented.

CHEATSHEET VISION (for Curator)

Append (Q4) to both bullets to record that the sequential-swap strategy was exercised on this problem type.

CURATOR add / update / delete / consolidate

STAGED EDITS

Tool call	Target & Edit
update_bullet	ctx-00001: Append (Q4); updated $\rightarrow$ 3, usage $\rightarrow$ 2
update_bullet	ctx-00002: Append (Q4); updated $\rightarrow$ 3, usage $\rightarrow$ 1
submit_curator_edits	Commits all staged edits

CURATOR SUMMARY

"Updated existing bullets to incorporate insights from Question #4 and ensure they reflect the correct application of sequential tracking in partner and position swaps."

CHEATSHEET MEMORY M_{i+1} (updated)

[ctx-00001] section=Solutions usage=2 helpful=0 harmful=0 active=1 created=0 updated=3 :: Track partner swaps sequentially to avoid errors in multi-step exchanges. Start with the initial partner of the target individual and update their partner after each swap, ensuring no assumptions are made about intermediate steps. (Q4)

[ctx-00002] section=General Meta-Reasoning Strategies usage=1 helpful=0 harmful=0 active=1 created=1 updated=3 :: Track position swaps sequentially to avoid errors in multi-step exchanges. Start with the initial position of the target individual and update their position after each swap, ensuring no assumptions are made about intermediate steps. (Q4)

Final answer: (F) Melissa Status: Correct Tokens: 10,409 Model: Qwen3-8B

Question #18 — Position Swap (Soccer)

QUESTION #18

PROBLEM

Alice, Bob, Claire, Dave, Eve, Fred, and Gertrude are on the same team in a soccer match. Initial positions:

Alice	→ center midfielder	Bob	→ right winger	Claire	→ right midfielder
Dave	→ benchwarmer	Eve	→ cheerleader	Fred	→ goalkeeper
Gertrude	→ left winger				

SWAP SEQUENCE

1. Fred ↔ Bob
2. Bob ↔ Claire
3. Alice ↔ Eve
4. Dave ↔ Claire
5. Fred ↔ Alice
6. Bob ↔ Fred
7. Alice ↔ Gertrude

OPTIONS

(A) center midfielder (B) right winger (C) right midfielder (D) benchwarmer (E) cheerleader (F) goalkeeper (G) left winger

Target: (C) right midfielder

CHEATSHEET MEMORY M_i

ACTIVE BULLETS

```
[ctx-00011] section=General Meta-Reasoning Strategies usage=0
helpful=0 harmful=0 active=1 created=16 updated=16 :: Track
multi-step partner swaps by updating records after each swap.
Example: Alice and Bob switch -> Alice has Izzi, Bob has Karl; Eve
and Gertrude switch -> Eve has Patrick, Gertrude has Rodrigo; Claire
and Gertrude switch -> Claire has Patrick, Gertrude has Melissa;
Gertrude and Bob switch -> Gertrude has Karl, Bob has Melissa;
Claire and Bob switch -> Claire has Karl, Bob has Melissa; Fred
and Gertrude switch -> Fred has Karl, Gertrude has Melissa; Claire
and Dave switch -> Claire has Dave, Dave has Melissa. Final partner
of Bob is (E)
```

GENERATOR submit_answer()

1. ANALYSIS & STRATEGY

Relevant cheatsheet bullet: ctx-00011 — advises tracking multi-step swaps by updating all affected records after each swap. Apply this to follow Fred's position through every trade.

2. SOLUTION DEVELOPMENT

Swap	Operation	Fred's position	Note
—	Initial state	goalkeeper	
1	Fred ↔ Bob	right winger	Fred takes Bob's position
2	Bob ↔ Claire	right winger	Fred uninvolved
3	Alice ↔ Eve	right winger	Fred uninvolved
4	Dave ↔ Claire	right winger	Fred uninvolved
5	Fred ↔ Alice	cheerleader	Fred takes Alice's position
6	Bob ↔ Fred	right midfielder	Fred takes Bob's position
7	Alice ↔ Gertrude	right midfielder	Fred uninvolved

3. FINAL ANSWER FORMAT

reasoning: "Track each swap step-by-step. Fred and Bob swap -> Fred is right winger, Bob is goalkeeper. Bob and Claire swap -> Bob is right midfielder, Claire is goalkeeper. Alice and Eve swap -> Alice is cheerleader, Eve is center midfielder. Dave and Claire swap -> Dave is goalkeeper, Claire is benchwarmer. Fred and Alice swap -> Fred is cheerleader, Alice is right winger. Bob and Fred swap -> Bob is cheerleader, Fred is right midfielder. Alice and Gertrude swap -> Alice is left winger, Gertrude is right winger. Final position of Fred is right midfielder."
bullet_{ids} : ["ctx - 00011"]final : "(C)"

Answer: (C) ✓ Correct

CITATION JUDGE [optional stage]

WAS THE BULLET MEANINGFULLY USED?

Bullet	Verdict	Reason
ctx-00011	Rejected	Sequential tracking was applied but not attributable to this specific bullet. Metadata unchanged.

$C_i = \{\text{ctx-00011}\} \xrightarrow{\text{citation judge}} C'_i =$ (no feedback tags applied)

REFLECTOR update_feedback()

REFLECTION SUMMARY

The model correctly tracked all position swaps using the provided strategy, leading to the correct final position for Fred. The reasoning trace accurately followed the full sequence of swaps.

BULLET FEEDBACK CALLS

Bullet	Proposed tag	Reason
ctx-00011	helpful	Provided a clear strategy for tracking multi-step swaps; directly applied to solve the problem correctly.

Tag suppressed by citation judge ($C'_i =$); helpful count not incremented.

CHEATSHEET VISION (for Curator)

Update ctx-00011 to generalise the example using this problem's swap sequence, replacing the partner-tracking framing with a position-tracking framing.

CURATOR add / update / delete / consolidate

STAGED EDITS

Tool call	Details
update_bullet	ctx-00011: replace partner-tracking example with position-tracking example from Q18; reframe strategy as " <i>Track multi-step position swaps by updating records after each swap.</i> "
submit_curator_edits	Commits staged edit (called 3 $\times$; duplicate calls due to model repetition — only first commit applied).

Note: curator made 6 tool calls (3 duplicate update_bullet + 3 duplicate submit_curator_edits); only the first update and first submit are effective.

CURATOR SUMMARY

"Updated the bullet to generalise the strategy for tracking position swaps with a clearer example and the correct final position for Fred."

CHEATSHEET MEMORY M_{i+1} (updated)

```
[ctx-00011] section=General Meta-Reasoning Strategies usage=1
helpful=0 harmful=0 active=1 created=16 updated=16 :: Track
multi-step position swaps by updating records after each swap.
Example: Fred and Bob swap -> Fred is right winger, Bob is
goalkeeper; Bob and Claire swap -> Bob is right midfielder, Claire
is goalkeeper; Alice and Eve swap -> Alice is cheerleader, Eve
is center midfielder; Dave and Claire swap -> Dave is goalkeeper,
Claire is benchwarmer; Fred and Alice swap -> Fred is cheerleader,
Alice is right winger; Bob and Fred swap -> Bob is cheerleader,
Fred is right midfielder; Alice and Gertrude swap -> Alice is left
winger, Gertrude is right winger. Final position of Fred is right
midfielder.
```

Final answer: (C) right midfielder Status: Correct Tokens: 24,293 Model: Qwen3-8B