

Collaborative Dynamic Cheatsheet: Multi-Agent Test-Time Learning with Small Language Models

Stanford CS224N Custom Project
Mentor: Mirac Suzgun

Malvyn Lai
Department of Computer Science
Stanford University
malvlai@stanford.edu

Erica Wang
Department of Computer Science
Stanford University
ericajw@stanford.edu

Team member	Late days remaining
Malvyn Lai	1
Erica Wang	1
Total pooled	2
Team size	2
Late days applied for the report	1

Abstract

Language models (LMs) typically solve reasoning tasks independently, preventing them from reusing insights across related problems and making them prone to repeating mistakes. The Dynamic Cheatsheet (DC) framework addresses this limitation by enabling test-time learning through a small external memory that stores reusable strategies discovered during inference. While DC has shown strong improvements for single frontier models, its effectiveness for smaller open-source models remains unclear.

We investigate whether collaborative inference can make test-time memory useful for smaller LMs. We introduce a *Multi-Agent Dynamic Cheatsheet* setting in which three generators (Qwen2.5-7B-Instruct, Qwen2.5-14B-Instruct, and Mistral-7B-Instruct) solve problems collaboratively while maintaining evolving cheatsheets. We evaluate both shared and separate memory configurations on two challenging reasoning benchmarks, MMLU-Pro Engineering and GPQA Diamond, and compare against strong inference-time baselines such as self-consistency.

Naive DC often degrades performance for small models due to noisy or overly specific memory entries. However, multi-agent DC with separate per-generator memory achieves the best results, reaching 52.0% accuracy on MMLU-Pro Engineering and 44.9% on GPQA Diamond—up to 11 percentage points above the strongest single-model baseline. Overall, our results show that collaborative inference with structured memory can substantially improve reasoning for smaller language models when memory quality and models develop their own reusable strategies.

1 Introduction

Language models (LMs) have achieved strong performance on many natural language processing tasks, including complex reasoning and structured problem solving. However, these models typically operate in isolation, processing each input independently. As a result, when solving sequences of related problems, LMs often re-derive similar solutions and repeat the same mistakes instead of reusing insights from earlier tasks.

This observation motivates the *Dynamic Cheatsheet* (DC) framework (Suzgun et al., 2025), which allows LMs to retain and reuse useful insights discovered during inference across sequences of related tasks. DC operates entirely at test time without parameter updates or fine-tuning. It maintains a small external memory—the “cheatsheet”—that stores reusable strategies, reasoning patterns, or code snippets discovered during inference. A curator LM extracts useful insights from generated solutions and updates the cheatsheet, which is then provided as context for subsequent problems.

In this work, we extend DC to study whether test-time memory can improve reasoning performance for smaller language models on related science and engineering tasks through collaborative inference. We introduce a *Multi-Agent Dynamic Cheatsheet* setting where three generator LMs—Qwen2.5-7B-Instruct, Qwen2.5-14B-Instruct (Yang et al., 2024), and Mistral-7B-Instruct (Jiang et al., 2023)—independently produce candidate solutions. Answers are aggregated via majority voting, after which a curator LM (Qwen2.5-14B-Instruct) extracts reusable strategies to update the cheatsheet using the DC-Cu approach.

We evaluate two memory configurations. In the shared-memory setting, all generators consult and update a single cheatsheet. In the separate-memory setting, each generator maintains its own cheatsheet, allowing models to explore more diverse reasoning strategies. We also introduce a retrieval-based extension, DC-RS, which retrieves the top- k most similar previously solved problems ($k = 3$) and synthesizes a refined cheatsheet before generation.

We evaluate these methods on challenging reasoning benchmarks including MMLU-Pro Engineering and GPQA-Diamond, which contain difficult multi-step science and engineering problems. Unlike standard evaluations where problems are solved independently, we process datasets sequentially so models can accumulate insights across earlier questions.

We also compare against standard inference-time baselines including single-model prompting and self-consistency decoding, where multiple reasoning samples are generated and the final answer is chosen by majority vote. This helps isolate improvements due to persistent memory from those due purely to additional sampling.

Our experiments reveal several key findings. First, naively applying DC to small models can decrease performance: both Qwen2.5-7B and Qwen2.5-14B experience accuracy drops under DC-Cu due to noisy or overly specific memory entries that fail to generalize across questions. However, collaborative inference significantly improves results for small models; a multi-agent ensemble without memory already improves performance on MMLU-Pro Engineering through majority voting, which demonstrates the benefit of combining diverse generators. The strongest results are obtained with MultiGen-Cu using separate memory for generators, which reaches 52.0% accuracy on MMLU-Pro Engineering and 44.9% on GPQA Diamond—improvements of +11.0 and +9.5 percentage points over the strongest single-model baseline. In contrast, shared memory across generators consistently underperforms separate memory, suggesting that strategies learned by one model may not transfer cleanly to others. Overall, these findings indicate that test-time memory can improve reasoning for smaller language models when paired with collaborative inference and carefully managed memory updates.

2 Related Work

Our work incorporates and builds on **Dynamic Cheatsheet (DC)** (Suzgun et al., 2025), which introduces a test-time learning framework where a single LM maintains an evolving memory of reusable strategies discovered while solving earlier problems. The paper studies several variants, including DC-Cumulative (DC-Cu), which updates memory after each query; DC-RS, which retrieves similar past problems to refine memory before generation; and Dynamic Retrieval (DR), which retrieves past input-output pairs without curation (Suzgun et al., 2025). The retrieval mechanism follows the retrieval-augmented generation (RAG) framework introduced by Lewis et al. (2020). DC yields substantial gains for LMs: for example, Claude 3.5 Sonnet improves by 9% on GPQA-Diamond and 8% on MMLU-Pro Engineering and Physics, while GPT-4o improves from roughly 10% to 99% on the arithmetic Game of 24 by reusing discovered code-based strategies.

However, the original study focuses on larger models in a single-generator setting. The authors found that DC depends heavily on generator competence; weaker models may produce incorrect strategies that populate the cheatsheet with errors, limiting its effectiveness. This makes it challenging for smaller models to benefit, which motivates our use of an ensemble of weaker models to try and increase performance. The authors found DC-RS effective, consistently outperforming DC-Cu and simpler baselines, while DC-Cu performed strong across different reasoning benchmarks including AIME, GPQA-Diamond, and MMLU-Pro. This motivates our use of DC-Cu and DC-RS in our work.

Wang et al. (2022) propose self-consistency decoding, where a model generates multiple stochastic reasoning paths and the final answer is chosen by majority vote. This approach improves reasoning performance by marginalizing over diverse chains of thought, but it operates independently for each query. In our experiments, self-consistency serves as a baseline to distinguish improvements from persistent memory from gains due purely to additional sampling.

Our multi-agent setting is also related to language model debate (Du et al., 2023), where multiple LMs iteratively generate and critique candidate solutions before producing a final answer. This collaborative process has been shown to improve reasoning accuracy by allowing models to identify errors and refine answers through interaction. However, debate operates within a single problem instance and does not retain knowledge across queries. Our approach extends this idea by combining collaborative generation with Dynamic Cheatsheets, enabling strategies discovered by different agents to be stored and reused for future problems.

3 Approach

Our approach builds on the Dynamic Cheatsheet (DC) framework (Suzgun et al., 2025), which enables test-time learning through a small external memory of reusable strategies provided as context during inference. We extend DC in two directions in order to improve performance for smaller models. First, we replace the single generator with a multi-agent ensemble whose outputs are aggregated through majority voting. Second, we introduce an optional auditor that performs quality control on the evolving cheatsheet by pruning incorrect or overly specific entries. These extensions allow us to evaluate whether collaborative inference and improved memory quality can improve the effectiveness of DC for smaller language models.

3.1 Single-Model Baselines

We use single-model configurations as baselines to measure the effect of sampling diversity and test-time memory independently of multi-agent collaboration.

Baseline (BL). Standard prompting for a single LM without a cheatsheet; each question is solved independently, reflecting traditional inference.

Self-Consistency (SC). We also evaluate self-consistency decoding Wang et al. (2022) in the no-memory setting. For each question, the model samples $k = 5$ responses at temperature $\tau = 0.7$, and the final answer is selected by majority vote over the sampled outputs, where ties are resolved by choosing the highest-confidence answer. This baseline isolates the effect of sampling diversity within a single model, separating it from improvements due to memory or multi-agent collaboration.

DC-Cumulative (DC-Cu). In DC-Cu, the cheatsheet is updated cumulatively during inference. After generating solutions to questions, a curator model summarizes useful strategies from the batch’s question–answer pairs and appends them to the cheatsheet. For single-model DC-Cu, we utilize batching 16 questions, and sequentially curate our cheatsheet twice with each curator updating the cheatsheet with 8 of the previous questions. The updated cheatsheet is then prepended to the generator’s prompt for subsequent questions. Because the full cheatsheet is always included, memory is cumulative rather than query-specific. Implementation details follow Suzgun et al. (2025).

DC-Retrieval and Synthesis (DC-RS). DC-RS instead constructs a query-specific cheatsheet before generation. For each question, the system embeds the input query and retrieves the top- k most similar past question–answer pairs using cosine similarity ($k = 3$). A curator then synthesizes a compact cheatsheet from these retrieved examples, which is provided to the generator for solving the current query. Unlike DC-Cu, memory is selective and adapted to each query. Further implementation details follow Suzgun et al. (2025).

3.2 Auditor

We introduce an optional auditor that performs quality control on the cheatsheet independently of the curator. After every batch of 16 questions, the auditor reviews the current cheatsheet and removes entries that are logically incorrect, mathematically unsound, or redundant. The auditor does not add new content, and instead only prunes existing entries, ensuring that strategy synthesis remains the responsibility of the curator while the auditor enforces memory quality. This additional filtering is motivated by the limited editing ability of DC-Cu, where new entries are appended but incorrect or overly specific strategies may persist in memory. We therefore use a stronger model, Qwen2.5-14B-Instruct, as the auditor so it can more reliably identify and remove flawed strategies without introducing new ones. We evaluate the auditor as an addition on top of the single-model baseline and DC-Cu to test whether cheatsheet quality, rather than quantity, affects performance.

3.3 Multi-Agent Extension

We extend DC to a multi-agent ensemble of $K = 3$ generator models in order to investigate whether collaborative inference improves reasoning performance for smaller models. Using multiple generators increases the chance that at least one model produces a correct reasoning chain, while majority voting can reduce individual model errors. Within this framework we evaluate two memory settings.

MultiGenerator (MultiGen). In this baseline, the cheatsheet is omitted ($M_i \equiv \emptyset$) and each generator answers the question independently. The final answer is determined by majority vote across generators. This configuration isolates the effect of ensemble inference without any memory-based adaptation.

MultiGenerator-Cumulative (MultiGen-Cu). We next enable cumulative cheatsheet memory in the multi-agent setting. After each question, the curator updates the cheatsheet based on the generator outputs, and the updated memory is provided to the generators for subsequent questions. We study two memory-sharing configurations:

- *Shared memory*: A single cheatsheet is maintained for all generators. After each question, the generators independently produce candidate answers, which are aggregated by majority vote. The curator then updates the shared cheatsheet using the generator outputs, and the updated memory is provided to all generators for the next question.

Explicitly, for questions $1, \dots, n$:

1. $M_0 \leftarrow \emptyset$ ▷ Initialise shared memory
2. **for** $i \in \{1, \dots, n\}$ **do**
3. $\tilde{y}_i^{(1)}, \dots, \tilde{y}_i^{(K)} \leftarrow \text{Gen}_k(x_i, M_{i-1})$ ▷ Independent generation
4. $\hat{y}_i \leftarrow \text{Aggregate}(\tilde{y}_i^{(1)}, \dots, \tilde{y}_i^{(K)})$ ▷ Majority vote
5. $M_i \leftarrow \text{Cur}(M_{i-1}, x_i, \{\tilde{y}_i^{(k)}\}_{k=1}^K)$ ▷ Update shared cheatsheet
6. **end for**

- *Separate memory*: Each generator maintains its own cheatsheet, which is updated from its own outputs by the curator. The generators therefore accumulate independent memories while their final answers are still aggregated through majority vote. This configuration tests whether generators benefit more from a shared memory or from maintaining independent reasoning strategies.

3.4 Verifier

In the multi-agent setting, a verifier is invoked when the generators fail to produce a majority answer or when the majority output is not a valid answer (e.g., “No final answer found”). The verifier, implemented using Qwen2.5-14B-Instruct, receives the question and all generator outputs and selects a single answer. Rather than solving the problem from scratch, the verifier acts as a tie-breaker by evaluating the reasoning presented in each generator response and selecting the most consistent answer.

3.5 MultiGenerator-Retrieval and Synthesis (MultiGen-RS)

Finally, we combine the retrieval-and-synthesis memory scheme with the multi-agent ensemble. For each question, relevant past question–answer pairs are retrieved and a curator synthesizes a compact cheatsheet as in DC-RS. This curated context is provided to all generators before generation, and their outputs are aggregated by majority vote (with verifier fallback when necessary). This configuration tests whether retrieval-based memory further improves performance when combined with ensemble inference.

4 Experiments

4.1 Data

We evaluate on two challenging multiple-choice benchmarks that represent distinct reasoning demands.

MMLU-Pro Engineering. A subset of MMLU-Pro Wang et al. (2024) restricted to engineering questions. MMLU-Pro extends the original MMLU benchmark by increasing the number of answer choices from four to ten, significantly reducing the chance of correct guessing and demanding more precise discrimination. Engineering questions cover topics such as electrical circuits, mechanics, thermodynamics, and materials science—domains where structured problem-solving strategies and formula recall can plausibly be encoded in a cheatsheet. We evaluate on 200 randomly sampled questions (shuffle seed 10).

GPQA Diamond. The *Graduate-Level Google-Proof Q&A Diamond* split Rein et al. (2024) consists of 198 expert-level multiple-choice questions in biology, chemistry, and physics, validated to be difficult for non-expert humans even with internet access. Questions require deep domain reasoning rather than surface-level pattern matching, making this a high-difficulty probe of whether test-time memory adaptation can help at the frontier of model capability. We evaluate on all 198 available questions.

Both datasets use a multiple-choice format (MCQ), where the model must select one answer from a fixed set of options. The input to all models is the question text together with its answer choices, optionally prepended by the current cheatsheet. The output is a single letter identifying the selected choice.

4.2 Evaluation Method

Metric. For both datasets we use *accuracy*: the fraction of questions for which the model’s final selected answer matches the ground-truth label. Because both benchmarks are multiple-choice with unambiguous gold labels, accuracy is a precise and directly interpretable metric. For multi-agent runs, the “model answer” is the majority-voted answer (or the verifier’s answer when no majority is reached).

Answer extraction. We utilize the original paper’s parsing mechanisms We use *Soft Match* (SM) Suzgun et al. (2025) for both datasets: a predicted answer is counted as correct if it matches the gold option letter up to minor formatting differences (e.g., surrounding whitespace, punctuation, or capitalisation variation). This avoids penalising models for stylistic variation in answer formatting, which is especially relevant for smaller models that may not reliably output a bare letter.

Self-consistency. For the self-consistency baseline, we sample $k = 5$ responses per question at temperature $\tau = 0.7$ and take the majority answer. All other experiments use greedy decoding ($\tau = 0.0$) to ensure deterministic, reproducible outputs.

Cheatsheet quality analysis. Beyond accuracy, we instrument all cheatsheet-bearing runs with verbose audit logging. This saves per-question cheatsheet snapshots, token growth curves, and per-entry reuse counts. These logs allow us to diagnose failures in curation, such as overly specific or noisy entries, low reuse rates, and abstraction mismatches between curator and generator that may explain cases where DC-Cu does not improve over the baseline. When the auditor is enabled, each run additionally reports the total number of words pruned and the number of auditor passes executed.

4.3 Experimental Configurations

All runs use the vLLM backend with 4-bit quantisation, a maximum of 2,048 output tokens, and a cheatsheet word limit of 3,000 words. DC-Cumulative and MultiGen-Cu operate on batches of 16 questions, with the curator processing sub-batches of 8 question-answer pairs per call. DC-RS retrieves the top $k = 3$ most similar past pairs using cosine similarity over the original Dynamic Cheatsheet paper’s OpenAI text-embedding-3-small embeddings.

Single-model runs. Generators: Qwen2.5-7B-Instruct and Qwen2.5-14B-Instruct. For DC-Cu and DC-RS, the curator is the same model as the generator (self-curation). Self-consistency runs use the same models with $k = 5$ samples.

Multi-agent runs. Generators: Qwen2.5-7B-Instruct, Qwen2.5-14B-Instruct, and Mistral-7B-Instruct-v0.2. Curator and verifier: Qwen2.5-14B-Instruct. This ensemble is intentionally heterogeneous: two models of the same family but different scales, and one model from a different architecture family. We hypothesize that architectural and scale diversity increases the probability that at least one generator produces a correct reasoning chain, while the shared cheatsheet allows successful strategies to propagate across the ensemble.

5 Results

Table 1 reports accuracy and per-baseline deltas for all conditions. Figure 1 reports cumulative accuracy trajectories over the 198 GPQA Diamond questions.

The best result scored is **MultiGen-Cu with separate (non-shared) memory**, which achieves **52.0%** on MMLU and **44.9%** on GPQA, the highest scores in the experiment on both benchmarks. This represents gains of +11.0% and +9.5% over the respective 14B baselines, and +15.5% and +15.1% over the 7B baselines.

Other configurations do not fare as well, with *plain DC-Cumulative hurting accuracy for both models on both datasets* without the auditor, with drops ranging from −1.0 to −4.0%.

The auditor improves DC-Cu on the 14B model (+4.5% on MMLU, +5.0% on GPQA over baseline), suggesting that the failure mode could be noisy curation rather than an inherent incompatibility between DC and small models.

5.1 Single-Model Baselines

The 7B and 14B default baselines score 36.5% / 41.0% on MMLU and 29.8% / 35.4% on GPQA, respectively, confirming the expected 4–6 percentage gap between model sizes. Self-consistency (SC5) consistently improves greedy decoding: the 7B model gains +1.5 on MMLU and +6.1% on GPQA; the 14B model gains +10.0% on MMLU and +2.5% on GPQA. The much larger SC5 benefit for 14B on MMLU (vs. 7B) suggests that the 14B model has sufficient base capability to produce diverse high-quality responses across MMLU’s engineering subtopics, so sampling five times reliably surfaces the correct answer. On GPQA, the gains are modest for both models (35.9% and 37.9%), consistent with the benchmark’s difficulty leaving little headroom for within-model diversity to help.

5.2 DC-Cumulative and the Auditor

Without the auditor, DC-Cu degrades accuracy relative to the baseline for all single-model configurations: −4.0 / −1.0 percent for 7B on MMLU / GPQA and −2.0 / −1.1 percent for 14B.

Introducing the auditor changes the picture markedly for the 14B model: DC-Cu + Aud achieves 45.5% on MMLU (+4.5% over 14B baseline, +6.5% over plain 14B DC-Cu) and 40.4% on GPQA (+5.0% over baseline, +6.1% over plain DC-Cu). This is the

Table 1: Accuracy (%) across all 30 experimental conditions on MMLU-Pro Engineering (200 samples) and GPQA Diamond (198 samples). **Bold**: best result per dataset column overall. **Underline**: best result within each category block. **SC5**: self-consistency $k=5$, $\tau=0.7$. **DC-Cu**: DC-Cumulative. **DC-RS**: Retrieval & Synthesis. **Aud**: runs with an additional auditor model that runs every 16 questions, pruning the cheatsheet of logical fallacies. **Sh**: for multi-agent runs, indicates the usage of shared memory (one collaborative cheatsheet among the three generators). **Sep**: for multi-agent runs, indicates the usage non-shared memory (three separate cheatsheets for each generator). ΔBL : accuracy gain over the same-size single-model baseline, set to the corresponding 7B or 14B default, or 14B default for multi-agent runs.

Category	Configuration	MMLU	ΔBL	GPQA	ΔBL
Single Model	7B Default (BL)	36.5	—	29.8	—
	14B Default (BL)	41.0	—	35.4	—
	<u>7B SC5</u>	38.0	+1.5	35.9	+6.1
	<u>14B SC5</u>	51.0	+10.0	37.9	+2.5
DC-Cumulative	7B DC-Cu	32.5	−4.0	28.8	−1.0
	14B DC-Cu	39.0	−2.0	34.3	−1.1
	7B DC-Cu + Aud	31.0	−5.5	30.3	+0.5
	<u>14B DC-Cu + Aud</u>	45.5	+4.5	<u>40.4</u>	+5.0
DC-RS	7B DC-RS	34.0	−2.5	30.3	+0.5
	<u>14B DC-RS</u>	40.5	−0.5	<u>36.4</u>	+1.0
Multi-Agent	MultiGen (no memory)	46.5	+5.5	31.3	−4.1
	MultiGen-Cu (Sh)	39.5	−1.5	37.9	+2.5
	MultiGen-Cu (Sep)	52.0	+11.0	44.9	+9.5
	MultiGen-RS (Sh)	42.0	+1.0	36.4	+1.0
	MultiGen-RS (Sep)	45.5	+4.5	33.8	−1.6

best single-model result on GPQA in the experiment. For the 7B model, the auditor provides no net benefit (31.0% vs. 32.5% plain DC-Cu on MMLU), suggesting with the 7B curator producing so little sound content that trimming has nothing useful to preserve.

5.3 DC-RS

DC-RS performs similarly to or slightly below the default baseline for both models: 7B DC-RS scores 34.0% / 30.3% (vs. 36.5% / 29.8% baseline) and 14B DC-RS scores 40.5% / 36.4% (vs. 41.0% / 35.4%), all on MMLU and GPQA, respectively. The retrieval mechanism—which was highly effective for frontier models in the original paper—does not help small models here, because retrieved past Q&A pairs are of low quality and the synthesized cheatsheet adds noise rather than useful context. MultiGen-RS (shared memory) recovers to 42.0% / 36.4%, but this is still substantially below plain MultiGen (46.5% / 31.3%) and MultiGen-Cu separate (52.0% / 44.9%), confirming that retrieval adds little value given the quality constraints of small-model curation.

5.4 Multi-Agent Ensemble

The no-memory MultiGenerator ensemble achieves 46.5% on MMLU (+5.5% over 14B baseline) through model heterogeneity and majority voting alone, confirming that combining a 7B, a 14B, and a Mistral-7B yields meaningful diversity. On GPQA, the ensemble (31.3%) underperforms the 14B baseline (35.4%), a result we did not expect: when all three generators individually struggle on extremely hard questions, majority voting amplifies the collective error rather than cancelling individual mistakes.

Adding the cumulative shared cheatsheet (MultiGen-Cu Sh) recovers GPQA performance to 37.9% but collapses MMLU to 39.5%—a −7.0% drop relative to no-memory MultiGen. Switching to separate memory resolves both problems simultaneously and then exceeds all other configurations: 52.0% on MMLU and 44.9% on GPQA. The separate memory advantage is 12.5% on MMLU and 7.0% on GPQA versus shared.

6 Analysis

6.1 Auditor Discrepancy with 14B Model versus 7B

The auditor’s trimming pass—which removes logically unsound, redundant, and overly specific entries without adding new content—unlocks DC gains for the 14B model (+4.5% MMLU, +5.0% GPQA over baseline) but provides no benefit to the 7B model. This asymmetry reveals a *minimum quality threshold* for auditor effectiveness.

For the 14B model, the curator writes a mix of useful general heuristics and problem-specific noise. The auditor removes the noisy entries, leaving a compact cheatsheet of genuinely transferable strategies. The 14B generator can then leverage these entries, recovering the accuracy loss from unaudited DC-Cu and surpassing the baseline.

For the 7B model, the curator writes almost exclusively low-quality entries: problem-specific numbers, shallow one-sentence summaries, or structurally malformed items. After auditor pruning, so little remains that the effective cheatsheet is nearly empty—only 2 sample questions remain, equivalent to no memory at all but with the computational overhead of curation. The auditor’s benefit is therefore gated on a minimum floor of curator quality, which the 7B model does not reach.

6.2 Separate vs. Shared Memory

The 12.5% MMLU gap and 7.0% GPQA gap between separate and shared memory in the multi-agent setting is the largest effect in our experiment.

In the shared memory, the Qwen2.5-14B-Instruct curator must synthesize a single cheatsheet entry from outputs produced by three models with different architectures, parameter counts, and error profiles. For example, in one question, the 7B generator misidentifies an RLC circuit’s resonant frequency while the 14B generator computes it correctly. Here, the curator has no ground truth to arbitrate—it must produce an entry acceptable to all three generators for the next question, which may be in an entirely different subdomain. The result is a cheatsheet entry that is either too generic to be useful or incorrectly weighted toward the wrong answer.

In the separate setting, each generator’s cheatsheet is curated from its own outputs only. The 7B generator’s memory captures its own consistent vocabulary and reasoning patterns; the 14B generator’s memory captures its own, independently. Majority voting at inference time then benefits from both each generator’s individual test-time learning *and* their architectural diversity—two orthogonal sources of improvement that compound multiplicatively.

The gap is larger on MMLU (12.5%) than on GPQA (7.0%). We attribute this to MMLU’s higher within-benchmark topic diversity: questions span circuits, thermodynamics, fluid mechanics, materials science, and probability, and have a wider range of technical techniques, meaning a shared cheatsheet conflates strategies from differing domains. GPQA is more uniformly difficult in reasoning style even across biology, chemistry, and physics, meaning a shared meta-strategy can transfer across questions better.

6.3 Batching and Practical Limitations

A significant practical limitation of the DC framework is the requirement for sequential processing. Unlike standard batch inference, where all questions can be processed in parallel, traditional DC-Cu processes questions in order: question $i + 1$ cannot begin until question i ’s answer has been generated and the curator has updated the cheatsheet. In the multi-agent setting with three generators and a separate curator, this means four model forward passes must complete before each cheatsheet update, serializing what could otherwise be a parallel workload. Similarly, the calls of the verifier and optionally the auditor slow down this process even further.

In practice, this means our full multi-agent DC runs took substantially longer than equivalent batch inference jobs.

Future work could explore asynchronous curation—updating the cheatsheet in the background while the next batch is already processing, and also investigate how much batching impacts the effectiveness of the cheatsheet.

6.4 Cheatsheet Length and Content Quality

The saved cheatsheet data reveals that cheatsheet length is not the binding constraint on performance—content quality is. In four of six runs with saved cheatsheets,¹ the cheatsheet stays near its initial size (~ 200 – 340 tokens) for the entire run: it does not grow toward the 3,000-word cap, because the curator either fails to update at all or produces malformed entries that do not survive. One of the other evolving runs (GPQA MultiGen-Cu) grows to $\sim 2,700$ tokens, with 60% of questions seeing

¹We saved cheatsheets from DC-Cu with an auditor for 7b, DC-Cu with an auditor for 14b, and MultiGen-Cu (Shared) on both datasets.

a cheatsheet over 2,000 tokens. At this length, the “lost in the middle” problem becomes relevant for 7B models, which have well-documented difficulty attending to information in the middle of long contexts.

The quality dimension is more revealing. Useful entries in the evolving GPQA run tend to be formula-level generalizations with symbolic rather than numeric content:

Luminosity ratio of two stars: $L_1/L_2 = (R_1/R_2)^2$, assuming equal surface temperature (Stefan-Boltzmann law).

Harmful entries encode specific answers or use placeholder text from the curator prompt template that was not correctly filled:

[Briefly describe the problem context, purpose, and key aspects of the solution.]
...correct answer: 3536.

The auditor is specifically designed to catch and remove the latter category. The frequency of such entries and the resulting auditor benefit is higher for the 14B model than the 7B model, which tends to produce low-quality and structurally malformed entries that are harder to reason about even qualitatively.

6.5 Model Capability Thresholding

Across all conditions, a clear capability threshold separates models that benefit from DC from those that do not. The 7B model benefits from *none* of the DC variants on MMLU (all configurations score below or equal to its 36.5% baseline) and from only SC5 and the 14B-curated multi-agent system on GPQA. The 14B model benefits from SC5, the audited DC-Cu, and all multi-agent configurations.

This threshold effect aligns with the original paper’s finding that “DC amplifies the strengths of models that can already produce high-quality outputs, but not fix foundational gaps in reasoning.” A model that cannot reliably solve a problem on the first attempt also cannot reliably *curate* a useful memory entry from its failed attempt. The auditor mitigates this by removing bad entries, but it cannot create good entries from nothing.

The practical implication is that DC should be applied selectively. For tasks and models where the baseline accuracy exceeds roughly 35–40%—suggesting the model has the domain knowledge to sometimes produce correct, generalizable solutions—DC with separate memory and an auditor is likely to help. Below this threshold, self-consistency or a larger model are more reliable inference-time improvements.

7 Conclusion

We investigated whether the Dynamic Cheatsheet (DC) framework can improve reasoning performance for smaller open-source language models and whether collaborative inference helps make test-time memory more effective. We evaluated DC on two challenging benchmarks, MMLU-Pro Engineering and GPQA Diamond, comparing the single-model with multi-agent configurations.

Our results show that naive application of DC does not reliably benefit smaller models. In particular, DC-Cumulative often decreases performance because noisy or overly specific strategies accumulate in the cheatsheet. However, collaborative inference significantly improves results. A multi-agent ensemble with separate per-generator memory achieves the best performance, reaching 52.0% accuracy on MMLU-Pro Engineering and 44.9% on GPQA Diamond—up to 11 percentage points above the strongest single-model baseline.

We find that both memory structure and memory quality are critical. Shared memory across generators often harms performance, while maintaining separate memory for each generator allows models to accumulate strategies that remain internally consistent with their own reasoning behavior. Similarly, introducing an auditor that removes incorrect or redundant entries improves DC-Cu performance for the stronger 14B model, demonstrating that controlling memory quality is essential for effective test-time learning.

Our experiments suggest that the benefits of Dynamic Cheatsheets observed for frontier models do not directly transfer to smaller models without additional mechanisms for collaboration and memory control. Experiments are limited to two datasets and a small set of models, and the sequential nature of DC makes evaluation computationally expensive. Future work could explore larger, stronger curator or verification models to improve the quality of cheatsheet entries, since our results show that noisy memory limits the effectiveness of DC for smaller models. Another possible direction is to organize memory by problem type so that generators retrieve strategies that are more relevant to the current question.

8 Team Contributions

The work was evenly divided.

References

- Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*.
- Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.
- Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*.
- David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2024. Gpqa: A graduate-level google-proof q&a benchmark. In *Proceedings of the Conference on Language Modeling (COLM)*.
- Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. 2025. Dynamic cheatsheet: Test-time learning with adaptive memory. *arXiv preprint arXiv:2504.07952*.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.
- Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyang Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhui Chen. 2024. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems (NeurIPS)*.
- An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

9 Appendix

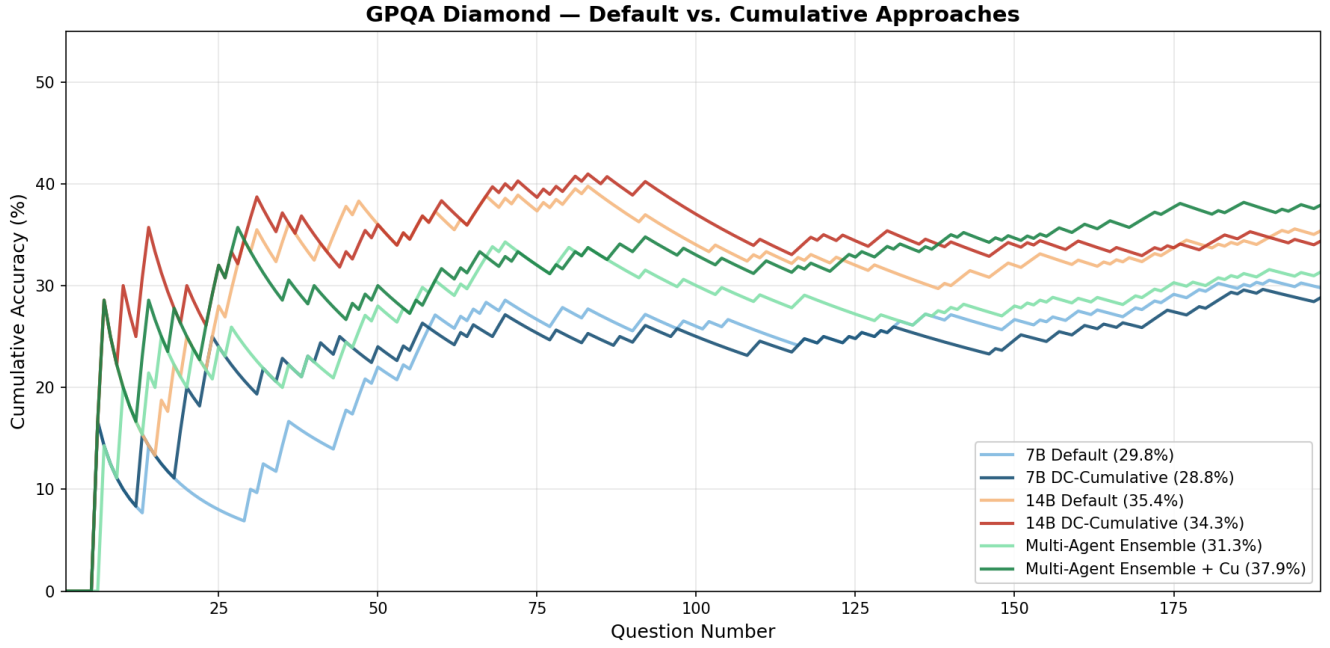


Figure 1: Cumulative accuracy on GPQA Diamond for single-model baselines, DC-Cumulative, and MultiGen-Cu (shared memory). The DC-Cu curves for both 7B and 14B track *below* their respective baselines from approximately question 30 onward, reflecting the frozen, first-question cheatsheet’s growing irrelevance. Multi-Agent + Cumulative (37.9%) is the only curve that persistently exceeds the 14B baseline in the second half of the run. Note: the non-shared memory variant (M6, 44.9%) is not plotted but achieves the best GPQA result in the experiment.