Augmenting Language Agents with Parametric Memory

Anonymous Author(s)

Affiliation Address email

Abstract

Large Language Models (LLMs) have demonstrated strong reasoning abilities, yet existing agent frameworks remain constrained by two limitations. First, they typically operate at the per-instance level, confining signals to individual problems and overlooking transferable patterns across tasks. Second, while some approaches attempt to incorporate global information through external memory, these are nonparametric in nature, and thus capture only shallow interactions across instances, failing to uncover deeper regularities. To overcome these limitations, we propose ParamAgent, a language agent framework that leverages a domain-adaptive parametric memory to internalize knowledge across samples into model parameters. In addition to capturing cross-sample regularities, ParamAgent provides twofold flexibility: (i) the parametric module can supply different forms of knowledge depending on various domains, and (ii) the same module can be integrated with different base LLMs, making ParamAgent broadly applicable. Moreover, ParamAgent naturally promotes diversity of outputs by adjusting the sampling temperature of the parametric module. Experiments on programming, math reasoning, and multi-hop question answering benchmarks show that ParamAgent consistently outperforms state-of-the-art baselines, surpassing the best baseline by up to 7.90%, 9.41%, and 24.30% respectively.

1 Introduction

3

5

6

8

10

11

12

13

14

15

16 17

18

19

Large language models (LLMs) [6, 10, 31] have exhibited striking progress in complex reasoning 20 tasks. Their ability to interleave reasoning with actions has led to the development of autonomous 21 language agents that treat an LLM as the core policy[37, 28, 43, 42, 50, 22, 38, 32, 15, 46]. For 23 example, Chain-of-Thought (CoT) prompting [37] elicits explicit intermediate steps that improve reasoning performance on complex tasks. Self-Refine [22] introduces an iterative self-feedback loop, 25 enabling models to progressively refine their outputs and achieve higher-quality results. Subsequent work expands agents' search and feedback mechanisms: Reflexion [28] stores verbalized feedback 26 in episodic memory (i.e., a long-term memory of the agent's self-reflections accumulated across 27 iterations) and yields noticable gains; Tree-of-Thoughts (ToT) [42] explores multiple reasoning paths 28 via tree search; LATS integrates Monte-Carlo Tree Search for long-horizon planning [50, 7]; Diversity 29 of Thought (DoT) [19] proposes to promote diversity for reflective feedbacks. 30

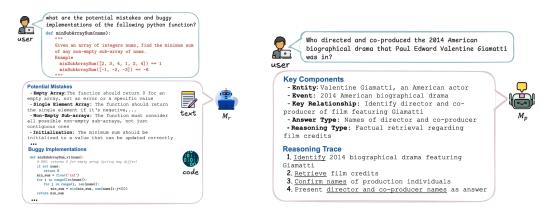
Despite these advances, current language agent frameworks remain limited by their localized perspective. For instance, self-reflection operates at the per-instance level, confining their reasoning signals to individual problems without exploiting connections across similar problem structures. As a result, agents miss **population-level** insights about shared failure modes and underlying cross-instance associations. Second, although recent frameworks equip agents with mechanisms for storing past experiences and retrieving them when solving new tasks [5, 27, 49, 19], allowing them to leverage global information, these approaches typically rely on non-parametric memory (e.g., textual logs or

retrieval-augmented stores). Consequently, they capitalize on **shallow** interactions across instances,

- which may fail to uncover deeper regularities or to extend beyond embedding-level similarities. 39
- One natural choice is to fine-tune the base LLM in the agent to internalize the cross-sample knowl-40
- edge. However, this poses two challenges: (1) a distribution mismatch, and (2) general capability 41
- degradation. Further detailed discussions on the challenge of fine-tuning the agent are provided in 42
- Appendix C. 43
- To overcome these limitations, we propose ParamAgent, a language agent framework that utilizes
- parametric knowledge. ParamAgent adopts a external module M_* (* indicates that the module can 45
- generate different forms of knowledge, adapting to various domains) to internalize knowledge across 46
- samples. When solving a new problem, the agent queries M_* to obtain population-level insights 47
- rather than instance-level feedback. For example, M_r in ParamAgent can synthesize potential 48
- mistakes for a programming and math task, whereas M_p generates semantic decompositions for 49
- multi-hop question answering (QA) (Sec. 3). Since the knowledge is encoded directly into the M_* 's 50
- model parameters via a training process, it captures deep interactions across samples, enabling it to
- synthesize insights that capture cross-sample similarities, which is in contrast with previous retrieval
- based methods that recall a handful of past trajectories as external memory. 53
- In addition to capturing cross-sample regularities, ParamAgent offers twofold flexibility. First, the 54
- parametric module can supply different forms of knowledge, such as generating population-level 55
- reflective feedback signals to augment self-reflection, or decomposing complex queries into semantic 56
- units as structured intermediate thoughts. Second, the same module can be integrated with different 57
- base LLMs, making ParamAgent broadly applicable and adaptable across various base LLMs.
- Moreover, by adjusting the sampling temperature of the parametric module, ParamAgent naturally 59
- promotes diversity of the outputs. 60
- We evaluate ParamAgent on math reasoning problems, programming, and multi-hop QA. In each 61
- domain, ParamAgent significantly outperforms state-of-the-art methods. Concretely, our approach 62
- surpasses the best baseline by up to 7.90% on programming, 9.41% on math reasoning, and 24.30%63
- on multi-hop QA. Our contributions can be summarized as follows:
- · We identify key limitations in existing agent frameworks and propose leveraging parametric 65 knowledge to capture cross-sample interactions, thereby augmenting the reasoning process. 66
- · We propose ParamAgent, a language agent that equips a parametric module to capture cross-67 sample regularities, and further introduce ParamAgent-plus, an enhanced variant that integrates 68 multiple forms of memory modules. 69
- The parametric module M_* is capable of synthesizing multiple forms of knowledge that support 70 adaptation to a wide range of domains, and it can be flexibly integrated with different base LLMs 71 in the agents. 72
- Through extensive experiments on programming, math reasoning, and multi-hop QA, ParamAgent 73 consistently outperforms state-of-the-art baselines, surpassing the second best by up to 7.90%, 74 9.41%, and 24.30% respectively. 75

Preliminaries

- 77 We consider a pretrained Language Model (LM) p_{θ} with parameters θ that operates on token
- sequences. Let $x=(x[1],\ldots,x[l_x])$ denote the input sequence and $y=(y[1],\ldots,y[l_y])$ the 78
- output sequence. The LM decodes autoregressively, i.e., $p_{\theta}(y \mid x) = \prod_{i=1}^{y} p_{\theta}(y[i] \mid x, y[1:i-1])$, and, more generally, with an auxiliary prompt π (e.g., instructions, exemplars, tool feedback, etc.), 79
- $p_{\theta}(y \mid x, \pi) = \prod_{i=1}^{l_y} p_{\theta}(y[i] \mid x, \pi, y[1:i-1])$. We use z_1, \dots, z_n to denote intermediate thoughts, and r_1, \dots, r_k to denote self-reflections. A node in a search tree is written as $s = [x, z_{1:i}]$.
- 82
- **Input-Output (IO) prompting** The LM is prompted with task instructions and/or few-shot IO 83 pairs and directly produces the final output: $y \sim p_{\theta}(\cdot \mid x, \pi_{IO})$.
- **Chain-of-Thought (CoT)** To handle $x \mapsto y$, CoT [37] instructs the model to first generate a 85
- sequence of thoughts and then the answer: $z_i \sim p_{\theta}(\cdot \mid x, z_{1:i-1}), \quad y \sim p_{\theta}(\cdot \mid x, z_{1:n})$. In practice,
- $[z_{1:n}, y]$ is sampled as a single contiguous sequence.



- (a) An output example of M_r on programming task using HumanEval dataset.
- (b) An output example of M_p on multi-hop QA task using HotpotQA dataset.

Figure 1: Illustration of the output produced by M_*

Reflexion Reflexion [28] augments the prompt with episodic self-reflections $r_{1:k}$ from the previous k iterations. The agent then generates new solutions conditioned on the previous feedbacks: $y \sim p_{\theta}(\cdot \mid x, r_{1:k})$. Intuitively, r_i provides textual semantic gradient signals [28, 45], indicating common errors to avoid and corrective cues.

- Diversity of Thoughts (DoT) DoT [19] enhances the diversity of reflection feedback by using explicit prompt-level instructions to generate a set of diversified reflections $\{r_i\}_{i=1}^k$, thereby reducing redundancy and improving coverage of solutions. The decoding objective remains: $y \sim p_{\theta}(\cdot \mid x, r_{1:k})$.
- Tree-of-Thought (ToT) and LATS ToT [42]lifts CoT into a search over partial solutions $s = [x, z_{1:i}]$. New thoughts are proposed via CoT-style sampling $z_i \sim p_{\theta}(\cdot \mid x, z_{1:i-1})$, while DFS/BFS is used to explore the search tree. LATS [50] extends this view with Monte-Carlo Tree Search (MCTS) [7], repeatedly selecting, expanding, simulating, and backpropagating values over nodes s, thereby constructing high-value trajectories of thoughts leading to a more probable correct y.

3 Augmenting Language Agents with Parametric Knowledge

In this section, we show how ParamAgent leverages parametric knowledge to augment LLM-based agents. Depending on the domain, ParamAgent employs different forms of parametric knowledge. Speficically, (1) A reflection-oriented module M_r to synthesize model-based reflection for programming and math, and (2) A decomposition-oriented module M_p to produce semantic units for multi-hop QA. The detailed pseudo-code can be found in Appendix B, a shorter version can be found in Algorithm 1.

3.1 Global-Local Reflection

101

108

To incorporate cross-sample reflective signals beyond instance-specific cues, we propose a global-local reflection mechanism. The key idea is to combine self-reflections derived from episodic memory with global reflections synthesized by a parametric module M_r .

Training M_r . We obtain M_r by fine-tuning a pretrained LLM on a curated dataset where reflective feedback is provided as supervision. Through this process, the module internalizes population-level patterns into its model parameters, and learns to synthesize model-based reflections and corrective cues. We provide more details regarding the dataset curation and training in Appendix D.2. An example output from M_r is shown in Figure 1a.

Formulation. Having obtained M_r , ParamAgent conditions jointly on two sources of feedback:

$$y \sim p_{\theta}(\cdot \mid x, r_{1:k}, r_k^g), \quad r_k^g \sim p_{\psi}(\cdot \mid x), \tag{1}$$

where $r_{1:k}$ denotes k self-reflections collected across iterations, and r_k^g is the global reflection generated by M_r at the k-th iteration.

Usage. In each episode, M_r samples a global reflection r_k^g , which is injected into the prompt alongside self-reflections from the memory. A low sampling temperature is used in the first round to ensure informative feedback, while later rounds adopt a higher temperature to promote diversity. Importantly, r_k^g is used only as a transient input and is not stored in memory; the episodic memory only maintains self-reflections generated by the agent itself.

3.2 Semantic Decomposition

125

151

160

161

162

163

164

165

Beyond reflections, the parametric mod-126 ule can also generate structured knowl-127 edge. Inspired by chunking and the 128 working memory model from cognitive 129 science [23, 2], we introduce M_p to 130 decompose complex multi-hop queries 131 into compact semantic units that guide 132 reasoning, one such example is illus-133 trated in Figure 1b. 134

Training M_p . Similar to M_r , we finetune a pretrained LLM where semantic decompositions (e.g., entities, relations, constraints, answer types, etc.) serve as the training signal. Details are deferred in Appendix D.2.

Formulation. ParamAgent then conditions jointly on the original query x, the semantic units Z, and a set of self-reflections $r_{1:k}$ derived from prior attempts, with the final answer produced as:

$$y \sim p_{\theta}(\cdot \mid x, Z, r_{1:k}). \tag{2}$$

```
Algorithm 1: Pseudocode for ParamAgent
Require: Dataset \mathcal{D}, base LM p_{\theta}, parametric module M_*
      with params \psi, max iterations T_{\rm max}
  1: M ← Ø
                                                       ▶ Initialize memory
 2: for x \in \mathcal{D} do
 3:
           for t=1 to T_{\rm max} do
                T \leftarrow \begin{cases} 0.2 & \text{if } t = 1\\ 1.0 & \text{otherwise} \end{cases}
 4:
                 G_{t-1} \sim p_{\psi}(\cdot \mid x; T)
 5:
 6:
                 r_{1:t-1} \leftarrow \text{RETRIEVEREFLECTIONS}(\mathcal{M}, x)
 7:
                 y_t \sim p_\theta(\cdot \mid x, r_{1:t-1}, G_{t-1})
 8:
                 if EVALUATE(y_t, x) then
 9:
                      break
10:
                 else
11:
                      r_t \leftarrow \text{GENERATESELFREFLECTION}(y_t)
12:
                      \mathcal{M} \leftarrow \mathcal{M} \cup \{(x, r_t)\} \triangleright Store reflection
13:
14:
            end for
```

By combining self reflection with model-based semantic decomposition, the agent benefits from both local reflective feedback and global structural guidance.

15: end for

Usage. Similarly, at the first round, M_p generates semantic units under a low temperature, and the temperature is increased in the remaining rounds to promote diversity.

3.3 Relation to Previous Studies

Global-local reflection Our design is inspired by Reflexion [28], which improves reasoning through iterative interaction between an actor M_a , an evaluator M_e , and a self-reflection module M_{sr} . The actor generates candidate outputs, the evaluator provides feedback based on task-specific signals, and the self-reflection module produces natural language feedback that is appended to the context of the next trial. In ParamAgent, we propose a new module M_r , which encodes cross-sample similarities into the model parameters, allowing the module to generate higher-quality reflections that facilitate the reasoning process of the language agent.

Semantic decomposition A key advantage of ParamAgent lies in its flexibility: the method can provide different forms of knowledge depending on the task. For multi-hop QA, we introduce semantic decomposition, which is conceptually related to CoT prompting. In standard CoT, intermediate thoughts z_i are elicited directly by the base LM. In contrast, our framework employs a dedicated parametric module M_p , which generates a structured set of intermediate thoughts Z that guide the reasoning process. Furthermore, we show that semantic decomposition can be seamlessly combined with reflection-based framework. Together they complement each other, yielding richer guidance to augment reasoning.

4 Experiments

In this section, we detail our experimental setup and present results across programming, math reasoning, and multi-hop QA. We then conduct more in-depth empirical analyses of our proposed method. More experimental results are included in Appendix D, including experiments with 70B scale LLMs.

4.1 Setup

172

Datasets We evaluate our framework across three domains. For programming, we use HumanEval [8] and MBPP [1]. For math reasoning, we adopt the MATH dataset [13], which covers competition-level problems of varying difficulty across seven subjects. For multi-hop QA, we use HotpotQA [41] and 2WikiMultiHopQA [14], which require reasoning across multiple passages. Further details about each dataset, as well as how we perform dataset splits are provided in later sections.

Evaluation For programming tasks, we follow prior work [28, 19] and report Pass@1. During generation, only visible or synthetic test cases are used, while final evaluation is conducted on hidden test cases; a score of 1 is assigned if all tests pass and 0 otherwise. For math reasoning and multi-hop QA, we report 0–1 accuracy on subsampled testsets.

Baselines We compare against: (1) **Base**, the underlying LLM agent without reflection; (2) **Re-**183 flexion [28], which uses episodic self-reflections; (3) **DoT** [19], which augments Reflexion with 185 prompt-level diversity. (4) **DoT-bank** [19], which further incorporate a memory bank to enrich the reflective feedbacks. (5) In addition, we develop a baseline that uses only the parametric module 186 to generate reflections or semantic units, referred to as model-based reflection or model-based CoT. 187 This baseline utilizes M_* purely as a parametric sampler, without performing self-reflection. The 188 pseudocode can be found in Appendix B. Our full model ParamAgent incorporates both episodic 189 memory and parametric memory for iterative reasoning, yielding stronger and more diverse feedback 190 signals. Finally, we also explore an extended variant ParamAgent-plus, where we further intro-191 duce a memory bank similar to DoT. This extension allows the agent to combine episodic memory, 192 parametric memory, and cross-sample memory, offering a comprehensive integration of different 193 knowledge sources. 194

To ensure a comprehensive evaluation, we employ three backbone LLMs with varying levels of reasoning capability: (1) **Llama-3.1-8B** [11], a strong open-source reasoning model; (2) **Mistral-197 7B-v0.2** [18], a competitive medium-sized model with efficient inference; and (3) **Qwen2-1.5B-instruct** [3], a TogetherAI's hosted version of Qwen2 1.5B, fine-tuned into an instruction-following variant. This selection of backbones allows us to examine how our approach performs across different model sizes and reasoning strengths. We also provide results with stronger base LLMs in Appendix D.4, showing that even when the parametric module remains an 8B model, it can still provide noticable gains to agents built on 70B-scale LLMs.

Hyperparameters Across all experiments, we fix the number of reflection iterations to 5 for both baseline methods and our proposed approach. For ParamAgent and its variants, we set the sampling temperature to T=0.2 during the first iteration, and T=1.0 in the subsequent iterations to promote diversity. For LoRA finetuning of the parametric modules, we use a rank of r=128, scaling factor $\alpha=32$, a learning rate of 2e-5, and train for 3 epochs.

Parametric module The parametric module is designed to internalize population-level knowledge across tasks $t \in \mathcal{D}$. We first construct a dataset $\{(t_i, m_i)\}_{i=1}^n$ by prompting an LLM on synthetic data or a subset of the training set to enumerate failure modes or semantic units, where n is typically around 10^4 . M_r or M_p is then obtained by finetuning a pretrained LLM using LoRA [16], encoding this knowledge into its parameters. In our experiments, we instantiate the module with Llama-3.1-8B. More details on the setup, dataset construction, training procedure, and implementation of the parametric module are deferred to Appendix D.2.

Table 1: Performance on HumanEval and MBPP datasets. **Bold** denotes the best result, and <u>underline</u> marks the second best. \uparrow and \downarrow indicate the absolute improvement or decrease relative to the Base method. For clarity, the prompt token usage of the Base method is normalized to 1. Table 2 and Table 3 use the same notation, which we omit from the captions due to space constraints.

Dataset	Method	Llama-3.1-8B Mistra		ral-7B-v0.2	Qv	ven2-1.5B	
		Pass@1	#Prompt Tokens	Pass@1	#Prompt Tokens	Pass@1	#Prompt Tokens
	Base	59.15	1.00	32.93	1.00	41.46	1.00
	Model-based Reflection	78.05 ↑ 18.90	9.15	68.29 † 35.36	23.73	68.91 ↑27.45	6.77
HumanEval	Reflexion	76.22 ↑ 17.07	9.29	51.22 ↑ 18.29	28.54	49.39 ↑ 7.93	18.30
Humanevai	DoT	73.17 ↑ 14.02	17.45	46.95 ↑ 14.02	43.06	56.56 ↑ 15.10	15.26
	DoT-bank	<u>79.56</u> ↑20.41	24.71	54.26 ↑21.33	61.62	60.10 18.64	31.28
	Ours	82.93 ↑23.78	19.18	<u>67.07</u> ↑34.14	70.38	<u>66.46</u> ↑25.00	33.45
	Base	47.61	1.00	24.94	1.00	42.06	1.00
	Model-based Reflection	52.90 ↑ 5.29	31.93	47.86 † 22.92	20.98	52.89 ↑ 10.83	25.35
MBPP	Reflexion	58.69 ↑11.08	37.18	28.46 ↑3.52	14.02	47.61 ↑5.55	26.95
МБРР	DoT	61.21 ↑ 13.60	51.83	19.79 \$_5.15	25.45	47.37 ↑5.31	21.48
	DoT-bank	<u>64.82</u> ↑17.21	69.41	24.68 \ \ 0.26	60.09	<u>53.38</u> ↑11.32	60.95
	Ours	67.00 ↑19.39	86.39	51.64 ↑ 26.70	36.88	54.90 ↑ 12.84	66.86

4.2 Programming

Datasets We evaluate our framework on two widely used programming benchmarks: HumanEval [8] and MBPP [1]. HumanEval consists of Python programming problems that test functional correctness using hidden unit tests, while MBPP covers beginner to intermediate-level Python problems designed for program synthesis.

Results From Table 1, we can observe that: (1) Model-based Reflection, which relies solely on parametric reflection without using self-reflection, already achieves substantial gains or performs comparably to the baseline methods across different LLM backbones. This indicates that parametric knowledge alone can provide useful reflective signals, effectively guiding the agent toward higher-quality solutions. (2) Furthermore, ParamAgent, which integrates both instance-specific self-reflections and model-based parametric reflections, achieves consistent improvements across most datasets and base models. This highlights the complementary benefits of combining feedback at different granularities: local signals capturing trial-specific errors and global signals capturing population-level patterns. (3) Notably, although DoT-bank also leverages global-level reflective feedback, it underperforms compared to ParamAgent in most scenarios. This highlights that retrieval-based memory modules are less effective than model-based parametric modules in capturing deep relational patterns across tasks. (4) Finally, ParamAgent is also cost-effective, its token consumption is on par or lower than DoT-bank while delivering stronger performance.

4.3 Math Reasoning

Datasets For mathematical reasoning, we evaluate on the MATH dataset [13], which consists of competition-level problems spanning seven subjects: Prealgebra, Algebra, Number Theory, Counting and Probability, Geometry, Intermediate Algebra, and Precalculus. To construct our evaluation set, we randomly sample 40 problems from each subject in testset, ensuring that the problems cover diverse topics and allows us to comprehensively assess the performance of different methods across varying topics.

Results From Table 2, we can make several observations. (1) Compared with model-based reflection alone, instance-level self-reflection proves more effective for math reasoning. However, when model-based reflective feedback is incorporated, ParamAgent consistently improves over Reflexion across all 3 backbone LLMs, and outperforms DoT on all LLM backbones. (2) Furthermore, ParamAgent-plus, which incorporates all levels of memory modules, achieves state-of-the-art results across all 3 backbone LLMs. This clearly demonstrates the added value of parametric knowledge: by combining episodic memory, parametric memory, and cross-sample memory, the agent gains access to richer and more complementary feedback signals, enabling stronger performance on competition-level mathematical reasoning tasks.

Table 2: Performance on MATH dataset.

Dataset	Method	Llama-3.1-8B		Mistral-7B-v0.2		Qwen2-1.5B	
		Acc	#Prompt Tokens	Acc	#Prompt Tokens	Acc	#Prompt Tokens
	Base	48.20	1.00	12.23	1.00	8.99	1.00
	Model-based Reflection	45.81 \ 2.39	2.58	13.31 ↑ 1.08	2.82	16.91 ↑7.92	2.84
	Reflexion	58.99 ↑10.79	23.33	19.78 ↑7.55	27.67	21.94 ↑ 12.95	18.39
MATH	DoT	64.38 ↑16.18	34.17	23.25 ↑11.02	40.51	22.30 ↑ 13.31	31.99
	DoT-bank	<u>73.02</u> ↑24.82	83.92	<u>35.61</u> ↑23.38	122.92	<u>24.37</u> ↑15.38	76.71
	ParamAgent	67.99 ↑19.79	57.01	28.06 ↑15.83	92.91	22.30 ↑ 13.31	70.07
	ParamAgent-plus	75.45 ↑27.25	111.32	38.96 ↑26.73	196.18	25.97 ↑16.98	144.25

Table 3: Performance on HotpotQA and 2WikiMultiHopQA datasets.

Dataset	Method	Llama-3.1-8B Mistral-7B-v0.2		Qwen2-1.5B			
		Acc	#Prompt Tokens	Acc	#Prompt Tokens	Acc	#Prompt Tokens
	Base	57.67	1.00	45.00	1.00	43.66	1.00
	Model-based CoT	61.67 ↑4.00	1.46	54.33 ↑9.33	1.46	48.10 ↑4.44	1.44
HotpotOA	Reflexion	71.33 ↑ 13.66	4.13	62.33 † 17.33	4.67	50.03 ↑6.37	6.22
HotpotQA	DoT	66.67 ↑9.00	7.10	58.33 ↑ 13.33	8.97	49.32 ↑5.66	58.05
	DoT-bank	<u>72.00</u> ↑ 14.33	13.28	<u>66.33</u> ↑21.33	19.35	<u>52.02</u> ↑8.36	109.54
	ParamAgent	78.33 ↑20.66	22.25	69.67 †24.67	34.99	64.66 ↑21.00	14.69
	Base	40.33	1.00	21.00	1.00	40.33	1.00
	Model-based CoT	54.67 ↑ 14.34	1.39	46.33 ↑25.33	1.21	40.66 ↑0.33	1.19
2WikiMultiHopQA	Reflexion	78.67 ↑38.34	5.47	61.33 ↑40.33	5.86	<u>51.00</u> ↑ 10.67	6.56
2 WikiMuluHopQA	DoT	66.67 ↑ 26.34	7.03	52.13 ↑ 31.13	6.40	47.83 ↑7.50	30.55
	DoT-bank	<u>80.33</u> ↑40.00	12.49	74.66 ↑53.66	8.10	50.49 ↑ 10.16	54.92
	ParamAgent	88.67 ↑48.34	10.41	81.33 ↑60.33	14.43	63.33 †23.00	17.39

4.4 Multi-hop QA

Datasets We use a subset of samples from HotpotQA and 2WikiMultiHopQA for the evaluation. Specifically, for HotpotQA, from the training set, we randomly sample 100 examples for each difficulty level (easy, medium, and hard), resulting in 300 test samples. For 2WikiMultiHopQA, we focus on 4 representative categories: bridge comparison, comparison, compositional, and inference. We randomly draw 75 examples from each category, leading to a total of 300 test samples.

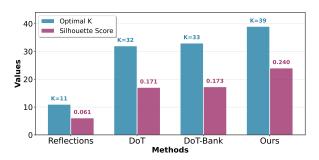
Results From Table 3, we can draw several conclusions. (1) Model-based CoT outperforms the Base method by a large margin across all three backbones. This indicates that decomposed semantics, when used as structured intermediate thoughts, facilitating the reasoning ability of the underlying LLM. (2) Although Model-based CoT underperforms compared to stronger baselines such as Reflexion and DoT-bank, ParamAgent achieves significant improvements over all state-of-the-art methods, even without incorporating a memory bank. This demonstrates the complementary advantages of semantic decomposition compared with reflective feedback, and highlights the effectiveness of combining them within a unified framework.

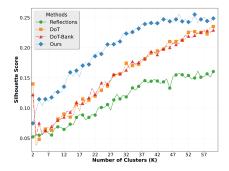
4.5 Additional Analysis

4.5.1 Quantifying the Diversity of Reflections

Setup We study to what extent the reflection trajectories produced by ParamAgent differ from those of other baselines. To this end, we maintain the complete reflection history for each sample and embed each reflection using the OpenAI text-embedding-3-small model. We focus on the HumanEval dataset for this analysis. Concretely, for N samples, and for each backbone LLM, we obtain a 2D tensor of shape $\mathbb{R}^{\sum_{i=1}^{N} n_i \times F}$, where n_i denotes the number of reflection iterations for sample i, and F is the embedding dimension. We then perform k-means clustering [21] over all reflections and apply the elbow method [30] to determine the optimal clustering number K^* . This provides a quantitative measure of the semantic diversity of reflective feedback across methods. Additionally, for each K, we compute the silhouette score to evaluate clustering quality.

Results As shown in Figure 2, ParamAgent achieves an optimal clustering number of $K^*=39$, substantially larger than that of Reflexion, DoT and DoT-bank. This indicates that the reflective outputs of Reflexion lack diversity, whereas the parametric feedback of ParamAgent introduces significantly richer and more varied reflective signals. Moreover, across all clustering numbers K, the silhouette score of ParamAgent is consistently higher than those of competing methods. This





(a) Optimal clustering number K^* across methods and the corresponding Silhouette scores.

279

280

281

282

283

284

285

286

287

288

289

290

292

293

294

295

296

297

298

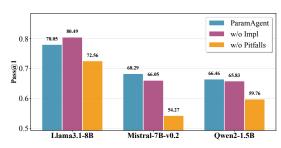
(b) Silhouette score for every clustering number $K \in [2, 60]$.

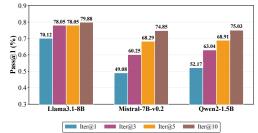
Figure 2: Quantitative analysis of reflection diversity: (a) optimal clustering number K^* ; (b) silhouette scores for different K.

suggests not only that ParamAgent generates more diverse reflections, but also that these reflections form more coherent semantic groups.

Effect of Reflection Format from M_r on Programming Tasks

In this section, we conduct an ablation study on the content of the reflections produced by M_r for programming tasks. As introduced in Section 3, for a given programming task t, M_r samples two complementary components: (1) textual reflective feedback describing potential pitfalls, and (2) code snippets implementing possible buggy solutions to illustrate failure modes. To analyze their contributions, we post-process the outputs from M_r by removing one of the components. We then evaluate the resulting variants on the HumanEval dataset.





- (a) Ablation study on the reflection format of M_r on (b) Ablation study on iteration numbers of Modelprogramming task.
- based Reflection on programming task.

Figure 3: Ablation studies.

Results From Figure 3a, we can observe that: (1) The textual reflective feedback (w/o Impl) proves to be more important across different backbone LLMs. When only buggy code implementations are preserved without natural language descriptions, the performance sometimes even degrades, as observed in Llama-3.1-8B. (2) When only textual reflective feedback is kept, both Mistral-7Bv0.2 and Qwen2-1.5B show moderate drops in accuracy. This suggests that including buggy code

implementations also provides useful complementary signals. Overall, these results indicate that outputting both natural language reflections and buggy code examples is a robust design choice.

4.5.3 Effect of Iteration Number on Model-based Reflection

In the main experiments, we fixed of iterations to 5 for fairness. Here we vary the iteration count of Model-based Reflection in {1, 3, 10} to examine how diversity affect the model performance.

Results As shown in Figure 3b, running only a single iteration yields subpar performance, indicating that providing one informative reflection (T=0.2) is insufficient. Performance improves steadily as the iteration number increases significantly, even without an episodic memory buffer. This highlights that the diversity introduced by the parametric memory is crucial and effective for performance gains.

302 5 Related Work

We discuss the most relevant work below, with additional related work in Appendix A.

LLMs have demonstrated emergent abilities to perform multi-step reasoning 304 when prompted appropriately. for instance, CoT prompting elicits the model to generate explicit 305 intermediate reasoning steps and significantly improves performance on complex tasks [37]. Self-306 Consistency [35] further improves CoT by sampling multiple reasoning paths and aggregating them 307 via majority voting, which increases robustness. ReAct [43] is a seminal approach that interleaves 308 reasoning steps with actions (e.g., tool uses or environment queries) in an interactive decision-making 309 loop, allowing the model to both "think" and "act" step-by-step. Other methods focus on iterative 310 self-feedback, highlighting that reasoning is a process, not a one-shot [22, 28]. ParamAgent follows 311 this iterative reasoning paradigm but avoids sophisticated search procedures. 312

External Memory in LLMs Memory has become central for agents tackling multi-step reasoning [47]. Short-term mechanisms such as Self-Refine [22] use the model's own recent outputs as 314 transient memory for iterative refinement, while Reflexion [28] maintains episodic logs of errors 315 and reflections to guide retries within a task. These approaches however, reset once a new problem 316 begins. To address this limitation, external memory has been proposed to augment agentic reason-317 ing [5, 27, 34, 27, 48, 36, 17, 40, 9, 19]. These methods mainly rely on non-parametric memory, 318 either through textual logs or retrieval databases. By contrast, ParamAgent introduces a external 319 parametric memory module M_* , which retrieves the knowledge from model-based sampler rather than recalling raw traces, enabling it to generate population-level knowledge that can be adapted to 321 322 different domains.

Cognitive Science Inspirations for Reasoning Cognitive science research shows that people manage complexity by chunking information into structured units, thereby reducing cognitive load and effectively increasing capacity for multi-step reasoning [23]. Classic working memory models further posit a central executive that maintains and manipulates only a small set of active items, motivating reasoning procedures that keep a compact buffer of intermediate results [2]. In parallel, problem solving is often modeled as hierarchical decomposition, where a complex goal is resolved by recursively addressing sub-goals and recombining their solutions [29]. These insights motivate semantic decomposition in multi-hop QA: parse a query into compact units (e.g., entities, relations, constraints, answer type) and an explicit sequence of inference steps; solve sub-queries sequentially while maintaining a small active buffer of intermediates; then integrate the chunks to produce the final answer.

334 6 Conclusions

323

325

326

327

328

329

330 331

332

341

We propose ParamAgent, a language agent framework that introduces a parametric module to move beyond instance-level reflection. By encoding cross-sample regularities into model parameters, ParamAgent internalizes global patterns and synthesizes population-level insights in a generative manner. Across programming, mathematical reasoning, and multi-hop QA, ParamAgent delivers substantial performance gains over state-of-the-art baselines while maintaining cost-effectiveness, highlighting the potential of parametric memory as a plug-in module for building language agents.

References

- Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv* preprint arXiv:2108.07732, 2021.
- 345 [2] Alan Baddeley. Working memory. Memory, pages 71–111, 2020.
- [3] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han,
 Fei Huang, et al. Qwen technical report. arXiv preprint arXiv:2309.16609, 2023.

- Yoshua Bengio, Salem Lahlou, Tristan Deleu, Edward J Hu, Mo Tiwari, and Emmanuel Bengio. Gflownet foundations. *Journal of Machine Learning Research*, 24(210):1–55, 2023.
- [5] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican,
 George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving
 language models by retrieving from trillions of tokens. In *International conference on machine learning*,
 pages 2206–2240. PMLR, 2022.
- [6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind
 Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners.
 Advances in neural information processing systems, 33:1877–1901, 2020.
- [7] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp
 Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte
 carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43,
 2012.
- [8] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan,
 Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models
 trained on code. arXiv preprint arXiv:2107.03374, 2021.
- 984 [9] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.
- [10] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts,
 Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language
 modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.
- 369 [11] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407, 2024.
- [12] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns,
 Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence
 with apps. NeurIPS, 2021.
- [13] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song,
 and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. arXiv preprint
 arXiv:2103.03874, 2021.
- 378 [14] Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa 379 dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060*, 2020.
- [15] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang,
 Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. Metagpt: Meta programming for a multi-agent
 collaborative framework. International Conference on Learning Representations, ICLR, 2024.
- Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.
- [17] Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. Hiagent: Hierarchical working memory management for solving long-horizon agent tasks with large language model. arXiv preprint arXiv:2408.09559, 2024.
- [18] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego
 de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud,
 Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and
 William El Sayed. Mistral 7b, 2023.
- 192 Vijay Lingam, Behrooz Omidvar-Tehrani, Sujay Sanghavi, Gaurav Gupta, Sayan Ghosh, Linbo Liu, Luke Huan, and Anoop Deoras. Enhancing language model agents using diversity of thoughts. 2025.
- [20] Zhengliang Liu, Yiwei Li, Peng Shu, Aoxiao Zhong, Longtao Yang, Chao Ju, Zihao Wu, Chong Ma,
 Jie Luo, Cheng Chen, et al. Radiology-llama2: Best-in-class large language model for radiology. arXiv
 preprint arXiv:2309.06419, 2023.
- [21] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

- Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon,
 Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback.
 Advances in Neural Information Processing Systems, 36:46534–46594, 2023.
- 402 [23] George A Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, 63(2):81, 1956.
- 404 [24] Ranjita Naik, Varun Chandrasekaran, Mert Yuksekgonul, Hamid Palangi, and Besmira Nushi. Diversity of thought improves reasoning abilities of large language models. 2023.
- 406 [25] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen,
 407 Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep
 408 learning library. Advances in neural information processing systems, 32, 2019.
- 409 [26] Yunfan Shao, Linyang Li, Junqi Dai, and Xipeng Qiu. Character-llm: A trainable agent for role-playing.
 410 *arXiv preprint arXiv:2310.10158*, 2023.
- 411 [27] Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Rich James, Mike Lewis, Luke Zettlemoyer, and
 412 Wen-tau Yih. Replug: Retrieval-augmented black-box language models. arXiv preprint arXiv:2301.12652,
 413 2023.
- [28] 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.
- 417 [29] Herbert A Simon. The architecture of complexity. In *The Roots of Logistics*, pages 335–361. Springer, 2012.
- 419 [30] Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a data set via 420 the gap statistic. *Journal of the royal statistical society: series b (statistical methodology)*, 63(2):411–423, 421 2001.
- Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix,
 Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation
 language models. arXiv preprint arXiv:2302.13971, 2023.
- 425 [32] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima
 426 Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint*427 *arXiv:2305.16291*, 2023.
- 428 [33] Haochun Wang, Chi Liu, Nuwa Xi, Zewen Qiang, Sendong Zhao, Bing Qin, and Ting Liu. Huatuo: Tuning llama model with chinese medical knowledge. *arXiv preprint arXiv:2304.06975*, 2023.
- 430 [34] Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. Augmenting
 431 language models with long-term memory. Advances in Neural Information Processing Systems, 36:74530–
 432 74543, 2023.
- 433 [35] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and
 434 Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint*435 *arXiv:2203.11171*, 2022.
- 436 [36] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv* preprint arXiv:2409.07429, 2024.
- 438 [37] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny
 439 Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. Advances in neural
 440 information processing systems, 35:24824–24837, 2022.
- [38] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang,
 Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation
 framework. arXiv preprint arXiv:2308.08155, 3(4), 2023.
- [39] Honglin Xiong, Sheng Wang, Yitao Zhu, Zihao Zhao, Yuxiao Liu, Linlin Huang, Qian Wang, and Dinggang
 Shen. Doctorglm: Fine-tuning your chinese doctor is not a herculean task. arXiv preprint arXiv:2304.01097,
 2023.
- [40] Wujiang Xu, Kai Mei, Hang Gao, Juntao Tan, Zujie Liang, and Yongfeng Zhang. A-mem: Agentic memory
 for llm agents. arXiv preprint arXiv:2502.12110, 2025.

- [41] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and
 Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. arXiv
 preprint arXiv:1809.09600, 2018.
- 452 [42] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan.
 453 Tree of thoughts: Deliberate problem solving with large language models. Advances in neural information
 454 processing systems, 36:11809–11822, 2023.
- [43] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. Re act: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.
- 458 [44] Fangxu Yu, Lai Jiang, Haoqiang Kang, Shibo Hao, and Lianhui Qin. Flow of reasoning: Training llms
 459 for divergent reasoning with minimal examples. In *Forty-second International Conference on Machine Learning*.
- [45] Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James
 Zou. Textgrad: Automatic" differentiation" via text. arXiv preprint arXiv:2406.07496, 2024.
- [46] Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin
 Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. arXiv preprint
 arXiv:2410.10762, 2024.
- 466 [47] Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and
 467 Ji-Rong Wen. A survey on the memory mechanism of large language model based agents, 2024.
- 468 [48] Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-exemplar prompting with memory for computer control. arXiv preprint arXiv:2306.07863, 2023.
- 470 [49] Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large
 471 language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*,
 472 volume 38, pages 19724–19731, 2024.
- 473 [50] Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent 474 tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 475 2023.

Appendix

Contents

476

490

491

492

493

495

496

497

498

499

500 501

502

503

504

505

506

507

508

509

510

511

513

514

515

516

517

478	A	Addi	Additional Related Work							
479	В	Pseu	docodes for ParamAgent	14						
480	C	Ratio	onale for External Module Fine-Tuning	14						
481	D	Mor	e Experimental Details Results	16						
482		D.1	Dataset Statistics	16						
483		D.2	Finetuning the Parametric Module	17						
484		D.3	More Implementation Details	17						
485		D.4	How does ParamAgent perform with stronger base LLMs?	17						
486		D.5	Cost Analysis	18						
487		D.6	A Case Study	19						
488		D.7	Prompt Templates	19						

489 A Additional Related Work

Diversity in LLM Reasoning A key challenge in multi-step reasoning is avoiding redundant or myopic thought patterns. Recent works therefore encourage diversity of reasoning paths. One simple but effective approach is self-consistency decoding [35], which samples multiple independent chains-of-thought and then selects the final answer by majority vote. Beyond this, researchers have proposed methods to actively inject diversity into the reasoning process. For example, prompting the model with different personas or perspectives (e.g. "Think like a mathematician" vs. "Explain as a teacher") can yield varied solution paths [24]. Another line of work trains LLMs to generate diverse solutions through specialized learning algorithms: Flow of Reasoning framework [44] the generation of reasoning steps as a search problem and uses a GFlowNet-based [4] fine-tuning to stochastically sample multiple high-reward reasoning trajectories, achieving greater coverage of the solution space. The recent DoT framework [19] explicitly tackles the lack of exploration by producing non-redundant self-reflections to ensure each iteration explores new solution paths, rather than repeating past failures. Empirically, DoT shows that encouraging such diversity yields substantial gains on challenging reasoning tasks. ParamAgent adopts the principle of diversification in a notably simple way: by drawing each new reasoning attempt from a high-temperature parametric sampler (the memory module). This high-temperature sampling from the learned model-based memory introduces stochasticity that is easy to implement yet effective at covering different problem-solving trajectories, without needing elaborate persona prompts or complex search procedure.

Parametric Memory in LLM Reasoning Compared with textual memory, parametric memory remains under-explored in LLM agents. While textual stores dominate due to interpretability and ease of use, recent studies have also explored encoding memory directly into model parameters, thereby avoiding the length limitations of textual memory. Character-LLM [26] fine-tunes role-playing agents with character experiences to faithfully simulate personas. HuaTuo [33] tunes LLaMA [31] with Chinese medical knowledge to enhance clinical QA and instruction following. DoctorGLM [39] develops a Chinese medical dialogue system, demonstrating that physician-style models can be obtained with moderate fine-tuning cost. Radiology-GPT [20] instruction-tunes on radiology corpora to outperform general LLMs on imaging-focused tasks. These approaches directly fine-tune the base LLM, yielding specialized models tailored to particular domains. By contrast, our framework keeps the base agent intact and fine-tunes an external parametric module that generates domain-specific reflective cues or semantic decompositions. This modular design allows the parametric module to

Algorithm 2 ParamAgent

```
Require: Dataset \mathcal{D}, Base LM p_{\theta}, Parametric Module M_* with parameters \psi, Max iterations T_{\text{max}},
     Pass@k K
Ensure: Solutions for each task
 1: Initialize: Episodic memory \mathcal{M} \leftarrow \emptyset
 2: for each task x \in \mathcal{D} do
 3:
         solved \leftarrow False, k \leftarrow 0
         while k < K and not solved do
 4:
 5:
              t \leftarrow 1, y_{\text{curr}} \leftarrow \text{None}
               while t \leq T_{\text{max}} and not solved do
 6:
 7:
                   # Generate parametric insights for iteration t
                   if t = 1 then
 8:
 9:
                        T \leftarrow 0.2

    informative first-round sampling

10:
                   else
11:
                        T \leftarrow 1.0
                                                                                         > promote diversity thereafter
12:
                   end if
13.
                   if task is coding/math then
14:
                        r_{t-1}^g \sim p_{\psi}(\cdot \mid x; T)
                                                                                           \triangleright global reflection from M_r
15:
                   else
                                                                                                          ⊳ multi-hop QA
                        Z \sim p_{\psi}(\cdot \mid x; T)
16:
                                                                                             \triangleright semantic units from M_p
17:
                   end if
18:
                   # Combine parametric and episodic knowledge
                   r_{1:t-1} \leftarrow \text{RETRIEVEREFLECTIONS}(\mathcal{M}, x)
19:
                                                                                   \triangleright local (self) reflections up to t-1
20:
                   if task is coding/math then
21:
                        y_{\text{curr}} \sim p_{\theta}(\cdot \mid x, r_{1:t-1}, r_{t-1}^g)

⊳ global–local fusion

                   else
22:
                        y_{\text{curr}} \sim p_{\theta}(\cdot \mid x, Z, r_{1:t-1})
23:
                                                                                             24:
25:
                   # Evaluate and update episodic memory
                   (passed, feedback) \leftarrow EVALUATE(y_{curr}, x)
26:
                   if passed then
27:
28:
                        solved \leftarrow True
                   else
29:
                        r_t \leftarrow \text{GENERATESELFREFLECTION}(y_{\text{curr}}, \text{feedback})
30:
                                                                                           31:
                        \mathcal{M} \leftarrow \mathcal{M} \cup \{(x, r_t)\}
32:
                   end if
33:
                   t \leftarrow t + 1
              end while
34:
35:
              k \leftarrow k + 1
         end while
36:
37: end for
```

serve as a plug-in component for different agents, while reducing the risk of catastrophic forgetting in the backbone. A detailed justification of this external design choice is provided in Appendix C. More details on parametric memory in agentic reasoning can be found in [47].

B Pseudocodes for ParamAgent

523

527

In this section, we present pseudocode for ParamAgent in Algorithm 2. We also include pseudocodes for Model-based Reflection and Model-based CoT in Algorithm 3 and ParamAgent-plus in Algorithm 4 for clarity.

C Rationale for External Module Fine-Tuning

We justify our choice to fine-tune an external parametric module rather than the base language agent.

Algorithm 3 Model-based Reflection (CoT)

Require: Dataset \mathcal{D} , Base LM p_{θ} , Parametric Module M_* with parameters ψ , Max iterations T_{max} , Pass@k K Ensure: Solutions for each task 1: **for** each task $x \in \mathcal{D}$ **do** solved \leftarrow False, $k \leftarrow 0$ 3: while k < K and not solved do 4: 5: while $t \leq T_{\text{max}}$ and not solved do # Parametric guidance only (no episodic memory) 6: $T \leftarrow \begin{cases} 0.2 & \text{if } t = 1 \\ 1.0 & \text{otherwise} \end{cases}$ 7: if task is coding/math then 8: $r_{t-1}^g \sim p_{\psi}(\cdot \mid x; T)$ $y_t \sim p_{\theta}(\cdot \mid x, r_{t-1}^g)$ else \triangleright global reflection from M_r 9: 10: 11: ⊳ multi-hop QA $Z \sim p_{\psi}(\cdot \mid x; T)$ $y_{t} \sim p_{\theta}(\cdot \mid x, Z)$ 12: \triangleright semantic units from M_p 13: 14: 15: # Evaluate (no memory write) 16: passed \leftarrow EVALUATE (y_t, x) 17: if passed then 18. $solved \leftarrow True$ end if 19. 20: $t \leftarrow t + 1$ end while 21: 22. $k \leftarrow k+1$ end while 23: 24: end for

Training objectives Given an input x, the external module is trained to produce either a reflection r or semantic units $Z = \{z_i\}_{i=1}^m$:

$$\max_{\substack{b \ p}} \ \mathbb{E}_{(x,r) \sim \mathcal{D}_r}[\log p_{\psi}(r \mid x)], \qquad \max_{\substack{b \ p}} \ \mathbb{E}_{(x,Z) \sim \mathcal{D}_Z}[\log p_{\psi}(Z \mid x)]. \tag{3}$$

At inference, the agent conditions on $r_k^g \sim p_\psi(\cdot \mid x)$ or on $Z \sim p_\psi(\cdot \mid x)$:

$$y \sim p_{\theta}(\cdot \mid x, r_{1:k}, r_k^g), \qquad y \sim p_{\theta}(\cdot \mid x, Z, r_{1:k}). \tag{4}$$

Why not fine-tune the agent directly? Directly fine-tuning the base LLM within the agent introduces the following challenges:

- (1) **Distribution mismatch.** In practice, an agent generates reflections autoregressively as $p_{\theta}(r_k \mid x, r_{1:k-1})$. If we fine-tune the base model only on $p_{\theta}(r \mid x)$ without its own history, the training distribution no longer matches the inference distribution $p_{\theta}(r_k \mid x, r_{1:k-1})$. Bridging this gap would require sequence-level supervision and far more data due to the more complex distribution form.
- (2) Capability interference. The agent must also maintain $p_{\theta}(y \mid x, r_{1:k})$ to act (e.g., generate code or multi-hop answers). Pushing the same parameters toward a specialized model for reflection generation can interfere with this objective, degrading the agent's general problem-solving ability.

Benefits of using external module In the meantime, adopt an external LLM module for parametric knowledge introduces several advantages:

- 543 **(1) Simpler supervision.** Decoupling the base LLM and the external LLM model yields a simpler objective $p_{\psi}(r \mid x)$ or $p_{\psi}(Z \mid x)$ rather than the history-conditioned $p_{\theta}(r_k \mid x, r_{1:k-1})$, reducing modeling complexity and data requirements.
- (2) Modular knowledge forms. The module can emit different forms of parametric knowledge (e.g., reflections r for programming/math via M_r , semantic units Z for multi-hop QA via M_p), complementing episodic self-reflection $r_{1:k}$ without altering the base agent.

Algorithm 4 ParamAgent-plus

```
Require: Dataset \mathcal{D}, Base LM p_{\theta}, Parametric Module M_*, Max iterations T_{\text{max}}
 1: Init: Episodic memory \mathcal{M} \leftarrow \emptyset, Memory bank \mathcal{B} \leftarrow \emptyset, Failed \mathcal{F} \leftarrow \emptyset
 2: Phase 1: Standard solving with memory banking
 3: for each task x \in \mathcal{D} do
              for t=1 to T_{\mathrm{max}}\, or until solved do
 4:
                     \begin{split} &r_{t-1}^g \sim p_{\psi}(\cdot \mid x) \text{ with } T \!=\! 0.2 \text{ if } t \!=\! 1 \text{ else } T \!=\! 1.0 \\ &r_{1:t-1} \leftarrow \text{Retrieve}(\mathcal{M}, x); \quad y \sim p_{\theta}(\cdot \mid x, r_{1:t-1}, r_{t-1}^g) \end{split}
 5:
                                                                                                                                                    ▶ Parametric insight
 6:
 7:
                      if Evaluate(y, x) passes then
                             Store (x, y, r_{t-1}^{\bar{g}}) in \mathcal{B}; mark solved; break
 8:
 9:
                            r_t \leftarrow \text{Reflect}(y); \quad \mathcal{M} \leftarrow \mathcal{M} \cup \{(x, r_t)\}
10:
                                                                                                                                                        ▶ Update episodic
                      end if
11:
              end for
12:
              if not solved then \mathcal{F} \leftarrow \mathcal{F} \cup \{x\}
13:
14:
              end if
15: end for
16: Phase 2: Memory-augmented reattempt
17: for each x \in \mathcal{F} do
              \mathcal{T} \leftarrow \mathsf{RetrieveTopK}(\mathcal{B}, x); \quad x_{\mathsf{aug}} \leftarrow \mathsf{Augment}(x, \mathcal{T})
18:
19:
              for t=1 to T_{\rm max} or until solved do
                     r_{t-1}^g \leftarrow \operatorname{Extract}(\mathcal{T}) \ \mathbf{or} \ p_{\psi}(\cdot \mid x_{\operatorname{aug}})
20:
                                                                                                        \triangleright Reuse or generate (same T rule as above)
                      \begin{array}{l} r_{1:t-1} \leftarrow \operatorname{Retrieve}(\mathcal{M}, x) + \operatorname{RetrieveByReflection}(\mathcal{B}) \\ y \sim p_{\theta}(\cdot \mid x_{\operatorname{aug}}, r_{1:t-1}, r_{t-1}^g); \quad \text{Evaluate and update } \mathcal{M}, \mathcal{B} \end{array} 
21:
22:
23:
24: end for
```

- (3) Stability and reuse. Keeping the base LLM in the agent fixed also avoids interference with $p_{\theta}(y \mid x, r_{1:k})$, mitigates catastrophic forgetting, and enables plug-in use across agents and backbones.
- In conclusion, fine-tuning an external module rather than the base LLM offers a simpler training objective, preserves the general capabilities of the agent, and enables flexible plug-in usage across domains and backbones, which justifies this design choice.

D More Experimental Details Results

555 D.1 Dataset Statistics

Programming. For programming tasks, we evaluate on HumanEval [8] and MBPP [1]. HumanEval consists of 164 hand-written Python programming problems, each accompanied by hidden unit tests and a small number of visible test cases. We additionally consider MBPP, which provides 974 crowd-sourced Python problems; following prior work, we use the 397 problems from the filtered evaluation split.

Math. For mathematical reasoning, we adopt the MATH dataset [13], which contains competitionstyle math problems spanning seven subjects including Algebra, Geometry, Number Theory, Counting and Probability, and Precalculus. We randomly sample a balanced subset across categories for evaluation.

Multi-hop QA. For multi-hop question answering, we use HotpotQA [41] and 2WikiMulti-HopQA [14]. In HotpotQA, we stratify by difficulty level and randomly sample 100 examples from each category (easy, medium, hard), yielding a total of 300 evaluation samples. For 2Wiki-MultiHopQA, we stratify by question type and randomly sample 75 examples from each of four categories (bridge comparison, comparison, compositional, inference), again yielding 300 samples in total. These stratified subsets ensure balanced evaluation across different reasoning styles.

Table 4: Datasets used for Programming, Math, and Multi-hop QA tasks.

Task Type	Dataset Name	Size	Metric
Programming Programming Math Multi-hop QA Multi-hop QA	HumanEval MBPP MATH HotpotQA 2WikiMultiHopQA	164 problems, ∼3 visible test cases/problem 397 sampled problems 278 sampled problems across 7 subjects 300 sampled problems (100 per difficulty) 300 sampled problems (75 per type)	Pass@1 Pass@1 0-1 Acc 0-1 Acc

D.2 Finetuning the Parametric Module

571

592

595

599

600

602

Programming For programming tasks, we curate a dataset by sampling 4000 coding problems from the APP dataset [12] at introductory level. In addition, we synthesize 4200 problems using GPT-40-mini, covering a diverse range of programming domains. The code templates and prompt used for data generation are provided in Figure 4. For each problem, GPT-40-mini is further asked to produce potential mistakes along with buggy implementations. This yields a dataset of reflective signals and corresponding erroneous code examples. We then finetune LLaMA-3.1-8B with LoRA on this dataset to obtain the programming-specific parametric module M_T .

Math For mathematical reasoning, we leverage the MATH training set [13]. From each subject area, we randomly sample 800 problems and adopt the same pipeline as in programming: GPT-40-mini is prompted to produce reflective feedback and buggy derivations for each sampled problem. The resulting dataset is used to LoRA-finetune LLaMA-3.1-8B to instantiate M_r for math reasoning.

Multi-hop QA For multi-hop QA, we randomly sample 10000 instances from the HotpotQA [41] and 2WikiMultiHopQA [14] training sets respectively. GPT-40-mini is prompted to output structured semantic units (e.g., entities, relations, constraints, answer types, and sub-questions) for each example. We then apply LoRA finetuning to LLaMA-3.1-8B on this dataset to build the parametric module M_p .

Across all domains, during dataset construction we provide one carefully designed demonstration example in the prompt to GPT-40-mini. This ensures that the generated outputs (reflective feedback, buggy code, or semantic units) adhere to the required format, making the synthetic supervision more reliable.

D.3 More Implementation Details

We use the TogetherAI API service¹ to access all backbone models in our experiments. Specifically, we call the following model identifiers in implementation:

- meta-llama/Meta-Llama-3.1-8B-Instruct-Turbo
- mistralai/Mistral-7B-Instruct-v0.2
- arize-ai/qwen-2-1.5b-instruct

In Section D.4, we use 70B scale LLMs in our framework, the model identifiers are:

- meta-llama/Meta-Llama-3.1-70B-Instruct-Turbo
 - Qwen/Qwen2.5-72B-Instruct-Turbo
- All experiments are implemented in PyTorch [25].

D.4 How does ParamAgent perform with stronger base LLMs?

We further study the performance of ParamAgent when paired with stronger base models of around 70B parameters. Specifically, we use Llama-3.1-70B and Qwen2.5-72B-Instruct as the underlying LLMs, while keeping the parametric module fixed as Llama-3.1-8B. We evaluate on HumanEval

https://www.together.ai

```
CATEGORIES = [
      # Core Text & Parsing
2
      "String Manipulation",
"Regular Expression Parsing",
3
4
      "Natural-Language Tokenisation",
5
      "CSV / JSON Parsing",
6
      "URL / URI Parsing",
      "Text Justification / Word-Wrapping",
9
      # Lists, Arrays, SEQ
      "Array / List Algorithms",
10
11
      "Two-Pointer / Sliding-Window",
12
      "Sorting & Searching",
      "Statistical Summary of Sequences",
13
      # Maths & Numbers
14
      "Elementary Arithmetic / Algebra",
15
      "Number Theory & Divisibility",
16
      "Bitwise Operations",
17
      "Combinatorics & Counting",
18
      "Probability / Statistics",
19
20
      # Data-Structures
      "Hash / Set / Dict Operations",
21
      "Stack / Queue Simulation",
22
      "Linked-List Manipulation",
23
      "Matrix Operations",
24
      "Heap / Priority Queue Operations",
      "Trie / Prefix-Tree",
26
      # Graphs & Trees
27
      "Graph / Tree Traversal",
28
      "Binary Search Trees",
29
      "Dynamic Programming",
30
      "Recursion / Backtracking"
31
      "Union-Find / Disjoint Set",
32
      # Geometry / Coordinates
33
      "Geometry & Coordinate Computation",
34
35
      # Dates / Times / Calendars
      "Date & Time Calculations",
36
      # Miscellaneous Practical
37
38
      "File & Path Utilities",
      "Data-Type Conversion & Formatting",
39
       "Cipher / Encoding",
40
      "Simulation / Game Logic",
41
      "Misc Small-Scale Algorithms"
42
43
```

Figure 4: Schema of categories for synthesizing programming tasks used in our parametric module construction.

for programming and HotpotQA for multi-hop QA. The results are reported in Table 5 and Table 6 respectively.

Results. Across tasks, ParamAgent achieves performance that is on par with, or even surpasses, state-of-the-art baselines. Moreover, ParamAgent-plus consistently outperforms the best baseline methods by a large margin, highlighting the effectiveness of the parametric module. It is worth noting that our parametric module itself is only an 8B model, yet it integrates effectively with base LLMs as large as 70B. This demonstrates the strong potential of our approach when scaled further.

D.5 Cost Analysis

608

609

610

611

612

613

Table 7 reports prompt/completion tokens and costs using Llama-3.1-8B. Costs are computed with TogetherAI pricing as of Aug 20, 2025 (\$0.18 per million tokens). We can see that Model-based Reflection (CoT) is highly efficient, achieving strong accuracy with far fewer tokens than reflection-heavy methods like DoT-bank. By contrast, ParamAgent delivers the best results on both HumanEval

```
system_content = (
    "You are an expert Python engineer crafting coding problems.\n"
    "Follow this EXACT format:\n\n<template_example>\n\n"
    "- Randomly pick ONE category from the list above.\n"
    "- Output EXACTLY two lines:\n"
    " func_sign: <signature with colon>\n"
    " docstring: '<single-quoted string with \\n escapes>'\n"
    "- Do NOT wrap in JSON or triple quotes.\n"
    "- Avoid any collisions with past tasks.\n\n"
)
```

Figure 5: Prompt for synthesizing programming tasks

Table 5: Performance on HumanEval. **Bold** denotes the best result, and <u>underline</u> marks the second best. \uparrow and \downarrow indicate absolute change relative to the Base method. For clarity, the prompt token usage of the Base method is normalized to 1.

Dataset	Method	Llama-3	.1-70B-Instruct	Qwen2.5-72B-Instruct		
		Pass@1	#Prompt Tokens	Pass@1	#Prompt Tokens	
	Base	80.49	1.00	82.92	1.00	
	Model-based Reflection	87.80 ↑7.31	6.39	89.64 ↑ 6.72	3.48	
	Reflexion	90.24 ↑9.75	4.31	88.41 ↑5.49	3.48	
HumanEval	DoT	90.85 \(\pm\)10.36	7.51	87.80 ↑4.88	6.05	
	DoT-bank	<u>92.68</u> ↑12.19	9.14	90.24 ↑7.32	8.17	
	ParamAgent	92.07 ↑11.58	11.90	<u>93.90</u> ↑ 10.98	8.93	
	ParamAgent-plus	95.03 ↑ 14.54	19.47	95.12 ↑ 12.20	16.81	

and HotpotQA, at higher but still moderate cost, this highlights the advantages of incorporating various forms of memory modules.

D.6 A Case Study

620

629

We present a case study from the MBPP dataset, where both Reflexion and DoT fail to generate the correct implementation, while ParamAgent succeeds. To better understand this difference, we analyze the reflective history of all three methods and highlight the gists, as illustrated in Figure 6.

From the analysis, we observe that Reflexion and DoT often produce unhelpful sometimes even misleading reflections, which push the agent further away from the correct solution. In contrast, ParamAgent generates fewer such misleading reflections. We hypothesize that this advantage arises from the parametric knowledge encoded in M_r , which helps ParamAgent avoid unhelpful or error-prone reflective signals.

D.7 Prompt Templates

We provide prompt templates used in ParamAgent across different domains. The 1-shot reflective example for programming tasks can be found in Figure 7, and the corresponding math reasoning template in Figure 8. For multi-hop QA, the semantic decomposition 1-shot example is shown in Figure 9.

Instruction templates for generating actions for the programming is shown in Figure 10, the math reasoning instruction in Figure 11, and the multi-hop QA instruction in Figure 12.

Table 6: Performance on HotpotQA dataset. **Bold** denotes the best result, and <u>underline</u> marks the second best. \uparrow and \downarrow indicate the absolute improvement or decrease relative to the Base method. For clarity, the prompt token usage of the Base method is normalized to 1.

Dataset	Method	Llama-3	1-70B-Instruct	Qwen2.5-72B-Instruct		
		Acc	#Prompt Tokens	Acc	#Prompt Tokens	
	Base	70.00	1.00	73.33	1.00	
	Model-based CoT	73.67 ↑3.67	1.43	74.10 ↑ 1.05	1.44	
	Reflexion	82.33 12.33	3.02	<u>82.67</u> ↑9.34	2.81	
HotpotQA	DoT	73.67 ↑3.67	3.43	80.67 ↑7.34	4.30	
	DoT-bank	80.00 \(\gamma\) 10.00	5.24	82.33 ↑9.00	7.87	
	ParamAgent	<u>84.00</u> ↑ 14.00	7.70	81.00 ↑ 7.67	7.90	
	ParamAgent-plus	89.67 ↑ 19.67	13.69	84.67 ↑ 11.34	15.43	

Table 7: Token usage and cost on HumanEval and HotpotQA datasets with Llama3.1-8B as backbone LLM. Best and second-best metrics are in **bold** and <u>underline</u> respectively.

Method	HumanEval				HotpotQA			
	#Prompt	#Prompt #Completion Total Cost Pass@1 #			#Prompt	#Completion	Total Cost	Acc
	Tokens	Tokens	(\$)	(%)	Tokens	Tokens	(\$)	(%)
Base	37,463	13,506	0.00917	59.15	164,013	1,801	0.02985	57.67
Model-based Reflection	342,805	82,280	0.07652	78.05	236,548	1,212	0.04280	61.67
Reflexion	348,068	73,538	0.07589	76.22	703,192	68,612	0.13892	71.33
DoT	653,981	169,986	0.14831	72.56	1,164,812	106,806	0.22889	66.67
DoT-bank	926,047	233,016	0.20863	79.88	2,179,148	195,283	0.42740	72.00
ParamAgent	814,627	163,257	0.17602	82.93	3,649,598	128,010	0.67997	78.33

Reflexion.

Problem: "The implementation failed the test case where it tries to return a single element that is not a list. This issue arises because the ^ operator in the set difference operation returns a set, and then we explicitly convert it to a list..."

Problem: "The implementation failed the test where it is expected to find a difference in the lists because the list to check for difference is a subset of the original list... This would return the union of the two lists, not the actual difference."

Problem: "The implementation failed the test where the difference should be a single element because it uses the symmetric difference operation which is not what's intended here... When converting the set back to a list using the list function, the order of elements is lost."

Problem: "When converting the set back to a list using the list function, the order of elements is lost."

DoT.

Problem: "The implementation failed the test due to a mismatch in the type of the function return value and the expected output."

Problem: "The function implementation does not handle the case where the input lists are not of the same length or contain duplicate elements."

Problem: "The function implementation does not correctly calculate the symmetric difference between two lists."

Problem: "The implementation failed due to a mismatch in the type of the function return value and the expected output, but the proposed fix in the previous reflection is incomplete."

Problem: "The function implementation does not correctly handle the case where the input lists are not of the same length or contain duplicate elements..."

Problem: "The function implementation does not correctly remove duplicates from the input lists, leading to incorrect results."

ParamAgent.

Problem: "The function implementation does not handle the case of empty lists correctly." **Fix:** "Add a check at the beginning of the function to return an empty list if either of the input lists is empty..."

Problem: "The function implementation does not preserve the original order of elements in the lists." **Fix:** "Instead of using set operations..., use a list comprehension to create the difference list."

Problem: "The function implementation does not handle non-integer values in the lists correctly." **Fix:** "Add a check at the beginning... ensure that both input lists are lists of integers..."

Problem: "The function implementation does not handle non-list types correctly." **Fix:** "Add a check... ensure that both input arguments are lists..."

Problem: "The function implementation does not handle duplicates correctly." **Fix:** "Use a list comprehension... and preserve the original order..."

Problem: "The function implementation does not handle the case where the lists are identical." **Fix:** "Return an empty list if the two input lists are identical."

Figure 6: Selected reflections on the mbpp_769_Diff problem. **Red** = critical errors that directly invalidate the implementation; **Orange** = conceptual pitfalls or misleading reasoning patterns; **Green** = key insights leading to a working solution; **Gray** = background noise (irrelevant or stylistic).

```
[Function Signature]:
  def has_close_elements(numbers: List[float], threshold: float) -> bool
      """Check if any two numbers in the list are closer than the
          threshold."""
5 [Potential mistakes]:
6 1. **Empty or Single-Element Lists** must return 'False', not 'True'.
7 2. **Duplicate Values** must be compared (difference 0), so never drop
       duplicates.
  3. Always use **absolute difference** ('abs(a - b)'), not raw
      subtraction.
  4. Use the correct **strictness** ('< threshold', not '<=').
10 5. Ensure you dont **exit too early** check
                                                    all distinct pairs.
[Flawed Implementations Illustrating Each Pitfall]:
13
14 def has_close_elements_v1(numbers: List[float], threshold: float) ->
      bool:
      # BUG: returns True for empty or single-element lists
15
      if len(numbers) < 2:</pre>
16
          return True
17
      for i in range(len(numbers)-1):
18
19
           for j in range(i+1, len(numbers)):
               if abs(numbers[i] - numbers[j]) < threshold:</pre>
20
                   return True
21
22
      return False
23
  def has_close_elements_v2(numbers: List[float], threshold: float) ->
      # BUG: removes duplicates, so identical values never compared
25
      numbers = sorted(set(numbers))
26
      for i in range(len(numbers)-1):
          if abs(numbers[i+1] - numbers[i]) < threshold:</pre>
28
               return True
29
      return False
30
31
  def has_close_elements_v3(numbers: List[float], threshold: float) ->
      bool:
      # BUG: uses raw subtraction instead of abs()
33
      for i in range(len(numbers)-1):
34
35
          for j in range(i+1, len(numbers)):
               if (numbers[i] - numbers[j]) < threshold:</pre>
36
                   return True
37
38
      return False
39
  def has_close_elements_v4(numbers: List[float], threshold: float) ->
40
      bool:
      # BUG: uses <= instead of <, misclassifies exactly-threshold pairs
41
      for i in range(len(numbers)-1):
42
           for j in range(i+1, len(numbers)):
43
               if abs(numbers[i] - numbers[j]) <= threshold:</pre>
44
45
                   return True
      return False
46
47
  def has_close_elements_v5(numbers: List[float], threshold: float) ->
48
      bool .
      # BUG: breaks out of outer loop too soon
49
      ... (omit due to limited page)
50
52 END OF EXAMPLE
```

Figure 7: 1-shot example for reflective dataset construction for programming task.

Question. Circle O is located on the coordinate plane with center at (2,3). One endpoint of a diameter is at (-1,-1). What are the coordinates of the other endpoint of this diameter? Express your answer as an ordered pair.

Pitfalls & Potential Mistakes

- 1. **Confusing the center with an endpoint.** Assuming the center is an endpoint leads to an incorrect reflection point.
- 2. Incorrect use of the midpoint formula. Forgetting that the center is the midpoint of the diameter, or solving $(x + x_2)/2 = \text{center}_x$ incorrectly.
- 3. **Using the wrong coordinates for the midpoint.** Plugging endpoint coordinates in place of the center (or vice versa) yields the wrong unknowns.
- 4. Arithmetic errors. Sign or algebra mistakes when solving, e.g. $2 = (-1 + x)/2 \Rightarrow x = 3$ (incorrect) instead of x = 5.
- 5. **Switching** x **and** y. Mixing x- and y-midpoint formulas, or using x values to solve for y.
- 6. **Incorrect interpretation of the diameter.** Thinking the diameter extends in the same direction from the center; doubling the vector or reflecting in the wrong direction.

Figure 8: 1-shot example for reflective dataset construction in math reasoning.

Example 1

Question. Anatoly Maltsev and Valentin Turchin were both from Russia, which of the two is known for his work as a mathematician?

Question Parsing and Intent Extraction

Key Components

- Entity A: Anatoly Maltsev mathematician/logician; contributions in mathematical logic and abstract algebra.
- Entity B: Valentin Turchin computer scientist/philosopher; work in cybernetics and philosophy of science.
- Implied Relationship: Comparative inquiry: which individual is more closely associated with mathematics.
- Answer Type Expected: Person name (e.g., "Anatoly Maltsev").
- **Reasoning Type:** Comparative factual reasoning.
- Required Background: Biographical profiles or retrieved professional records.

Inference Trace

- 1. Retrieve factual data about Maltsev's and Turchin's primary academic domains.
- 2. Classify Maltsev as a mathematician (core contributions to mathematical logic).
- 3. Classify Turchin as mainly in cybernetics and philosophy.
- 4. Eliminate Turchin as the primary mathematician.
- 5. Conclude: Anatoly Maltsev.

Disambiguation Note

Nationality (Russia) does not help differentiate them.

Figure 9: 1-shot example used in ParamAgent for semantic decomposition dataset construction in multi-hop QA.

You are an AI Python assistant. You will be given some potential pitfalls and several flawed implementations for the coding challenge, as well as your previous implementation of a function, a series of unit-test results, and your self-reflection on your previous implementation. Try to avoid the errors from your previous implementation and the listed pitfalls.

Instruction: ALWAYS WRITE your full implementation (restate the function signature).

Figure 10: Instruction prompt used by ParamAgent to generate next-round solutions for programming tasks.

You are revising your previous answer to a mathematics problem.

You will receive:

- (1) the original question,
- (2) potential mistakes and pitfalls,
- (3) your last answer, (4) feedback (Right or Wrong) explaining why that answer was unsatisfactory, and (5) your brief self-reflection on the mistake.

Respond with:

- 1. **Reasoning**: updated step-by-step thoughts.
- 2. **Answer**: the corrected final result.

Formatting: The final answer should be simplified to its simplest form, e.g., 25, 25_{16} , $\frac{1}{36}$, etc.

Figure 11: Instruction prompt used by ParamAgent to generate next-round solutions for math reasoning.

You are revising your previous answer to a multi-hop QA question.

You will receive:

- (1) the original question,
- (2) some key points, the underlying intent, and possible inference patterns that facilitate answering this question,
- (3) your last answer,
- (4) supporting context,
- (5) feedback (Right or Wrong) explaining why that answer was unsatisfactory,
- (6) your brief self-reflection on the mistake.

Instruction: Based on the inputs, produce a new single-phrase answer that resolves the error and fully answers the question. Output only the answer — no commentary, no code.

Figure 12: The prompt of ParamAgent to generate next-round answers for multi-hop QA tasks.