# From Frontier to Frugal: Evaluating Self-Evolution Frameworks with Small Language Models

**Shayan A. Akbar, Jiaming Qu, Chen Ling, Madhu Gopinathan, Erwin Cornejo**
Amazon
shayaakb@amazon.com

## Abstract

Optimizing Large Language Models (LLMs) for production AI agent deployment demands substantial computational resources and specialized human expertise (e.g., prompt engineering). Self-evolution offers a promising solution by enabling agents to autonomously enhance capabilities through structured feedback, improving performance without expensive manual optimization. However, most existing self-evolving agents rely on costly frontier LLMs, limiting scalability. Can small language models (SLMs) achieve effective self-evolution while remaining cost-efficient? We investigate this by applying two SLMs (Qwen and Claude Haiku) to self-evolution frameworks (Gödel Agent and GEPA) on MGSM mathematical reasoning, MMLU knowledge reasoning, and ARC science questions benchmarks. Our experiments demonstrate that SLMs equipped with feedback loops successfully evolve by rewriting code, optimizing parameters, and refining prompt strategies. Remarkably, SLMs achieve accuracy gains up to 15.5% with evolution costs as low as $0.04 per optimization cycle, enabling scalable production deployments. While gains are more modest than frontier LLMs (specifically on certain benchmarks like MGSM), our results demonstrate that self-improving SLMs offer a viable, cost-effective path toward scalable autonomous AI systems.

## 1 Introduction

Large Language Models (LLMs) have enabled a new generation of AI agents capable of solving complex real-world tasks, from mathematical reasoning and code generation to scientific research and multi-step problem solving (OpenAI, 2025; Anthropic, 2025). However, the costs of deploying LLM-based agents, along with the manual effort required to design and optimize each agent, pose significant challenges in scaling these systems.

To tackle this, a promising research direction is developing self-evolving agents that autonomously improve their own capabilities through iterative self-modification, including tool creation, prompt strategy changes, and adaptive learning without human intervention. Self-evolving agents reduce human effort in designing efficient agentic AI systems while enabling agents to explore design spaces that humans may overlook. This paradigm represents a step toward more adaptive and autonomous AI systems (Gao et al., 2025).

Recent work has demonstrated the feasibility of self-evolution in LLM-powered agents. Systems such as Godel Agent (Yin et al., 2025), Darwin Gödel Machine (DGM) (Zhang et al., 2025a), and Alita (Qiu et al., 2025) have shown that agents can autonomously enhance their performance on benchmarks like MMLU (Hendrycks et al., 2020), MGSM (Shi et al., 2022), and SWE-bench (Jimenez et al., 2023), through mechanisms such as tool synthesis, dynamic code generation, and self-supervised data augmentation. These systems observe feedback from the environment and refine their actions with self-reflection.

However, these approaches largely rely on frontier proprietary LLMs such as GPT and Claude series — models having hundreds of billions of parameters — resulting in substantial computational costs, energy consumption, and deployment barriers with limited accessibility. More critically, these cost constraints become exponentially worse in multi-agent settings, where deploying even a modest number of self-evolving LLM agents becomes economically infeasible. This scalability bottleneck limits the potential of self-evolving multi-agent systems in real-world applications.

In contrast, Small Language Models (SLMs), typically defined as models with fewer than 10B parameters, offer a compelling alternative. SLMs can be deployed on consumer-grade hardware, require less energy, and enable local execution with reduced latency and privacy concerns, while showing competitive performance on specific tasks. De-

spite these advantages, the capacity of SLMs to drive self-evolving agents remains underexplored, which raises a critical question: *Can SLMs guide self-improvement processes? And, to what extent?*

To answer this question, we draw attention to the underexplored potential of SLM-powered self-evolving agents. We adapt established self-evolution frameworks, Gödel Agent (Yin et al., 2024) and GEPA (Agrawal et al., 2025), to operate with SLMs including Qwen3-4B (Yang et al., 2025) and Claude Haiku (Anthropic, 2024), evaluating on MGSM, MMLU, and ARC benchmarks. Our empirical results demonstrate that SLMs can drive self-evolving agents through autonomous tool updates, prompt evolution, etc., with Qwen3-4B improving by **3.3%**, Claude Haiku 3 by **15.5%** (evaluated over 3 independent runs: 74.75% $\pm$ 3.85%), and Claude Haiku 4.5 by **1.3%**. Compared to frontier LLMs (GPT-4 achieves $\sim$5% improvement on MMLU and $\sim$36% on MGSM with Gödel Agent), SLM gains are more modest but demonstrate that self-evolving capabilities are not exclusive to frontier-scale models, opening avenues for accessible and cost-effective autonomous agent research.

## 2 Related Work

**Evolution in AI Agents:** Evolving AI agents represent a paradigm shift from static models to adaptive systems capable of autonomous improvement and recursive self-modification (Yin et al., 2024; Zhang et al., 2025b; Qiu et al., 2025; Gao et al., 2025).

Gödel Agent (Yin et al., 2024) introduces self-improvement through recursive self-modification, where LLMs dynamically modify their own logic and behavior. Unlike traditional meta-learning where only policy evolves, Gödel Agent enables both policy and meta-learning algorithm to update together through self-referential learning, achieving improvements via dynamic code modification.

Several methods have explored automated prompt engineering. APE (Zhou et al., 2022) generates and selects instructions automatically using LLMs as inference models. OPRO (Yang et al., 2023) formulates prompt optimization as an optimization problem where LLMs iteratively generate improved prompts guided by past performance. EvoPrompt (Guo et al., 2023) applies evolutionary algorithms (genetic algorithms and differential evolution) to discrete prompt optimization, combining LLM-based variation operators with population-based search. These methods share the goal of reducing manual prompt engineering effort, but none explicitly investigate whether SLMs can serve as both optimizer and executor in a self-evolving loop.

GEPA (Agrawal et al., 2025) introduces fine-grained prompt optimization through Pareto-aware evolution. Unlike existing optimizers that optimize for aggregate performance (Yuksekgonul et al., 2025; Opsahl-Ong et al., 2024), GEPA maintains a Pareto front of prompts where each performs optimally on different instance subsets, enabling systematic exploration of diverse prompt strategies.

We utilize Gödel Agent and GEPA frameworks to analyze SLM potential in self-evolution, as they represent practical implementations of self-referential agents and prompt evolution.

**Small Language Models:** Recent SLMs with fewer than 10 billion parameters have demonstrated competitive performance with computational efficiency, including Phi-4 (Abdin et al., 2024), SmolLM (Allal et al., 2025), Gemma (Team et al., 2025), and Qwen (Yang et al., 2025). We use Qwen and Claude Haiku models in our experiments.

While self-evolution and SLMs have achieved individual success, current self-evolution frameworks largely rely on LLMs. Therefore, we investigate whether SLMs can effectively drive self-evolution processes, measuring both the extent of improvement and the associated cost.

## 3 Agent Evolution Methodologies

We review two agent evolution approaches used in our study: Gödel Agent and GEPA.

### 3.1 Gödel Agent: Self-Referential Framework

Gödel Agent uses LLMs to dynamically modify their own logic and behavior based on high-level instructions, eliminating human design priors allowing agent to freely decide its own routine, modules, and update mechanisms.

The core innovation is *recursive self-improvement*: both policy $\pi$ and meta-learning algorithm $I$ update simultaneously through $\pi_{t+1}, I_{t+1} = I_t(\pi_t, I_t, r_t, g)$, where $r_t = U(E, \pi_t)$ is the utility function $U$ applied on environment state $E$, and $g$ represents the optimization goal. This self-referential learning enables the agent to rewrite its entire codebase after each update, distinguishing it from traditional meta-learning where only the policy evolves. For details, we refer readers to the original paper (Yin et al., 2024).

The agent achieves *self-awareness* through runtime memory inspection of Python variables, functions, and classes. *Self-improvement* occurs via dynamic code modification, where the agent reasons about necessary changes and writes new code directly into runtime memory, recursively refining its logic each iteration.

To prevent SLMs from prematurely halting evolution when performance appears satisfactory, we augmented the Gödel Agent goal prompt with the following instruction:

> CRITICAL EVOLUTION MANDATE: You MUST continue evolving in EVERY cycle until you reach maximum iterations. Even if performance seems satisfactory, you MUST find new ways to improve: explore alternative algorithms, add robustness features, optimize efficiency, enhance error handling, implement new validation methods, or experiment with different approaches. NEVER conclude that no improvements are needed - there are ALWAYS optimizations to discover.

### 3.2 GEPA: Genetic-Pareto Prompt Optimizer

GEPA is a sample-efficient optimizer for evolving SLMs through reflective prompt optimization using a teacher-student framework, where a teacher LLM guides prompt evolution to improve student SLM performance. Unlike traditional approaches relying on scalar rewards, GEPA leverages complete execution traces including reasoning chains, tool calls, and evaluation feedback to enable improvements.

The key insight is maintaining a Pareto front tracking which prompt strategies work best for different problem types at the instance level, rather than treating the training set as monolithic. During each iteration, GEPA evaluates candidate prompts on training minibatches, collecting execution traces. The teacher reflects on these traces to diagnose failures and propose prompt refinements. Improved prompts are evaluated on the validation set and added to tracked candidates. GEPA stochastically samples from candidates achieving best scores on at least one validation instance, preserving diverse strategies during evolution. This enables efficient exploration while preventing premature convergence, requiring only small rollouts since each trace provides rich feedback.

### 3.3 GEPA vs. Gödel Agent for Real-World Production Implementation

GEPA and Gödel Agent represent different optimization paradigms: GEPA operates within constrained boundaries by only modifying textual prompts through natural language, while Gödel Agent has almost unrestricted access to modify any system component through arbitrary code execution. GEPA is immediately deployable in production due to its predictable nature, whereas Gödel Agent requires extensive oversight to ensure safe autonomous modifications.

## 4 Experimental Design and Results

### 4.1 Experimental Design and Setup

**Benchmark Datasets:** We evaluate on three benchmarks: (1) *MMLU* (Hendrycks et al., 2020), which assesses knowledge and reasoning capabilities across 57 subjects ranging from elementary to professional difficulty; (2) *MGSM* (Shi et al., 2022), a multilingual mathematical reasoning dataset spanning 10 languages that measures reasoning capabilities across different languages; and (3) *ARC* (Clark et al., 2018), a dataset containing grade-school level, multiple-choice science questions.
**Small Language Models:** We use three SLMs: *Qwen3-4B-Instruct-2507*, a 4B parameter model with strong instruction following and reasoning capabilities supporting up to 260K context length; *Claude Haiku 3*, a fast cost-efficient model processing approximately 21K tokens per second; and *Claude Haiku 4.5*, the latest small Claude model delivering coding performance similar to Claude Sonnet 4 at one-third the cost but twice the speed.
**Implementation Details:** Qwen3-4B was deployed on 8 NVIDIA A10G GPUs (192GB total memory) with Python multiprocessing for parallel inference. Claude Haiku models were accessed via Amazon Bedrock API using boto3. We used the DSPy framework for GEPA and the original Gödel Agent codebase. Benchmark datasets were obtained from the Gödel Agent repository.

### 4.2 Experimental Results

Tables 1 and 2 show the main results of our study evaluating SLM-driven self-improvement using Gödel Agent and GEPA frameworks.

#### 4.2.1 Gödel Agent Results

Applying Gödel Agent's recursive self-improvement resulted in measurable accuracy

Table 1: Gödel Agent performance. SLM (Qwen3-4B) improvements compared with frontier LLM results from Yin et al. (2024). GPT-4o serves as the optimizer with GPT-3.5 as executor in the original work. While SLM gains are more modest, they demonstrate viable self-evolution at significantly lower cost. Note: the original work uses GPT-4o as optimizer and GPT-3.5 as executor, whereas our SLM setup uses Qwen3-4B for both roles. The higher SLM baselines partially explain smaller relative gains due to reduced headroom.

| **Model** | **MMLU** | **+%** | **MGSM** | **+%** |
|---|---|---|---|---|
| *Frontier LLM (Yin et al., 2024)* | | | | |
| GPT-3.5 (CoT) | 0.65 | – | 0.28 | – |
| GPT-3.5+Gödel (GPT-4o opt.) | 0.70 | +5 | 0.64 | +36 |
| *SLM (Ours)* | | | | |
| Qwen3-4B | 0.77 | – | 0.75 | – |
| Qwen3-4B+Gödel | 0.79 | +3.3 | 0.76 | +1.7 |

gains for Qwen3-4B across both benchmarks. As shown in Table 1, accuracy increases from 0.771 to 0.797 on MMLU (+3.3%) and from 0.753 to 0.766 on MGSM (+1.7%). The more substantial gain on MMLU suggests that Gödel Agent's self-referential learning is particularly effective for knowledge-intensive reasoning tasks. Qualitative analysis of the evolved solver code reveals that the SLM autonomously discovered software engineering best practices including defensive programming, regex-based answer extraction, confidence scoring, and parameter optimization (see Appendix A).

The evolution process revealed interesting patterns in how SLMs approach self-modification. Over multiple iterations, Qwen3-4B progressively identified failure modes through runtime memory inspection and implemented targeted fixes. Early iterations focused on basic error handling and response parsing, while later iterations introduced more sophisticated mechanisms such as confidence-based fallbacks and multi-pattern answer extraction. Notably, the model demonstrated ability to reason about its own limitations and proactively add defensive code to handle edge cases it had not yet encountered, suggesting genuine self-awareness capabilities even in smaller models.

#### 4.2.2 GEPA Results

GEPA's reflective prompt evolution yielded significant gains, particularly for Haiku 3. For GEPA experiments, we used the same SLM as both teacher and student, relying solely on SLM-driven evolution, though GEPA's framework supports using a more capable LLM as the teacher. As shown in Table 2, Haiku 3 demonstrates substantial improvement on all benchmarks: accuracy increases from 0.70 to 0.73 on MMLU (+3.3%), from 0.82 to 0.86 on ARC (+4%), and most notably from 0.67 to 0.77 on MGSM (+15.5%). In contrast, Haiku 4.5 exhibits different behavior. Starting from higher baselines (0.883 MMLU, 0.93 MGSM), it shows only marginal improvements of +0.1% and +1.3% respectively, while ARC performance remains flat. This pattern suggests that models with stronger initial performance have less room for improvement through prompt evolution.

**Multi-Run Validation.** To confirm robustness, we conducted 3 independent trials of GEPA with Claude Haiku 3 on MGSM (our headline result). Across these runs, we observe a test accuracy of $74.75\% \pm 3.85\%$ (individual runs: 78.0%, 70.5%, 75.75%), all showing substantial improvement over the baseline of 67%. This confirms that the gains are consistent and not an artifact of a single favorable run. The variance across runs reflects the stochastic nature of prompt evolution (different initial samples, mutation paths), but all runs converge to meaningful improvement.

The disparity between Haiku 3 and 4.5 suggests smaller models have greater headroom for enhancement through prompt evolution. Haiku 3 benefits substantially from GEPA's injected reasoning support, while Haiku 4.5 has already internalized effective strategies, yielding diminishing returns. This implies prompt optimization can significantly enhance weaker SLMs, while models approaching task-optimality naturally yield smaller improvements. Qualitative analysis shows GEPA transforms generic prompts into domain-specific, task-aware instructions (see Appendix B).

The dramatic 15.5% improvement for Haiku 3 on MGSM warrants further examination. Analysis of the evolved prompts reveals that GEPA identified multilingual handling as a critical weakness and systematically injected language-aware reasoning strategies. The evolved prompts explicitly guide the model to handle non-English mathemat-

Table 2: GEPA performance driven by SLM. Using GEPA framework Haiku SLMs improve their accuracy on MMLU, MGSM, and ARC benchmarks (e.g., 15% improvement on MGSM).

| Model | MMLU | MGSM | ARC |
|---|---|---|---|
| Haiku3 | 0.70 | 0.67 | 0.82 |
| +GEPA | 0.73 (+3.3%) | 0.77 (+15%) | 0.86 (+4%) |
| Haiku4.5 | 0.883 | 0.93 | 0.938 |
| +GEPA | 0.884 (+0.1%) | 0.942 (+1.3%) | 0.937 (–) |

Table 3: Cost breakdown for GEPA + Haiku 3 on MGSM evolution.

| Metric | Value |
|---|---|
| Prompt tokens consumed | 80,484 |
| Completion tokens consumed | 16,442 |
| Total tokens | 96,926 |
| Total evolution cost | $0.04 |
| Baseline prompt length | 13 words |
| Evolved prompt length | 467 words |
| Accuracy gain | +15.5% |

ical problems by first understanding the problem semantics before applying arithmetic operations. Additionally, GEPA introduced arithmetic verification steps and common pitfall warnings that proved particularly effective for the math reasoning task. These findings suggest that GEPA's instance-level Pareto tracking successfully identified diverse failure modes and generated targeted prompt improvements.

### 4.3 Cost Analysis

A key motivation for SLM-driven self-evolution is cost efficiency. We provide a detailed quantitative breakdown of evolution costs.

**Evolution Phase Cost.** For our headline configuration — GEPA with Claude Haiku 3 on MGSM — the entire evolution process consumed approximately 97K tokens (80,484 prompt tokens and 16,442 completion tokens), totaling **$0.04** based on Haiku 3's pricing ($0.25/MTok input, $1.25/MTok output). This represents a negligible one-time optimization cost.

**Inference Cost Trade-off.** GEPA-optimized prompts are substantially longer than baseline prompts (467 vs. 13 words on average), increasing per-query inference costs. However, the 15.5% absolute accuracy improvement provides significant value when amortized across production deployments handling millions of inference queries. Additionally, the prompt can be cached using advanced techniques to substantially reduce continuous inference costs.

**Comparison with Alternatives.** Manual prompt engineering requires recurring expert time (estimated at hours per task per model update), with costs that recur whenever models evolve or task requirements change. In contrast, GEPA's automated evolution completes in a single optimization cycle at $0.04, producing prompts that remain effective until the underlying model changes. For frontier LLM-based evolution (e.g., GPT-4 as Gödel Agent optimizer), token costs are approximately 30–60× higher per token, and the resulting system also requires expensive frontier models at inference time. SLM evolution enables both cheap optimization *and* cheap deployment.

### 4.4 Ceiling Effect Analysis

The contrast between Haiku 3 and Haiku 4.5 results reveals an important phenomenon: *diminishing marginal returns of prompt-level evolution near optimal performance*. Haiku 3 gains +15.5% on MGSM (67%→77%) while Haiku 4.5 gains only +1.3% (93%→94.2%). This pattern is consistent across benchmarks (Table 2).

We hypothesize two contributing factors. First, models with stronger baselines have already internalized effective reasoning strategies during pretraining, leaving less room for prompt-injected improvements. Second, near-ceiling performance means remaining errors are "hard" instances requiring capabilities beyond what prompt engineering can provide (e.g., knowledge gaps, fundamental reasoning limitations).

This observation has practical implications: prompt-level self-evolution is most impactful for models operating below their theoretical ceiling, suggesting that *weaker SLMs stand to benefit most* from self-evolution frameworks. For near-optimal models, code-level self-modification (as in Gödel Agent) or fine-tuning may be necessary to achieve further gains. Future work should investigate using a frontier LLM as teacher in GEPA to isolate whether the bottleneck lies in the student's execution capacity or the teacher's ability to propose effective improvements, and analyze patterns in rejected/unsuccessful prompt mutations to identify systematic failure modes.

## 5 Conclusion

This work presents an empirical study measuring how self-evolution effectiveness scales down

from frontier models to Small Language Models (SLMs). Using Gödel Agent and GEPA frameworks, we demonstrate that models like Qwen3-4B and Claude Haiku can achieve meaningful autonomous improvement through recursive self-modification and reflective prompt evolution, with accuracy gains up to 15.5% on mathematical reasoning (MGSM, confirmed over multiple runs: 74.75% $\pm$ 3.85%), 4% on ARC science assessment, and 3.3% on general knowledge assessment (MMLU). While these gains are more modest compared to frontier LLMs (GPT-4 achieves $\sim$36% improvement on MGSM), the evolution cost of just $0.04 per optimization cycle demonstrates the economic viability of SLM-driven self-evolution for scalable production deployments.

These findings suggest that self-evolution is feasible at smaller model scales, enabling deployments with significant computational and economic advantages, including consumer-grade hardware deployment, reduced energy consumption, and enhanced privacy through local execution. The ceiling effect observed in Haiku 4.5 indicates that prompt-level evolution is most impactful for models with headroom for improvement, while near-optimal models may require code-level or fine-tuning-based approaches.

Future work should extend this investigation to agentic task domains (e.g., SWE-bench for coding (Jimenez et al., 2023) beyond reasoning and QA, evaluate additional SLM architectures and self-evolution frameworks, investigate using frontier LLMs as teachers to isolate student vs. optimizer bottlenecks, and validate self-evolution effectiveness in multi-agent coordination scenarios where multiple SLM agents must collaboratively solve complex real-world tasks.

## 6 Limitations

While our study demonstrates promising results for SLM-driven evolution, several limitations should be considered:

1. *Limited model selection:* We evaluated two SLM families (Qwen3 and Haiku). Evolution capabilities may not generalize to other architectures (e.g., Phi-4, Gemma).

2. *Benchmark specificity:* Experiments focused on reasoning and QA benchmarks (MMLU, ARC, MGSM). Effectiveness on agentic tasks such as tool use (SWE-bench), web browsing (WebArena), and multi-turn dialogue remains unexplored.

3. *Modest gains relative to frontier models:* While SLMs achieve consistent improvements, the magnitude is substantially smaller than frontier LLMs (e.g., +3.3% vs. $\sim$+5% on MMLU, +1.7% vs. $\sim$+36% on MGSM for Gödel Agent). Our contribution is demonstrating feasibility, not parity.

4. *Cross-framework comparison:* We did not directly compare Gödel Agent and GEPA using the same SLMs, as our study aimed to establish whether SLMs can drive self-evolution rather than benchmark evolution frameworks against each other.

## References


Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J Hewett, Mojan Javaheripi, Piero Kauffmann, and 1 others. 2024. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*.

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, and 1 others. 2025. Gepa: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*.

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, and 1 others. 2025. Smollm2: When smol goes big–data-centric training of a small language model. *arXiv preprint arXiv:2502.02737*.

Anthropic. 2024. The claude 3 model family: Opus, sonnet, haiku. Technical report, Anthropic.

Anthropic. 2025. Claude sonnet 4.5.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, and 1 others. 2025. A survey of self-evolving agents: On path to artificial super intelligence. *arXiv preprint arXiv:2507.21046*.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2023. Connecting large language models

with evolutionary algorithms yields powerful prompt optimizers. *arXiv preprint arXiv:2309.08532*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*.

OpenAI. 2025. Gpt-5 technical documentation. Ai model documentation, OpenAI.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. 2024. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 9340–9366, Miami, Florida, USA. Association for Computational Linguistics.

Jiahao Qiu, Xuan Qi, Tongcheng Zhang, Xinzhe Juan, Jiacheng Guo, Yifu Lu, Yimin Wang, Zixin Yao, Qihan Ren, Xun Jiang, and 1 others. 2025. Alita: Generalist agent enabling scalable agentic reasoning with minimal predefinition and maximal self-evolution. *arXiv preprint arXiv:2505.20286*.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, and 1 others. 2022. Language models are multilingual chain-of-thought reasoners. *arXiv preprint arXiv:2210.03057*.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, and 1 others. 2025. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2023. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*.

Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. 2024. G"odel agent: A self-referential agent framework for recursively self-improvement. *arXiv preprint arXiv:2410.04444*.

Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. 2025. Gödel agent: A self-referential agent framework for recursively self-improvement. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 27890–27913, Vienna, Austria. Association for Computational Linguistics.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. 2025. Optimizing generative ai by backpropagating language model feedback. *Nature*, 639(8055):609–616.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. 2025a. Darwin godel machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. 2025b. Darwin godel machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2022. Steering large language models using ape. In *NeurIPS ML Safety Workshop*.

## A Gödel Agent Qualitative Analysis

We examined how Gödel Agent's self-referential learning transforms solver implementations, evolving from simple function wrappers into sophisticated, fault-tolerant reasoning systems.

**Solver Evolution:** Figures 1 and 2 show the transformation of the MMLU solver. The initial implementation uses a single LLM call with temperature 0.8, basic requirements, and minimal error handling. Through recursive self-modification, Gödel Agent evolved this into a robust system with:

- **Enhanced Robustness:** Multiple error handling layers with JSON parsing recovery and graceful fallbacks
- **Intelligent Answer Extraction:** Regex-based pattern matching to extract answers from reasoning text
- **Confidence-Based Decision Making:** Scoring mechanism evaluating reasoning quality across multiple dimensions
- **Parameter Optimization:** Temperature reduction from 0.8 to 0.7 and more explicit directive requirements

## B GEPA Qualitative Analysis

We analyzed how prompts evolved with GEPA, observing systematic expansion from generic instructions to task-specific guidance.

**Initial Solver (Baseline)**

```python
def solver(agent, task: str):
    print(f"SOLVER CALLED: {task[:100]}")

    messages = [{
        "role": "user",
        "content": f"# Your Task:\n{task}"
    }]

    print(f"Sending: {messages}")

    response = agent.action_call_json(
        model="default",
        messages=messages,
        temperature=0.8,
        num_of_response=1,
        role="knowledge and reasoning expert",
        return_dict_keys=[
            "reasoning", "answer"
        ],
        requirements=(
            "1. Explain step by step.\n"
            "2. Answer MUST be A/B/C/D.\n"
        ).strip(),
    )

    print(f"Raw response: {response}")

    return_dict = response[0]
    return_dict["answer"] = str(
        return_dict.get("answer", "")
    )

    print(f"Final: {return_dict}")
    return return_dict
```

**Characteristics:**

- Debug logging (3 prints)
- Temperature: 0.8
- No error handling
- No validation
- Basic string conversion

Figure 1: Initial Solver (Baseline). A simple 30-line function with basic functionality and debug logging.

**MMLU Prompt Evolution:** The original prompt *"You are an expert test-taker"* evolved into multi-paragraph instructions with:

- **Domain Recognition:** Specialized guidance for history, law, economics, etc.
- **Reasoning Strategies:** Step-by-step reasoning and elimination heuristics
- **Common Pitfalls:** Explicit warnings about typical errors

**MGSM Prompt Evolution:** The original prompt *"You are a math expert"* evolved to include:

- **Multilingual Awareness:** Instructions for non-English problems
- **Step-by-Step Framework:** Structured mathematical problem-solving guidance
- **Arithmetic Safeguards:** Instructions to prevent common calculation errors

**Evolved Solver (Post-Gödel)**

```python
def solver(agent, task: str):

    ...

    response = agent.action_call_json(
        model="default", messages=messages,
        temperature=0.7 ,  # Changed!
        num_of_response=1,
        role="knowledge and reasoning expert",
        return_dict_keys=["reasoning","answer"],
        requirements=(
            "1. Explain step by step.\n"
            "2. Answer MUST be A/B/C/D.\n"
            "3. If unsure, respond with 'D'."
            "4. Provide a clear, concise reasoning
            "with a single confident answer."
        ).strip(),
    )

    return_response = response[0]

    # === ERROR HANDLING ===
    if hasattr(return_response, 'content'):
        try:
            return_response = json.loads(
                return_response.content)
        except (JSONDecodeError, TypeError):
            return_response = {
                "reasoning": "Failed",
                "answer": "D"}

    # === ANSWER VALIDATION ===
    final_answer = return_response.get(
        "answer", "")
    if not final_answer:
        final_answer = "D"

    valid_answers = {"A", "B", "C", "D"}
    if final_answer not in valid_answers:
        # Extract from reasoning text
        reasoning = return_response.get(
            "reasoning", "")
        for opt in valid_answers:
            if opt.lower() in reasoning.lower():
                final_answer = opt
                break
        else:
            final_answer = "D"

    # === PATTERN MATCHING ===
    import re
    reasoning = return_response.get("reasoning", "")
    patterns = [r'\bA\b', r'\bB\b', ...]
    matches = []
    for pattern in patterns:
        matches.extend(re.findall(
            pattern, reasoning, re.IGNORECASE))
    ...  # Frequency analysis
    if matches:
        final_answer = most_frequent_option

    # === CONFIDENCE SCORING ===
    confidence = 0
    if "step by step" in reasoning.lower():
        confidence += 1
    if len(matches) > 0:
        confidence += len(matches) // 2
    ...  # More scoring logic

    if confidence < 1:
        final_answer = "D"

    return_response["answer"] = final_answer
    return return_response
```

**New Features:**

- **Error handling** (try-catch)
- **Multi-layer validation**
- **Regex pattern extraction**
- **Confidence-based fallback**

Figure 2: Evolved Solver (Post-Gödel). Orange highlights modified parameters, green highlights new functionality.