

000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015 016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031 032 033 034 035 036 037 038 039 040 041 042 043 044 045 046 047 048 049 050 051 052 053 DARWIN GÖDEL MACHINE: OPEN-ENDED EVOLUTION OF SELF-IMPROVING AGENTS

Anonymous authors

Paper under double-blind review

ABSTRACT

Most of today’s AI systems are constrained by human-designed, fixed architectures and cannot autonomously and continuously improve themselves. The scientific method, on the other hand, is a cumulative and open-ended system, where each innovation builds upon previous artifacts, enabling future discoveries. There is growing hope that the current manual process of advancing AI could itself be automated. If done safely, such automation would accelerate AI development and allow us to reap its benefits much sooner. This prospect raises the question of how AI systems can endlessly improve themselves while getting better at solving relevant problems. Meta-learning can automate the discovery of novel algorithms, but is limited by first-order improvements and the human design of a suitable search space. The Gödel machine (Schmidhuber, 2007) proposed a theoretical alternative: a self-improving AI that repeatedly modifies itself in a provably beneficial manner. Unfortunately, proving that most changes are net beneficial is impossible in practice. We introduce the Darwin Gödel Machine (DGM), a novel self-improving system that iteratively modifies its own code (thereby also improving its ability to modify its own codebase) and empirically validates each change using coding benchmarks. Inspired by Darwinian evolution and open-endedness research, the DGM grows an archive of generated coding agents. It samples agents from this archive, which self-modify to create new, interesting versions of themselves. This open-ended exploration forms a growing tree of diverse, high-quality agents and allows the parallel exploration of many different paths through the search space. Empirically, the DGM automatically improves its coding capabilities (e.g., better code editing tools, long-context window management, peer-review mechanisms), increasing performance on SWE-bench from 20.0% to 50.0%, and on Polyglot from 14.2% to 30.7%. Furthermore, the DGM significantly outperforms baselines without self-improvement or open-ended exploration. All experiments were done with safety precautions (e.g., sandboxing, human oversight). Overall, the DGM represents a significant step toward self-improving AI, capable of gathering its own stepping stones along a path that unfolds into endless innovation.

1 INTRODUCTION

Scientific progress is cumulative and open-ended, with each breakthrough standing on the shoulders of countless prior insights. In the same way, our most advanced AI systems are built upon a long lineage of innovations. For instance, transformers (Vaswani et al., 2017), the backbone of current large language models (LLMs) (Brown et al., 2020), did not emerge in isolation but were built upon years of past innovations, such as recurrent neural networks (Linnainmaa, 1970; Amari, 1972; Hopfield, 1982; Rumelhart et al., 1985) and attention mechanisms (Schmidhuber & Huber, 1990; Bahdanau et al., 2015; Kim et al., 2017; Parikh et al., 2016). However, most of today’s AI systems remain bound by fixed, human-designed architectures that learn within predefined boundaries, without the capacity to autonomously rewrite their own source code to self-improve. As a result, each advancement in AI development still leans heavily on human interventions, tethering the pace of progress. This paper investigates the intriguing possibility of safely automating the search for ever-better AI. One can imagine an AI system that, like scientific discovery itself, becomes an engine of its own advancement: building upon its past, recursively improving, and propelling itself toward more advanced capabilities.

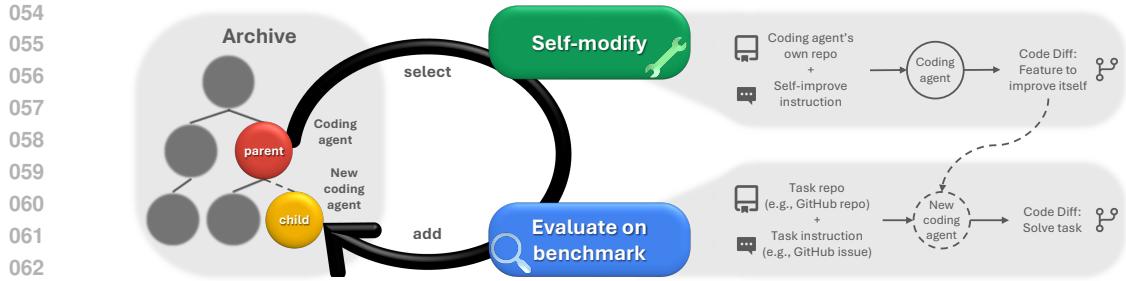


Figure 1: **Darwin Gödel Machine**. The DGM iteratively builds a growing archive of agents by interleaving self-modification with downstream task evaluation. Agents in the archive are selected for self-modification through open-ended exploration.

Schmidhuber (2007) presented a class of mathematically rigorous, self-referential, self-improving problem solvers. It relies on formal proofs to justify code rewrites, ensuring that any self-modification is provably beneficial. However, in practice and without restrictive assumptions about the system, it is impossible to formally prove whether a modification to an AI system will be beneficial. For example, while it may seem that an LLM-based coding agent would benefit from access to more tools (e.g., code search, test runners), the actual impact depends heavily on the model’s training and task context (e.g., a testing tool that is optimized for one setup may confuse the agent when working with others). Instead of requiring formal proofs, we empirically validate self-modifications against a benchmark, allowing the system to improve and explore based on observed results. This approach mirrors biological evolution, where mutations and adaptations are not verified in advance but are produced, trialed, and then selected via natural selection. We also take inspiration from Darwinian evolution (Darwin, 2023) and investigate the effectiveness of maintaining a library of previously discovered agents to serve as stepping stones for future generations.

We propose the **Darwin Gödel Machine (DGM)**, a self-referential, self-improving system that writes and modifies its own code to become a better coding agent. Each self-modification requires the DGM to edit its own codebase. We use Python, which is Turing-complete, giving the DGM the potential to build any computable machine. Our framework envisions agents that can rewrite their own training scripts (including training a new foundation model (FM)). However, we do not show that in this paper, as training FMs is computationally intensive and would introduce substantial additional complexity, which we leave as future work. Instead, this paper focuses on improving the design of coding agents with frozen pretrained FMs (e.g., tool use, workflows). The DGM alternates between self-modification and evaluation phases. During the self-modification phase, selected coding agents from the archive generate modified versions of themselves. During the evaluation phase, each modified agent is tested on a coding benchmark, estimating the agent’s coding capabilities, and then added to the archive. By improving its own capabilities through this loop, the DGM becomes better at both solving coding tasks and making future self-improvements. A key assumption is that an increase in performance on coding benchmarks indicates better coding capabilities, and hence better ability to self-modify and self-improve. Furthermore, the DGM maintains an archive of generated coding agents, initialized with only one agent, and continuously accumulates all generated variants over time. To support continual self-improvement, the DGM draws inspiration from open-endedness research (Wang et al., 2019; Fernando et al., 2024; Faldor et al., 2025), accumulating diverse stepping stones (i.e., interesting yet suboptimal solutions or features that may enable future breakthroughs). This open-ended exploration encourages the discovery of novel and potentially useful self-modifications beyond immediate performance gains.

We present results on two coding benchmarks: SWE-bench (Jimenez et al., 2024) and Polyglot (Paul Gauthier, 2024). The DGM automatically improves itself from 20.0% to 50.0% on SWE-bench, and from 14.2% to 30.7% on Polyglot. We show that self-improvement enables continued progress, as the DGM outperforms the baseline where the same base agent is repeatedly used to modify and generate new agents without self-improvement. We also show that open-ended exploration and keeping an archive of all previously generated agents lead to the discovery of better coding agents. The DGM outperforms the baseline of not having open-ended exploration (i.e., a baseline without the accumulation of an archive of interestingly different stepping stones), where the coding agent always builds off the most recent version of itself. Overall, the DGM represents a step toward AI systems

108 that can build upon their own prior innovations and improve recursively. We consider and discuss
 109 safety aspects extensively, including sandboxing and traceability of self-modifications, to ensure
 110 responsible experimentation (Section 5). By advancing the possibility of safe, self-referential, self-
 111 improving models, the DGM moves us closer to AI that not only learns but evolves in an open-ended,
 112 self-accelerating trajectory, much like science itself.

2 RELATED WORK

117 **Open-Endedness.** A grand challenge for driving unbounded innovation is designing open-ended AI
 118 systems that continuously generate novel and learnable artifacts (Stanley et al., 2017). Hughes et al.
 119 (2024) characterized open-endedness as a system’s capacity to generate sequences of artifacts that are
 120 both novel and learnable from an observer’s perspective. A central difficulty lies in structuring and
 121 exploring vast search spaces to consistently produce artifacts that are interesting to humans (Clune,
 122 2019; Jiang et al., 2023). Early progress drew on quality-diversity algorithms, goal-directed ex-
 123 ploration, intrinsic motivation, and learning-progress frameworks (Pugh et al., 2016; Ecoffet et al.,
 124 2019; Lehman & Stanley, 2011; Oudeyer et al., 2007), while recent advances leverage large-scale
 125 foundation models (FMs) as proxies for human interestingness and versatile engines for generating
 126 and evaluating novel behaviors across diverse domains (Brown et al., 2020; Hu et al., 2025; Zhang
 127 et al., 2024b). However, these approaches have yet to close the self-referential self-improvement
 128 loop, meaning improvements on downstream tasks do not translate into enhanced capabilities for
 129 self-modification or the acceleration of further innovations. We aim to mimic the acceleration of sci-
 130 ence and technology, where new tools and discoveries catalyze the creation of even more discoveries.
 131 How can we emulate nature’s arc of evolution, which bends not only toward complexity but also an
 132 ever greater capacity to evolve (Dawkins, 2019; Gerhart & Kirschner, 2007; Hendrikse et al., 2007)?

133 **Meta-Learning FM Agents.** Many FM-based agents are handcrafted. Some building blocks include
 134 prompt engineering (Chen et al., 2023; Schulhoff et al., 2024), chain-of-thought (Wei et al., 2022;
 135 Yao et al., 2023; Hu & Clune, 2024; Guo et al., 2025; Lightman et al., 2023; Muennighoff et al., 2025;
 136 Zelikman et al., 2024a), self-reflection (Shinn et al., 2023; Yao et al., 2023; Madaan et al., 2023),
 137 multi-agent debate (Zhuge et al., 2023; Liang et al., 2023; Khan et al., 2024), memory (Liu et al., 2023;
 138 Zhong et al., 2024; Modarressi et al., 2023), temperature sampling (Zhu et al., 2024), and retrieval
 139 augmented generation (Lewis et al., 2020). The manual composition of these components limits
 140 the system’s abilities to the ingenuity of its human designer. More recently, several meta-learning
 141 approaches have emerged that leverage FM to automatically optimize prompts (Fernando et al., 2024;
 142 FAIR; Khattab et al., 2023; Cheng et al., 2024; Yuksekgonul et al., 2024; Yuan et al., 2024) and
 143 design agentic modules (Zhang et al., 2024c; Zhou et al., 2024; Yin et al., 2024; Zhuge et al., 2024;
 144 Rosser & Foerster, 2025; Zhang et al., 2025a; Ye et al., 2025; Gao et al., 2025; Nie et al., 2025;
 145 Su et al., 2025; Zhang et al., 2025b; Niu et al., 2025). The Automated Design of Agentic Systems
 146 (ADAS, Hu et al., 2025) iteratively generates downstream agents with a fixed meta-agent, evaluates
 147 them against a target benchmark, and incorporates feedback to refine subsequent generations. In
 148 contrast, the DGM is a single system that both solves downstream tasks (i.e., coding problems)
 149 and refines its own implementation (i.e., its codebase), removing the need for a fixed, handcrafted
 150 meta-agent and enabling self-referential improvements.

151 **Self-Improving AI.** Early on, various researchers outlined theoretical and conceptual approaches to
 152 self-improvement (Good, 1966; Schmidhuber, 1987; 2007). Some practical approaches to automated
 153 self-improvement include systems defined by neural network weight parameterizations (Schmidhuber,
 154 1993; Hall, 2007; Hobbhahn, 2025; Kirsch & Schmidhuber, 2022; Irie et al., 2022; 2025; Lu et al.,
 155 2023; Havrilla et al., 2024b). Metz et al. (2021) developed a gradient-based optimizer that is self-
 156 referentially meta-trained using a variant of population-based training (Jaderberg et al., 2017). Lange
 157 et al. (2023) extended this approach to gradient-free learning. Silver et al. (2017) used self-play
 158 to continuously evolve agents, achieving superhuman performance in challenging domains such
 159 as chess and Go. More closely related to the DGM are recent approaches that leverage FM-based
 160 agents for self-improvement (Yin et al., 2024; Robeyns et al., 2025; Hu et al., 2024; Zelikman
 161 et al., 2024b; Huang et al., 2022; Singh et al., 2023). Zelikman et al. (2024b) use a meta-agent
 162 to generate downstream agents, updating the meta-agent based on the meta-utility derived from
 163 the generated solutions. Yin et al. (2024) use a single system to both solve downstream tasks and
 164 recursively modify itself. However, the downstream tasks or the meta-utility do not always align
 165 with the capabilities required for self-improvement. In the DGM, improvement in downstream tasks

162 directly reflects an increase in self-improvement ability, enabling the potential for self-accelerating
 163 progress. Most similar is concurrent work by Robeyns et al. (2025), which also has a single agent
 164 recursively solving coding problems and modifying its own codebase. The main difference from
 165 Robeyns et al. (2025) (and also Zelikman et al. (2024b); Yin et al. (2024)) is that the DGM has an
 166 open-ended exploration loop, encouraging self-modifications beyond immediate performance gains
 167 and thus avoiding stagnation in suboptimal states. Appendix B also discusses additional related work
 168 on program synthesis and Darwinian evolution.

170 3 DARWIN GÖDEL MACHINE

172 A Gödel Machine is a theoretical idea of an AI that searches for ways that *provably* improve
 173 itself (Schmidhuber, 2007). In this paper, we propose Darwin Gödel Machine (DGM), an attempt to
 174 realize the long-held dream of creating a Gödel Machine. The DGM relaxes the Gödel Machine’s
 175 impractical requirement of theoretically *proving* that a change will improve the system, instead
 176 requiring *empirical evidence* from experiments to demonstrate that a proposed new version enhances
 177 performance. Additionally, since the DGM relies on empirical evidence of improvement, it may
 178 get stuck in a local optimum within the vast search space of possible systems (i.e., all computable
 179 algorithms). To address this, the DGM maintains an archive of discovered solutions during the
 180 search, facilitating open-ended exploration rather than relying on evolving a single solution. Since the
 181 principles echo Darwinian evolution (Darwin, 2023) (Appendix B), where new innovations emerge
 182 by selecting an entity from an archive of previously discovered solutions, modifying it, and keeping
 183 it if it is interestingly new (Zhang et al., 2024b; Faldor et al., 2025; Stanley & Lehman, 2015), we
 184 call our algorithm a Darwin Gödel Machine (Figure 1).

185 **Self-referential Self-improvement of Coding Agents.** The DGM is initialized with only one coding
 186 agent, and its progression is evaluated on coding benchmarks. A coding agent is defined as a single
 187 system, implemented with a code repository and powered by frozen pretrained foundation models
 188 (FMs), capable of reading, writing, and executing code. Code, when expressed in a general-purpose
 189 Turing-complete language (e.g., Python), is a powerful medium for building and improving intelligent
 190 systems because it can represent any computable process. Recent works (Hu et al., 2025; Zhang et al.,
 191 2024c) demonstrate that such agents can be improved through meta-learning of their designs (e.g.,
 192 prompts, workflows, and tools), which are implemented in their code repository. Therefore, we define
 193 self-improvement as a coding task that involves modifying the design of an agent’s own components
 194 (i.e., its own code, which does not include the open-ended exploration process described in the next
 195 paragraph). The key motivation is that the empirical evidence must reflect the system’s ability to both
 196 self-improve and solve downstream tasks. By configuring the DGM as a coding agent and testing
 197 its coding capabilities, the observed improvements demonstrate not only enhanced performance
 198 in downstream tasks but also the capacity for further self-improvement, as self-improvement is
 199 fundamentally a coding task that modifies the coding agent’s own code repository.

200 **Population-based Open-ended Exploration.** Starting from a single initial coding agent, the DGM
 201 builds an archive of all discovered agents. In each iteration, the DGM selects parent agents to self-
 202 modify and branch off to produce new agents. Parent selection is roughly proportional to each agent’s
 203 performance score and inversely proportional to the number of its children with codebase-editing
 204 functionality (Appendix C.2). This favors high-performing agents that have been underexplored
 205 (i.e., have fewer existing children), thereby promoting both exploitation of strong performers and
 206 exploration of promising but less-sampled lineages. All agents retain a non-zero selection probability,
 207 ensuring that any path to improvement remains feasible given sufficient compute. Each selected
 208 parent analyzes its own benchmark evaluation logs, proposes the next feature to implement, and
 209 receives this proposal as a problem statement to execute (Appendix C.3). The parent then implements
 210 the suggested feature into its own codebase, generating a new coding agent. Each newly generated
 211 agent is quantitatively evaluated on a chosen coding benchmark to estimate its coding abilities. Only
 212 agents that compile successfully and retain the ability to edit a given codebase are added to the
 213 DGM archive, as only they can continue self-modification. All others are discarded. The cycle of
 214 parent selection, self-modification, and evaluation continues, progressively growing the archive of
 215 solutions. Importantly, we note that archived solutions can serve as stepping stones that result in
 improvements much later than their original discovery, making our approach substantially different
 from hill-climbing agentic design approaches (Robeyns et al., 2025). Currently, the open-ended
 exploration process (i.e., archive maintenance, parent selection) is fixed and not modifiable by the

216 DGM, which we leave as an avenue for future work. Appendix C.4 shows the pseudocode for the
 217 DGM algorithm.
 218

219 220 4 EXPERIMENTS

221
 222 Section 4.1 describes the experimental setup, including the initial coding agent that initializes the
 223 DGM. We conduct experiments on two coding benchmarks: SWE-bench (Jimenez et al., 2024) and
 224 Polyglot (Paul Gauthier, 2024) (Section 4.2). For each benchmark, we compare the DGM against
 225 two baselines: DGM without self-improving agents and DGM without open-ended exploration
 226 (Section 4.3). Across all experiments, we find that the DGM outperforms both baselines, showing that
 227 the self-improvement mechanism and open-ended exploration are essential for sustained performance
 228 gains (Section 4.4). Furthermore, we show that the features discovered by the DGM transfer across
 229 models, benchmarks, and tasks (Section 4.4).
 230

231 4.1 EXPERIMENT SETUP

232 The DGM is initialized with a single coding agent. This base agent is built around a frozen FM and
 233 augmented with tool use capabilities (Schick et al., 2023; Anthropic, 2024b). The FM autonomously
 234 determines its action at each step, using whichever tool it deems appropriate. The base agent has
 235 access to two tools: a Bash tool for executing bash commands, and an edit tool for viewing and
 236 editing directories and entire files (Appendix C.1). The agent receives a single input prompt that
 237 specifies the location of the target repository, the problem statement, and the repository’s testing
 238 framework (Appendix C.1). The base agent is intentionally lightweight to provide a simple starting
 239 point from which self-improvement can be clearly demonstrated. We run the DGM for 80 iterations
 240 (generating one new agent per iteration), with two iterations running in parallel for SWE-bench
 241 and four for Polyglot (Appendix C.2). During self-modifications, coding agents are powered by
 242 Claude 3.5 Sonnet (New) (Anthropic, 2024a) in both SWE-bench and Polyglot experiments. During
 243 benchmark evaluation, coding agents are powered by Claude 3.5 Sonnet (New) for SWE-bench and
 244 o3-mini (OpenAI, 2025) for Polyglot (Appendix D.1).
 245

246 4.2 BENCHMARKS

247 We evaluate the DGM on two popular benchmarks that assess different aspects of coding tasks
 248 to validate the algorithm’s effectiveness across various use cases. Both SWE-bench and Polyglot
 249 are benchmarks that evaluate how well AI agents solve coding tasks automatically. Given a code
 250 repository and a task instruction, the agent is expected to make changes to the repository in order to
 251 fulfill the task. Both SWE-bench and Polyglot are widely used benchmarks (Zhang et al., 2024d;a;
 252 Xia et al., 2024; Cao et al., 2024; Google DeepMind, 2025; Gauthier, 2024) that require the AI agent
 253 to navigate a code repository, understand the interplay between functions in different files, and spot
 254 small errors in convoluted code. SWE-bench only has Python tasks, while Polyglot has tasks in
 255 multiple programming languages. Another difference is that each SWE-bench task may require edits
 256 to multiple files, whereas each Polyglot task primarily involves implementing a solution from scratch
 257 in a single file (although the agent still needs to examine other files to understand what changes are
 258 necessary), resulting in fewer file edits overall.

259 **SWE-bench.** To avoid wasting compute on unsolvable tasks, we use SWE-bench Verified (OpenAI,
 260 2024), a human-filtered subset of SWE-bench (Jimenez et al., 2024) where all tasks are solvable.
 261 Throughout this paper, the term SWE-bench refers by default to the SWE-bench Verified subset.

262 **Polyglot.** Polyglot includes tasks in multiple programming languages (C++, Rust, Python, etc.) (Paul
 263 Gauthier, 2024). Compared to SWE-bench, one of the most widely used coding benchmarks and
 264 likely included in the training sets of FMs, Polyglot is more niche and less likely to be included
 265 in FMs’ post-training data. Additionally, Polyglot is primarily used by its developer to evaluate
 266 Aider (Gauthier, 2024). This provides an opportunity to compare automatically designed agents with
 267 a representative coding agent in the community, which has been continuously developed and tested
 268 against the Polyglot benchmark over a year by human developers. Finally, Polyglot enables testing of
 269 the DGM’s ability to generalize across multiple programming languages beyond just Python. We
 follow the Polyglot leaderboard setup (Paul Gauthier, 2024), with one key difference: the leaderboard

270 reports pass@2, where the agent can view feedback from ground-truth tests once, whereas we report
 271 pass@1, where the agent never sees ground-truth test results.
 272

273 Since it is expensive to evaluate on the full benchmarks (Appendix E.1), we use a staged evaluation
 274 strategy to estimate the coding agent’s performance. In general, we evaluate the coding agent on
 275 more tasks when we are more confident that it is a relatively strong performer. We first evaluate
 276 each coding agent on a set of 10 tasks to verify basic codebase-editing functionality. Only agents
 277 that retain the ability to edit code can solve downstream coding tasks or perform further self-
 278 modifications. Agents that pass this initial stage are then evaluated on an expanded set of 50 tasks.
 279 For SWE-bench, tasks are selected based on SWE-bench-verified-mini (Hobbhahn, 2025), which is
 280 designed to be a representative sample of the full benchmark (Appendix E.2). For Polyglot, tasks
 281 are chosen as a random subset of the full benchmark (Appendix E.3). Because the LLMs we use
 282 are inherently stochastic, performance can be noisy. We use smaller subset evaluations to estimate
 283 general effectiveness and identify top candidates. For SWE-bench, if a coding agent achieves over
 284 40% success (a heuristic chosen based on the noise observed in preliminary runs) on the 60-task
 285 subset and ranks among the top two performers in the archive, it is then evaluated on 200 tasks
 286 (140 plus the previous 60) to more accurately assess its true coding capability (Appendix E.2). For
 287 Polyglot, if a coding agent achieves over 40% success on the 10-task subset, it is evaluated on an
 288 expanded 50-task subset (Appendix E.3).
 289

290 4.3 BASELINES

291 To isolate and evaluate the contribution of the DGM’s core components, we compare the DGM against
 292 two baselines: DGM without self-improving agents (**DGM w/o self-improve**) and DGM without
 293 open-ended exploration (**DGM w/o open-ended exploration**). DGM w/o self-improve replicates the
 294 approach of ADAS (Hu et al., 2025) in this setting, wherein the meta agent responsible for modifying
 295 the coding agents remains fixed as the base agent throughout the experiment (Algorithm 2). This
 296 baseline allows us to assess the role of iterative self-improvement in accelerating the evolution of
 297 better coding agents. DGM w/o open-ended exploration eliminates the use of an archive and always
 298 self-modifies the latest stored version of itself (Algorithm 3). If a coding agent self-modifies to the
 299 point where it loses the basic functionality required to edit a codebase, it can no longer modify itself
 300 or solve any coding task. Therefore, DGM w/o open-ended exploration retains the latest version
 301 of itself that still maintains the basic functionality for codebase editing. This baseline allows us to
 302 evaluate the impact of having an archive and the well-documented beneficial principles of open-ended
 303 exploration (Clune, 2019; Stanley & Lehman, 2015; Zhang et al., 2024b; Fernando et al., 2024; Lee
 304 et al., 2020; Samvelyan et al., 2024; Colas et al., 2022b) in guiding the agent’s evolution.

305 In addition to the learned baselines, we compare the DGM against handcrafted, open-source solutions.
 306 For SWE-bench, we take the state-of-the-art (SoTA) open-source solution that has been checked
 307 (i.e., the SWE-bench team was able to reproduce the results) (Appendix E.4). For Polyglot, we take
 308 the representative agent (Aider) (Gauthier, 2024), which is open-sourced and designed to support
 309 multiple programming languages and large codebase editing (Appendix E.5). For a fair comparison,
 310 we measure the percentage of solved tasks on the same benchmark subsets used to evaluate the DGM
 311 (Appendix E.2, Appendix E.3). These values are shown as dotted horizontal lines in Figure 2.

312 4.4 RESULTS

313 After 80 iterations of the DGM, the coding agent’s performance increases from 20.0% to 50.0% on
 314 SWE-bench, and from 14.0% to 38.0% on Polyglot (Figure 2). Since the DGM is evaluated on only
 315 50 tasks in the Polyglot experiment setup (Section 4.2), we additionally evaluate both the base agent
 316 and the best DGM-discovered agent on the full Polyglot benchmark to more accurately estimate the
 317 improvement. On the full Polyglot benchmark, the DGM improves the coding agent from 14.2% to
 318 30.7%. This shows that the DGM can automatically self-improve to create a better coding agent.
 319 Moreover, the performance of the best DGM-discovered agent is comparable to that of the checked,
 320 open-source, human-designed SoTA on SWE-bench (Figure 2). On Polyglot, although the DGM
 321 starts with a base agent whose performance is lower than that of Aider, it discovers an agent that far
 322 surpasses Aider (Figure 2). The DGM-discovered agents are comparable to or outperform handcrafted
 323 agents on both benchmarks. While the SoTA SWE-bench agent and Aider were painstakingly shaped

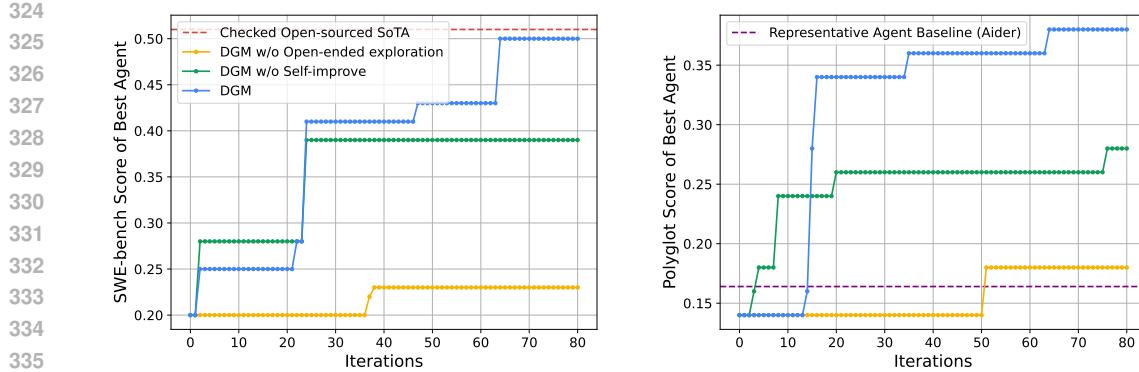


Figure 2: **Self-improvement and open-ended exploration enable the DGM to continue making progress and improve its performance.** The DGM automatically discovers increasingly better coding agents and performs better on both (Left) SWE-bench and (Right) Polyglot. It outperforms baselines that lack either self-improvement or open-ended exploration, showing that both components are essential for continual self-improvement. These scores are obtained from evaluating on the benchmark subsets detailed in Section 4.2.

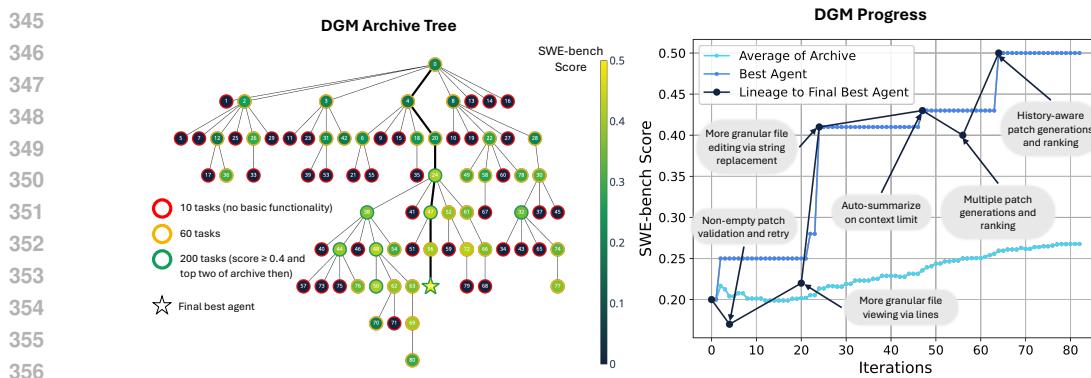


Figure 3: **The DGM automatically self-improves to become a better coding agent.** (Left) Archive of coding agents generated during the DGM run on SWE-bench. Each node represents a coding agent, with node 0 corresponding to the base agent. Node color indicates performance on SWE-bench (percentage of solved tasks), while border color reflects the number of tasks for which the agent was evaluated. Edges show which agents self-modified to produce the offsprings. Many paths to innovation traverse lower-performing nodes, and key innovations (like node 24) lead to an explosion of innovations built on top of them. Both properties underscore the benefits of open-ended search. (Right) Progress plot of the DGM on SWE-bench. The light blue line shows the average score of all agents possessing basic codebase-editing functionality. The blue line tracks the best score achieved by any agent in the archive at each iteration. The dark line shows the lineage of the final best-discovered agent and its precursor nodes, which includes two performance dips. This illustrates the benefits of open-ended search, which explores a diverse set of interesting stepping stones instead of focusing only on branching off the best solution found so far.

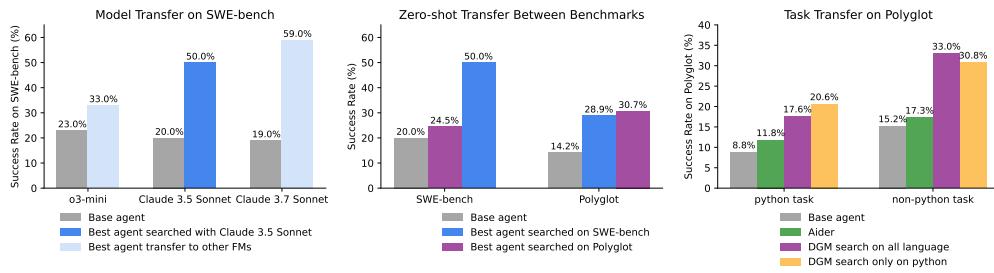
by human efforts, the DGM hints at a future in which such ingenuity is automated, evolving through self-referential cycles of continuous self-improvements.

The DGM automatically improves both the tools and the workflow of how FMs are utilized (Figure 3). For example, the DGM enhanced the edit tool to allow more granular file viewing (by lines) and more precise file editing (by string replacement), instead of always viewing or replacing the entire file. Workflow improvements include making multiple attempts to solve a task and using another FM to evaluate and select the best solution. Other workflow improvements include considering previous

378 attempts when generating subsequent ones. Appendix F.1 and Appendix F.2 show all modifications
 379 leading up to the final best-discovered agents on SWE-bench and Polyglot respectively.
 380

381 Because open-ended exploration allows branching from any agent in the archive with non-zero
 382 probability, the DGM can get out of deceptive dips or peaks in performance. For example, at
 383 iterations 4 and 56 of the experiment on SWE-bench, although the agent’s score temporarily fell
 384 below that of its parent, the DGM was still able to explore innovations along that path and create a
 385 new agent that outperformed all of its predecessors (Figure 3). Furthermore, open-ended exploration
 386 allows different implementations of the same target functionality to be attempted. For example, while
 387 the goal is to provide finer-grained editing tools, the specific implementation of this feature can vary
 388 greatly and hence lead to very different performance (Appendix G). The DGM can explore multiple
 389 implementations to find the most suitable one and avoid getting trapped in a suboptimal one.
 390

391 The DGM outperforms the baselines of DGM w/o self-improve and DGM w/o open-ended exploration
 392 on both benchmarks (Figure 2). Without updating the meta agent that modifies coding agents, DGM
 393 w/o self-improve improves the agents in early iterations, but its gains taper off quickly (Appendix A.1).
 394 In DGM w/o open-ended exploration, only the most recent agent is retained, so a poorly performing
 395 self-modification makes subsequent improvements harder to achieve (Appendix A.1).
 396



404 **Figure 4: Transfer between Models, Benchmarks, and Tasks.** The superior performance of
 405 DGM-discovered agents can be transferred across (Left) different models, (Middle) benchmarks, and
 406 (Right) different programming language tasks in Polyglot, such as from Python tasks to C++ tasks.
 407

408 To evaluate the generality of the improvements from the DGM, we tested the base agent (Section 4.1)
 409 and the best agent discovered during the DGM run (Figure 2) with different FMs than those used
 410 during optimization. For SWE-bench, where the DGM was run using Claude 3.5 Sonnet (New),
 411 we replaced the FM with Claude 3.7 Sonnet (Anthropic, 2025) or o3-mini, and evaluated on 200
 412 tasks (Figure 4, Left). With o3-mini, the base agent achieved 23.0% and the DGM-discovered agent
 413 33.0%. With Claude 3.7 Sonnet, the base agent achieved 19.0% and the DGM-discovered agent
 414 59.5%. These results suggest that the DGM yields improvements that generalize across FMs, rather
 415 than being tightly coupled to the specific FM used during its run (Figure 4). Model transfer results on
 416 Polyglot are presented in Appendix A.2.

417 Furthermore, we investigate the transferability of the DGM-discovered agent across different bench-
 418 marks and programming languages. First, we evaluate the best DGM-discovered agent from one
 419 benchmark (e.g., SWE-bench) on a completely held-out benchmark (e.g., Polyglot), and vice versa
 420 (Figure 4, Middle). The best agent evolved on SWE-bench achieves 28.9% on Polyglot, compared
 421 to the initial agent’s baseline of 14.2%. Conversely, the best agent evolved on Polyglot achieves
 422 24.5% on SWE-bench, outperforming the original baseline of 20.0%. Since each agent was optimized
 423 without ever accessing the alternate benchmark, these evaluations represent truly held-out tests. The
 424 consistent performance gains across benchmarks support our claim that DGM’s improvements reflect
 425 general skill acquisition rather than overfitting or exploitation of benchmark-specific artifacts. Second,
 426 we experiment with a version of the DGM trained exclusively on Python tasks from Polyglot and then
 427 transfer the discovered agent to tasks in other languages. Focusing primarily on Python tasks slightly
 428 improves performance on Python tasks but reduces performance on non-Python tasks compared to
 429 the DGM trained on all languages (Figure 4, Right). However, after being transferred from Python to
 430 other unseen languages during the search, the agent still achieves performance comparable to that
 431 of the DGM trained on all languages and substantially outperforms both the base agent and Aider.
 These results demonstrate the robustness of the discovered improvements, showing that they do not
 overfit to a specific programming language. We also present additional results in Appendix A.

432

5 SAFETY DISCUSSION

433

434 Systems capable of self-improvement, such as the DGM, represent a step toward more autonomous
435 AI development, aligning with long-standing goals in the field of making capable AI that can benefit
436 humanity (Schmidhuber, 1987; Clune, 2019; Markoff, 2016; Lehman, 2023). However, this capability
437 introduces unique safety considerations stemming from the system’s ability to autonomously modify
438 its own code. Modifications optimized solely for benchmark performance might inadvertently
439 introduce vulnerabilities or behaviors misaligned with human intentions, even if they improve the
440 target metric (Bostrom, 2020). In particular, if evaluation benchmarks do not fully capture all desired
441 agent properties (e.g., safety and robustness), the self-improvement loop could amplify misalignment
442 over successive generations. Iterative self-modification could also lead to increasingly complex and
443 uninterpretable internal logic, hindering human understanding, oversight, and control (Sheth et al.,
444 2025; Anwar et al., 2024; Greenblatt et al., 2024; Ganguli et al., 2022).
445 Recognizing these challenges, the current implementation and experimental setup of the DGM
446 incorporates several safeguards. All agent execution and self-modification processes are conducted
447 within isolated sandboxed environments, limiting their ability to affect the host system, and thereby
448 mitigating the risk of unintended actions. Each execution within the sandbox is subjected to a strict
449 time limit, reducing the risk of resource exhaustion or unbounded behavior. The self-improvement
450 process is currently confined to the well-defined domain of enhancing performance on specific coding
451 benchmarks by modifying the agent’s own Python codebase, thus limiting the scope of potential
452 modifications. Additionally, we actively monitor agent performance and code changes, with the
453 DGM archive providing a traceable lineage of modifications for review. At this stage, we have found
454 no evidence of harmful or malicious behavior in the generated agents, and the self-modifications have
455 been primarily focused on improving coding capabilities.
456 Conversely, a significant potential benefit of the self-improvement paradigm is that it could, in princi-
457 ple, be directed toward enhancing safety and interpretability themselves. We conduct a preliminary
458 investigation into how the DGM can be deployed in AI safety settings to develop countermeasures for
459 FM hallucination (Appendix H). Just as the DGM learns to improve its coding capabilities, it could
460 potentially discover and integrate better internal safeguards or modify itself for greater transparency
461 (e.g., incorporating principles akin to Constitutional AI (Bai et al., 2022)), if such properties were
462 included in its evaluation criteria (Rosser & Foerster, 2025). This suggests a promising, albeit
463 challenging, pathway in which self-improvement becomes a tool for building more trustworthy AI
464 systems. Additional research could also explore weaving Constitutional AI in from the start, though
465 the challenge would be incentivizing the system to retain these directives (an option worth exploring
466 is to create an unmodifiable part of the system to be able to evaluate at halt the rest).
467 The DGM demonstrates the potential of self-improving AI while still operating within safe research
468 boundaries due to the current limitations of frontier FMs and effective mitigations like sandboxing.
469 Appendix I presents additional discussion on broader safety uncertainties. We include this safety
470 discussion proactively to raise awareness about the emerging prospect of self-improving AI systems
471 and their associated safety implications, particularly as these systems inevitably become more
472 capable (Yudkowsky et al., 2008; Bostrom, 2002; Ecoffet et al., 2020; Bengio et al., 2024; Clune,
473 2019). Accordingly, we advocate for continued investigation into the safe and beneficial evolution of
474 AI-Generating Algorithms (Clune, 2019) and self-improving systems.
475

6 CONCLUSION AND LIMITATIONS

476

477 We introduce the Darwin Gödel Machine (DGM), the first self-improving system powered by FMs
478 with open-ended exploration, where progress on its evaluation benchmarks can directly translate
479 into better self-improvement capabilities. We demonstrate the automatic discovery of better tools
480 and FM systems, resulting in better performance on two benchmarks: SWE-bench and Polyglot.
481 Through self-improvement and open-ended exploration, the DGM shows a continuous increase in
482 performance, bringing us one step closer to self-accelerating, self-improving AI systems.
483 We demonstrate that the DGM can autonomously achieve performance on par with openly available
484 solutions. However, it still falls short of closed-source SoTA SWE-bench solutions. An open question
485 is whether running the DGM for longer would continue to yield performance gains and eventually
486 surpass closed-source solutions. These closed-source solutions often rely on elaborately handcrafted

486 techniques developed by teams of highly skilled experts. Since FMs have yet to match the capabilities
487 of such experts (e.g., in reasoning), the DGM currently requires extensive compute to discover
488 improvements. A single run of the DGM on SWE-bench, as presented in Section 4, takes about 2
489 weeks and incurs significant API costs (Appendix E.1). We hypothesize that further progress will
490 require more efficient use of computational resources and the development of better reasoning skills.
491

492 Since this version of the DGM is mainly powered by FMs, it is inherently limited by the capabilities
493 of the underlying FM. Hence, an exciting future direction is to extend self-modification beyond just
494 prompts or FM workflows, to include more computationally intensive methods, such as rewriting
495 its own training script to update the FM itself. While this version of the DGM focuses on coding,
496 AI systems are increasingly applied across a wide range of domains (e.g., computer vision, creative
497 writing). Another promising extension is to develop self-improving AI systems capable of enhancing
498 themselves beyond just the coding domain. A key assumption in this work is that coding benchmarks
499 are a good reflection of the agent’s ability to self-improve, since the self-modification task requires
500 the agent to modify its own codebase. However, one could envision an alternative approach that
501 co-evolves the target task distribution (Faldor et al., 2025; Wang et al., 2023c), thereby removing
502 the constraint of self-improvement being tied to a single objective, as in true open-ended processes.
503 Appendix J presents additional potential directions for future work. As we continue to explore this
504 powerful technology, we must also keep safety front and center, as discussed in Section 5.
505

506 In conclusion, the DGM represents a significant step toward the automation of AI development
507 through self-improving systems capable of editing their own codebase. While current limitations in
508 compute and reasoning constrain its full potential, continued advances in FMs and infrastructure may
509 unlock more powerful and general-purpose self-improvements. Provided that the safety concerns
510 are carefully navigated (Section 5), the future of self-improving AI systems and AI-Generating
511 Algorithms (Clune, 2019) holds immense promise to open-endedly evolve AI, continually rewriting
512 or retraining itself in pursuit of greater capabilities aligned with human values.
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539

540
541
ETHICS STATEMENT542
543
544
545
We affirm compliance with the ICLR Code of Ethics. This work studies self-improving AI systems
in the limited context of code-editing agents evaluated on standard programming benchmarks. No
human subjects were involved and no personally identifiable information (PII) was collected or
processed; IRB approval was therefore not required.546
547
548
549
550
551
552
553
554
Safety and misuse. Self-modifying systems can pose safety risks if allowed to act without constraints
or if optimizations inadvertently introduce unsafe behaviors. To mitigate this, all agents in our
experiments ran inside isolated sandboxes with strict resource and time limits; agents had limited
network access and no ability to modify the host environment. The self-improvement scope was
restricted to the agent’s own Python codebase and evaluation harnesses. We maintained a complete,
auditable lineage (archive) of code changes and evaluations, enabling rollback and post-hoc analysis.
We did not deploy discovered agents in real development environments. Our release plan (code,
prompts, and evaluation artifacts) will exclude any components that grant elevated system access and
will include default sandboxing, guardrails, and clear documentation of intended use.555
556
557
558
559
560
561
562
563
Dual-use, downstream impact, and limitations. Stronger autonomous coding agents could be
dual-use (e.g., aiding software maintenance, but also potentially facilitating creation of harmful code
if misapplied). We believe the research benefits (e.g., advancing methods for controlled, auditable
self-improvement and demonstrating practical safeguards) outweigh the risks. Nevertheless, we
explicitly discourage security-sensitive or unsandboxed deployment and provide concrete safety
recommendations (Section 5). Our empirical focus on benchmark optimization may not capture
all desirable properties (robustness, interpretability, or broader social values). We therefore treat
benchmark gains as necessary but insufficient indicators of general AI development, and discuss
avenues to integrate other objectives (e.g., safety, reasoning) into the optimization loop.564
565
566
567
568
569
Data governance, IP, and licensing. We evaluate on SWE-bench Verified and Polyglot, which are
composed of open-source repositories and tasks. We complied with dataset licenses and usage terms
to the best of our knowledge. We did not introduce or distribute proprietary code. Foundation models
(FMs) were accessed via provider APIs under their terms of service; we did not submit sensitive data,
nor attempt to circumvent usage policies. Logs released with this work will be scrubbed of API keys
and any incidental sensitive strings.570
571
572
573
574
Bias, fairness, and equity. Although our domain is software code rather than human-centered text,
FM behavior can still reflect biases (e.g., language or ecosystem preferences) and may unevenly
benefit communities whose tooling is better represented in training data. We partially address this by
evaluating across multiple languages (Polyglot) and reporting cross-benchmark transfer. Future work
should add diagnostics for biased failure modes and include broader, community-driven task sets.575
576
577
578
Conflicts of interest and funding. No author has a financial interest in products whose performance
is evaluated here. Sponsors and employers did not influence experimental design, analysis, or the
decision to publish, beyond providing salary or standard research support. Any external compute or
API credits are acknowledged in the appendix.580
581
REPRODUCIBILITY STATEMENT582
583
584
585
586
587
588
589
590
We will open-source all code and full agent logs, including the complete archive lineage of self-
modifications (diffs, prompts, and configs) as well as the evaluation harness. To support exact repli-
cation, we reference the following: algorithmic details and pseudocode (Section 3, Appendix C.4);
parent selection and open-ended exploration settings (Appendix C.2); foundation model choices
and hyperparameters (Appendix D.1); benchmark task subsets for SWE-bench and Polyglot (Ap-
pendix E.2, Appendix E.3); staged evaluation protocols and scripts (Section 4.2); implementations
and diffs for the best discovered agents (Appendix F.1, Appendix F.2); and compute and cost estimates
(Appendix E.1). The released code repository will include environment specifications and scripts to
reproduce all results, figures, and tables.591
592
593

594 REFERENCES
595

596 Fuma Aki, Riku Ikeda, Takumi Saito, Ciaran Regan, and Mizuki Oka. Llm-poet: Evolving complex
597 environments using large language models. In *Proceedings of the Genetic and Evolutionary
598 Computation Conference Companion*, pp. 243–246, 2024.

599 Rajeev Alur, Rishabh Singh, Dana Fisman, and Armando Solar-Lezama. Search-based program
600 synthesis. *Communications of the ACM*, 61(12):84–93, 2018.

601 S-I Amari. Learning patterns and pattern sequences by self-organizing nets of threshold elements.
602 *IEEE Transactions on computers*, 100(11):1197–1206, 1972.

603 Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob
604 McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay.
605 *Advances in neural information processing systems*, 30, 2017.

606 Anthropic. Claude 3.5 Sonnet. <https://www.anthropic.com/news/claude-3-5-sonnet>, June 2024a. [Accessed 17 April 2025].

607 Anthropic. Claude can now use tools, May 2024b. URL <https://www.anthropic.com/news/tool-use-ga>. Accessed: 2025-05-03.

608 Anthropic. Claude 3.7 sonnet and claude code, February 2025. URL <https://www.anthropic.com/news/claude-3-7-sonnet>. Accessed: 2025-05-06.

609 Usman Anwar, Abulhair Saparov, Javier Rando, Daniel Paleka, Miles Turpin, Peter Hase,
610 Ekdeep Singh Lubana, Erik Jenner, Stephen Casper, Oliver Sourbut, et al. Foundational challenges
611 in assuring alignment and safety of large language models. *arXiv preprint arXiv:2404.09932*,
612 2024.

613 Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. Neural machine translation by jointly
614 learning to align and translate. In *International Conference on Learning Representations*, 2015.

615 Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones,
616 Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional AI:
617 Harmlessness from AI feedback. *arXiv preprint arXiv:2212.08073*, 2022.

618 Adrien Baranes and Pierre-Yves Oudeyer. Active learning of inverse models with intrinsically
619 motivated goal exploration in robots. *Robotics and Autonomous Systems*, 61(1):49–73, 2013.

620 Shraddha Barke, Emmanuel Anaya Gonzalez, Saketh Ram Kasibatla, Taylor Berg-Kirkpatrick, and
621 Nadia Polikarpova. Hysynth: Context-free llm approximation for guiding program synthesis.
622 *Advances in Neural Information Processing Systems*, 37:15612–15645, 2024.

623 Joshua Bengio, Geoffrey Hinton, Andrew Yao, Dawn Song, Pieter Abbeel, Trevor Darrell, Yu-
624 val Noah Harari, Ya-Qin Zhang, Lan Xue, Shai Shalev-Shwartz, et al. Managing extreme AI risks
625 amid rapid progress. *Science*, 384(6698):842–845, 2024.

626 N Bostrom. Existential Risks: analyzing human extinction scenarios and related hazards. *Journal of
627 Evolution and Technology*, 9, 2002.

628 Nick Bostrom. Ethical issues in advanced artificial intelligence. *Machine Ethics and Robot Ethics*,
629 pp. 69–75, 2020.

630 Herbie Bradley, Andrew Dai, Hannah Benita Teufel, Jenny Zhang, Koen Oostermeijer, Marco
631 Bellagente, Jeff Clune, Kenneth Stanley, Gregory Schott, and Joel Lehman. Quality-diversity
632 through ai feedback. In *The Twelfth International Conference on Learning Representations*, 2024.

633 Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal,
634 Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are
635 few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

636 Jake Bruce, Michael D Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes,
637 Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, et al. Genie: Generative
638 interactive environments. In *Forty-first International Conference on Machine Learning*, 2024.

648 J Richard Buchi and Lawrence H Landweber. Solving sequential conditions by finite-state strategies.
 649 In *The collected works of J. Richard Büchi*, pp. 525–541. Springer, 1990.
 650

651 Ruisheng Cao, Fangyu Lei, Haoyuan Wu, Jixuan Chen, Yeqiao Fu, Hongcheng Gao, Xinzhuang
 652 Xiong, Hanchong Zhang, Wenjing Hu, Yuchen Mao, et al. Spider2-v: How far are multimodal
 653 agents from automating data science and engineering workflows? *Advances in Neural Information
 654 Processing Systems*, 37:107703–107744, 2024.

655 Konstantinos Chatzilygeroudis, Antoine Cully, Vassilis Vassiliades, and Jean-Baptiste Mouret.
 656 Quality-diversity optimization: a novel branch of stochastic optimization. In *Black Box Op-
 657 timization, Machine Learning, and No-Free Lunch Theorems*, pp. 109–135. Springer, 2021.
 658

659 Banghao Chen, Zhaofeng Zhang, Nicolas Langrené, and Shengxin Zhu. Unleashing the poten-
 660 tial of prompt engineering in large language models: a comprehensive review. *arXiv preprint
 661 arXiv:2310.14735*, 2023.

662 Ching-An Cheng, Allen Nie, and Adith Swaminathan. Trace is the next autodiff: Generative
 663 optimization with rich feedback, execution traces, and llms. *Advances in Neural Information
 664 Processing Systems*, 37:71596–71642, 2024.
 665

666 Jeff Clune. AI-GAs: AI-generating algorithms, an alternate paradigm for producing general artificial
 667 intelligence. *arXiv preprint arXiv:1905.10985*, 2019.

668 Cédric Colas, Pierre Fournier, Mohamed Chetouani, Olivier Sigaud, and Pierre-Yves Oudeyer.
 669 Curious: intrinsically motivated modular multi-goal reinforcement learning. In *International
 670 conference on machine learning*, pp. 1331–1340. PMLR, 2019.
 671

672 Cédric Colas, Tristan Karch, Clément Moulin-Frier, and Pierre-Yves Oudeyer. Language and culture
 673 internalization for human-like autotelic AI. *Nature Machine Intelligence*, 4(12):1068–1076, 2022a.

674 Cédric Colas, Tristan Karch, Olivier Sigaud, and Pierre-Yves Oudeyer. Autotelic agents with
 675 intrinsically motivated goal-conditioned reinforcement learning: a short survey. *Journal of
 676 Artificial Intelligence Research*, 74:1159–1199, 2022b.
 677

678 Cédric Colas, Laetitia Teodorescu, Pierre-Yves Oudeyer, Xingdi Yuan, and Marc-Alexandre Côté.
 679 Augmenting autotelic agents with large language models. In *Conference on Lifelong Learning
 680 Agents*, pp. 205–226. PMLR, 2023.

681 Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International
 682 conference on computers and games*, pp. 72–83. Springer, 2006.
 683

684 Charles Darwin. Origin of the species. In *British Politics and the environment in the long nineteenth
 685 century*, pp. 47–55. Routledge, 2023.

686 Richard Dawkins. The evolution of evolvability. In *Artificial life*, pp. 201–220. Routledge, 2019.

687 Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch,
 688 and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment
 689 design. *Advances in neural information processing systems*, 33:13049–13061, 2020.
 690

691 Aaron Dharna, Cong Lu, and Jeff Clune. Quality-Diversity Self-Play: Open-Ended Strategy Innova-
 692 tion via Foundation Models. In *NeurIPS 2024 Workshop on Open-World Agents*, 2024.

693

694 Li Ding, Jenny Zhang, Jeff Clune, Lee Spector, and Joel Lehman. Quality diversity through
 695 human feedback: towards open-ended diversity-driven optimization. In *Proceedings of the 41st
 696 International Conference on Machine Learning*, pp. 11072–11090, 2024.
 697

698 Theodosius Dobzhansky. *Genetics of the evolutionary process*, volume 139. Columbia University
 699 Press, 1970.
 700

701 Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Go-explore: a new
 702 approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*, 2019.

702 Adrien Ecoffet, Jeff Clune, and Joel Lehman. Open questions in creating safe open-ended AI: tensions
 703 between control and creativity. In *Artificial Life Conference Proceedings 32*, pp. 27–35. MIT Press
 704 One Rogers Street, Cambridge, MA 02142-1209, USA journals-info ..., 2020.

705 Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. First return, then
 706 explore. *Nature*, 590(7847):580–586, 2021.

708 Anthony W Fisher Edwards. The genetical theory of natural selection. *Genetics*, 154(4):1419–1426,
 709 2000.

710 Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc
 711 Cary, Armando Solar-Lezama, and Joshua B Tenenbaum. Dreamcoder: Bootstrapping inductive
 712 program synthesis with wake-sleep library learning. In *Proceedings of the 42nd ACM SIGPLAN
 713 International Conference on Programming Language Design and Implementation*, pp. 835–850,
 714 2021.

715 Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need:
 716 Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

718 Meta Fundamental AI Research Diplomacy Team (FAIR)†, Anton Bakhtin, Noam Brown, Emily
 719 Dinan, Gabriele Farina, Colin Flaherty, Daniel Fried, Andrew Goff, Jonathan Gray, Hengyuan Hu,
 720 et al. Human-level play in the game of Diplomacy by combining language models with strategic
 721 reasoning. *Science*, 378(6624):1067–1074, 2022.

722 Maxence Faldor, Jenny Zhang, Antoine Cully, and Jeff Clune. OMNI-EPIC: Open-endedness
 723 via Models of human Notions of Interestingness with Environments Programmed in Code. In
 724 *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=Y1XkzMJpPd>.

727 Chrisantha Fernando, Dylan Sunil Banarse, Henryk Michalewski, Simon Osindero, and Tim Rock-
 728 täschel. Promptbreeder: Self-Referential Self-Improvement via Prompt Evolution. In *Forty-first
 729 International Conference on Machine Learning*, 2024.

730 Deep Ganguli, Liane Lovitt, Jackson Kernion, Amanda Askell, Yuntao Bai, Saurav Kadavath, Ben
 731 Mann, Ethan Perez, Nicholas Schiefer, Kamal Ndousse, et al. Red teaming language models to
 732 reduce harms: Methods, scaling behaviors, and lessons learned. *arXiv preprint arXiv:2209.07858*,
 733 2022.

734 Hongcheng Gao, Yue Liu, Yufei He, Longxu Dou, Chao Du, Zhijie Deng, Bryan Hooi, Min
 735 Lin, and Tianyu Pang. Flowreasoner: Reinforcing query-level meta-agents. *arXiv preprint
 736 arXiv:2504.15257*, 2025.

738 Paul Gauthier. Aider: Ai pair programming in your terminal. <https://github.com/Aider-AI/aider>, 2024. Accessed: 2025-05-14.

741 Loris Gaven, Thomas Carta, Clément Romac, Cédric Colas, Sylvain Lamprier, Olivier Sigaud, and
 742 Pierre-Yves Oudeyer. MAGELLAN: Metacognitive predictions of learning progress guide autotelic
 743 LLM agents in large goal spaces. *arXiv preprint arXiv:2502.07709*, 2025.

744 John Gerhart and Marc Kirschner. The theory of facilitated variation. *Proceedings of the National
 745 Academy of Sciences*, 104(suppl_1):8582–8589, 2007.

747 Irving John Good. Speculations concerning the first ultraintelligent machine. In *Advances in
 748 computers*, volume 6, pp. 31–88. Elsevier, 1966.

749 Google DeepMind. Gemini model “thinking” updates — march
 750 2025. [https://blog.google/technology/google-deepmind/
 751 gemini-model-thinking-updates-march-2025/#gemini-2-5-thinking](https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/#gemini-2-5-thinking),
 752 March 2025. Accessed: 2025-05-11.

754 Ryan Greenblatt, Carson Denison, Benjamin Wright, Fabien Roger, Monte MacDiarmid, Sam Marks,
 755 Johannes Treutlein, Tim Belonax, Jack Chen, David Duvenaud, et al. Alignment faking in large
 language models. *arXiv preprint arXiv:2412.14093*, 2024.

756 Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. *ACM*
 757 *Sigplan Notices*, 46(1):317–330, 2011.

758
 759 Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu,
 760 Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms
 761 via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

762 John Storrs Hall. Self-improving AI: An analysis. *Minds and Machines*, 17(3):249–259, 2007.

763
 764 Alex Havrilla, Andrew Dai, Laura O’Mahony, Koen Oostermeijer, Vera Zisler, Alon Albalak, Fabrizio
 765 Milo, Sharath Chandra Raparthy, Kanishk Gandhi, Baber Abbasi, et al. Surveying the effects of
 766 quality, diversity, and complexity in synthetic data from large language models. *arXiv preprint*
 767 *arXiv:2412.02980*, 2024a.

768 Alex Havrilla, Sharath Raparthy, Christoforus Nalmpantis, Jane Dwivedi-Yu, Maksym Zhuravinskyi,
 769 Eric Hambro, and Roberta Raileanu. Glore: When, where, and how to improve llm reasoning via
 770 global and local refinements. *arXiv preprint arXiv:2402.10963*, 2024b.

771 Jesse Love Hendrikse, Trish Elizabeth Parsons, and Benedikt Hallgrímsson. Evolvability as the
 772 proper focus of evolutionary developmental biology. *Evolution & development*, 9(4):393–401,
 773 2007.

774 Nathan Herr, Tim Rocktäschel, and Roberta Raileanu. Llm-first search: Self-guided exploration of
 775 the solution space. *arXiv preprint arXiv:2506.05213*, 2025.

776
 777 Marius Hobbhahn. Swe-bench verified mini. <https://github.com/mariushobbhahn/SWEBenchmark-verified-mini>, April 2025. Accessed: 2025-04-16.

778
 779 John J Hopfield. Neural networks and physical systems with emergent collective computational
 780 abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

781
 782 Shengran Hu and Jeff Clune. Thought Cloning: Learning to think while acting by imitating human
 783 thinking. *Advances in Neural Information Processing Systems*, 36, 2024.

784
 785 Shengran Hu, Cong Lu, and Jeff Clune. Automated Design of Agentic Systems. In *The Thirteenth*
 786 *International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=t9U3LW7JXV>.

787
 788 Yue Hu, Yuzhu Cai, Yixin Du, Xinyu Zhu, Xiangrui Liu, Zijie Yu, Yuchen Hou, Shuo Tang, and
 789 Siheng Chen. Self-evolving multi-agent collaboration networks for software development. *arXiv*
 790 *preprint arXiv:2410.16946*, 2024.

791 Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuxin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han.
 792 Large language models can self-improve. *arXiv preprint arXiv:2210.11610*, 2022.

793
 794 Edward Hughes, Michael Dennis, Jack Parker-Holder, Feryal Behbahani, Aditi Mavalankar, Yuge
 795 Shi, Tom Schaul, and Tim Rocktaschel. Open-endedness is essential for artificial superhuman
 796 intelligence. *arXiv preprint arXiv:2406.04268*, 2024.

797 Kazuki Irie, Imanol Schlag, Róbert Csordás, and Jürgen Schmidhuber. A modern self-referential
 798 weight matrix that learns to modify itself. In *International Conference on Machine Learning*, pp.
 799 9660–9677. PMLR, 2022.

800
 801 Kazuki Irie, Róbert Csordás, and Jürgen Schmidhuber. Metalearning continual learning algorithms.
 802 *Transactions on Machine Learning Research*, 2025.

803 Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali
 804 Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training
 805 of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

806
 807 Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. In *International*
 808 *Conference on Machine Learning*, pp. 4940–4950. PMLR, 2021.

809
 810 Minqi Jiang, Tim Rocktäschel, and Edward Grefenstette. General intelligence requires rethinking
 811 exploration. *Royal Society Open Science*, 10(6):230539, 2023.

810 Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and
 811 Karthik R Narasimhan. SWE-bench: Can Language Models Resolve Real-world Github Is-
 812 sues? In *The Twelfth International Conference on Learning Representations*, 2024. URL
 813 <https://openreview.net/forum?id=VTF8yNQM66>.

814 Ingmar Kanitscheider, Joost Huizinga, David Farhi, William Heben Guss, Brandon Houghton, Raul
 815 Sampedro, Peter Zhokhov, Bowen Baker, Adrien Ecoffet, Jie Tang, Oleg Klimov, and Jeff Clune.
 816 Multi-task curriculum learning in a complex, visual, hard-exploration domain: Minecraft. *arXiv*
 817 preprint *arXiv:2106.14876*, 2021.

818 Akbir Khan, John Hughes, Dan Valentine, Laura Ruis, Kshitij Sachan, Ansh Radhakrishnan, Edward
 819 Grefenstette, Samuel R Bowman, Tim Rocktäschel, and Ethan Perez. Debating with more
 820 persuasive llms leads to more truthful answers. *arXiv preprint arXiv:2402.06782*, 2024.

822 Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vard-
 823 hamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. Dspy: Compiling
 824 declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*,
 825 2023.

826 Yoon Kim, Carl Denton, Luong Hoang, and Alexander M Rush. Structured Attention Networks. In
 827 *International Conference on Learning Representations*, 2017.

828 Motoo Kimura. The neutral theory of molecular evolution. *Scientific American*, 241(5):98–129,
 829 1979.

831 Louis Kirsch and Jürgen Schmidhuber. Self-referential meta learning. In *First Conference on
 832 Automated Machine Learning (Late-Breaking Workshop)*, 2022.

833 Martin Klissarov, Pierluca D’Oro, Shagun Sodhani, Roberta Raileanu, Pierre-Luc Bacon, Pascal
 834 Vincent, Amy Zhang, and Mikael Henaff. Motif: Intrinsic motivation from artificial intelligence
 835 feedback. *arXiv preprint arXiv:2310.00166*, 2023.

837 Martin Klissarov, Mikael Henaff, Roberta Raileanu, Shagun Sodhani, Pascal Vincent, Amy Zhang,
 838 Pierre-Luc Bacon, Doina Precup, Marlos C Machado, and Pierluca D’Oro. MaestroMotif: Skill
 839 Design from Artificial Intelligence Feedback. *arXiv preprint arXiv:2412.08542*, 2024.

840 Varun Raj Kompella, Marijn Stollenga, Matthew Luciw, and Juergen Schmidhuber. Continual
 841 curiosity-driven skill acquisition from high-dimensional video inputs for humanoid robots. *Artificial
 842 Intelligence*, 247:313–335, 2017.

843 Robert Lange, Tom Schaul, Yutian Chen, Tom Zahavy, Valentin Dalibard, Chris Lu, Satinder Singh,
 844 and Sebastian Flennerhag. Discovering evolution strategies via meta-black-box optimization. In
 845 *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, pp. 29–30,
 846 2023.

848 Robert Lange, Yingtao Tian, and Yujin Tang. Large language models as evolution strategies. In
 849 *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 579–582,
 850 2024.

851 Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning
 852 quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47):eabc5986, 2020.

854 Joel Lehman. Machine love. *arXiv preprint arXiv:2302.09248*, 2023.

855 Joel Lehman and Kenneth O Stanley. Novelty search and the problem with objectives. *Genetic
 856 programming theory and practice IX*, pp. 37–56, 2011.

857 Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O Stanley.
 858 Evolution through large models. In *Handbook of Evolutionary Machine Learning*, pp. 331–366.
 859 Springer, 2023.

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

864 J. Li, Storie J., and J. Clune. Encouraging creative thinking in robots improves their ability to solve
 865 challenging problems. In *Proceedings of the Genetic and Evolutionary Computation Conference*,
 866 pp. 193–200, 2014.

867 Yixuan Li, Julian Parsert, and Elizabeth Polgreen. Guiding enumerative program synthesis with
 868 large language models. In *International Conference on Computer Aided Verification*, pp. 280–301.
 869 Springer, 2024.

870 Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi,
 871 and Zhaopeng Tu. Encouraging divergent thinking in large language models through multi-agent
 872 debate. *arXiv preprint arXiv:2305.19118*, 2023.

873 Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan
 874 Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s verify step by step. In *The Twelfth
 875 International Conference on Learning Representations*, 2023.

876 Bryan Lim, Manon Flageat, and Antoine Cully. Large language models as in-context ai generators
 877 for quality-diversity. In *ALIFE 2024: Proceedings of the 2024 Artificial Life Conference*. MIT
 878 Press, 2024.

879 Seppo Linnainmaa. *The representation of the cumulative rounding error of an algorithm as a Taylor
 880 expansion of the local rounding errors*. PhD thesis, Master’s Thesis (in Finnish), Univ. Helsinki,
 881 1970.

882 Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu
 883 Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language
 884 model. *arXiv preprint arXiv:2401.02051*, 2024.

885 Lei Liu, Xiaoyan Yang, Yue Shen, Binbin Hu, Zhiqiang Zhang, Jinjie Gu, and Guannan Zhang.
 886 Think-in-memory: Recalling and post-thinking enable llms with long-term memory. *arXiv preprint
 887 arXiv:2311.08719*, 2023.

888 Chris Lu, Sebastian Towers, and Jakob Foerster. Arbitrary order meta-learning with simple population-
 889 based evolution. In *Artificial Life Conference Proceedings 35*, volume 2023, pp. 67. MIT Press
 890 One Rogers Street, Cambridge, MA 02142-1209, USA journals-info . . . , 2023.

891 Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist:
 892 Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024a.

893 Cong Lu, Shengran Hu, and Jeff Clune. Intelligent go-explore: Standing on the shoulders of giant
 894 foundation models. *arXiv preprint arXiv:2405.15143*, 2024b.

895 Cong Lu, Shengran Hu, and Jeff Clune. Automated capability discovery via model self-exploration.
 896 *arXiv preprint arXiv:2502.0757*, 2025.

897 Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman,
 898 Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding
 899 large language models. *arXiv preprint arXiv:2310.12931*, 2023.

900 Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri
 901 Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement
 902 with self-feedback, 2023. URL <https://arxiv.org/abs/2303.17651>, 2023.

903 John Markoff. *Machines of loving grace: The quest for common ground between humans and robots*.
 904 HarperCollins Publishers, 2016.

905 Ernst Mayr. *The growth of biological thought: Diversity, evolution, and inheritance*. Harvard
 906 University Press, 1982.

907 Luke Metz, C Daniel Freeman, Niru Maheswaranathan, and Jascha Sohl-Dickstein. Training learned
 908 optimizers with randomly initialized learned optimizers. *arXiv preprint arXiv:2101.07367*, 2021.

909 Ali Modarressi, Ayyoob Imani, Mohsen Fayyaz, and Hinrich Schütze. Ret-llm: Towards a general
 910 read-write memory for large language models. *arXiv preprint arXiv:2305.14322*, 2023.

918 Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint*
919 *arXiv:1504.04909*, 2015.

920

921 Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke
922 Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time
923 scaling. *arXiv preprint arXiv:2501.19393*, 2025.

924

925 Muhammad U Nasir and Julian Togelius. Practical PCG through large language models. In *2023*
926 *IEEE Conference on Games (CoG)*, pp. 1–4. IEEE, 2023.

927

928 Muhammad U Nasir, Steven James, and Julian Togelius. Word2world: Generating stories and worlds
929 through large language models. *arXiv preprint arXiv:2405.06686*, 2024.

930

931 Anh Mai Nguyen, Jason Yosinski, and Jeff Clune. Innovation engines: Automated creativity and
932 improved stochastic optimization via deep learning. In *Proceedings of the 2015 annual conference*
933 *on genetic and evolutionary computation*, pp. 959–966, 2015.

934

935 Fan Nie, Lan Feng, Haotian Ye, Weixin Liang, Pan Lu, Huaxiu Yao, Alexandre Alahi, and James
936 Zou. Weak-for-strong: Training weak meta-agent to harness strong executors. *arXiv preprint*
937 *arXiv:2504.04785*, 2025.

938

939 Boye Niu, Yiliao Song, Kai Lian, Yifan Shen, Yu Yao, Kun Zhang, and Tongliang Liu. Flow: Modularized
940 agentic workflow automation. In *The Thirteenth International Conference on Learning*
941 *Representations*, 2025.

942

943 Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wag-
944 ner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan
945 Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Push-
946 meet Kohli, and Matej Balog. Alphaevolve: A coding agent for scientific and algorithmic discovery.
947 Technical report, Google DeepMind, 2025.

948

949 OpenAI. Introducing swe-bench verified. <https://openai.com/index/introducing-swe-bench-verified/>, August 2024. Accessed: 2025-04-16.

950

951 OpenAI. OpenAI o3-mini. <https://openai.com/index/openai-o3-mini/>, January
952 2025. Accessed: 2025-05-01.

953

954 Pierre-Yves Oudeyer, Frdric Kaplan, and Verena V Hafner. Intrinsic motivation systems for au-
955 tonomous mental development. *IEEE transactions on evolutionary computation*, 11(2):265–286,
956 2007.

957

958 Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong
959 Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow
960 instructions with human feedback. *Advances in neural information processing systems*, 35:27730–
961 27744, 2022.

962

963 Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A Decomposable Attention
964 Model for Natural Language Inference. In *Proceedings of the 2016 Conference on Empirical*
965 *Methods in Natural Language Processing*, pp. 2249–2255, 2016.

966

967 Jack Parker-Holder, Philip Ball, Jake Bruce, Vibhavari Dasagi, Kristian Holsheimer, Chris-
968 tos Kaplanis, Alexandre Moufarek, Guy Scully, Jeremy Shar, Jimmy Shi, Stephen Spencer,
969 Jessica Yung, Michael Dennis, Sultan Kenjeyev, Shangbang Long, Vlad Mnih, Harris
970 Chan, Maxime Gazeau, Bonnie Li, Fabio Pardo, Luyu Wang, Lei Zhang, Frederic Besse,
971 Tim Harley, Anna Mitenkova, Jane Wang, Jeff Clune, Demis Hassabis, Raia Hadsell,
972 Adrian Bolton, Satinder Singh, and Tim Rocktäschel. Genie 2: A large-scale foun-
973 dation world model, 2024. URL <https://deepmind.google/discover/blog/genie-2-a-large-scale-foundation-world-model/>.

974

975 Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration
976 by self-supervised prediction. In *International conference on machine learning*, pp. 2778–2787.
977 PMLR, 2017.

972 Paul Gauthier. o1 tops aider’s new polyglot leaderboard. <https://aider.chat/2024/12/21/polyglot.html>, December 2024. Accessed: 2025-04-16.

973

974

975 Oleksandr Polozov and Sumit Gulwani. Flashmeta: A framework for inductive program synthesis. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pp. 107–126, 2015.

976

977

978 Justin K Pugh, Lisa B Soros, and Kenneth O Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40, 2016.

979

980

981 Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

982

983 Maxime Robeys, Martin Szummer, and Laurence Aitchison. A Self-Improving Coding Agent. *arXiv preprint arXiv:2504.15228*, 2025.

984

985

986 Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

987

988

989

990 J Rosser and Jakob Nicolaus Foerster. Agentbreeder: Mitigating the AI safety impact of multi-agent scaffolds via self-improvement. In *Scaling Self-Improving Foundation Models without Human Supervision*, 2025. URL <https://openreview.net/forum?id=j0n3BJJTct>.

991

992

993 David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning internal representations by error propagation, 1985.

994

995

996 Mikayel Samvelyan, Sharath Chandra Raparthy, Andrei Lupu, Eric Hambro, Aram Markosyan, Manish Bhatt, Yuning Mao, Minqi Jiang, Jack Parker-Holder, Jakob Foerster, et al. Rainbow teaming: Open-ended generation of diverse adversarial prompts. *Advances in Neural Information Processing Systems*, 37:69747–69786, 2024.

997

998

999

1000 Cansu Sancaktar, Christian Gumbsch, Andrii Zadaianchuk, Pavel Kolev, and Georg Martius. SENSEI: Semantic Exploration Guided by Foundation Models to Learn Versatile World Models. *arXiv preprint arXiv:2503.01584*, 2025.

1001

1002

1003 Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International conference on machine learning*, pp. 1312–1320. PMLR, 2015.

1004

1005

1006 Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551, 2023.

1007

1008

1009

1010 Jürgen Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

1011

1012

1013 Jürgen Schmidhuber. A ‘self-referential’ weight matrix. In *International conference on artificial neural networks*, pp. 446–450. Springer, 1993.

1014

1015 Jürgen Schmidhuber. Gödel machines: Fully self-referential optimal universal self-improvers. In *Artificial general intelligence*, pp. 199–226. Springer, 2007.

1016

1017

1018 Jürgen Schmidhuber. Driven by compression progress: A simple principle explains essential aspects of subjective beauty, novelty, surprise, interestingness, attention, curiosity, creativity, art, science, music, jokes. In *Workshop on anticipatory behavior in adaptive learning systems*, pp. 48–76. Springer, 2008.

1019

1020

1021

1022 Jürgen Schmidhuber. Powerplay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem. *Frontiers in psychology*, 4:313, 2013.

1023

1024

1025 Jürgen Schmidhuber and Rudolf Huber. *Learning to generate focus trajectories for attentive vision*. Institut für Informatik, 1990.

1026 Sander Schulhoff, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei
 1027 Si, Yinheng Li, Aayush Gupta, HyoJung Han, Sevien Schulhoff, et al. The prompt report: A
 1028 systematic survey of prompting techniques. *arXiv preprint arXiv:2406.06608*, 2024.

1029 Ivaxi Sheth, Jan Wehner, Sahar Abdelnabi, Ruta Binkyte, and Mario Fritz. Safety is Essential for
 1030 Responsible Open-Ended Systems. *arXiv preprint arXiv:2502.04512*, 2025.

1032 Kensen Shi, Hanjun Dai, Wen-Ding Li, Kevin Ellis, and Charles Sutton. Lambdabeam: Neural pro-
 1033 gram search with higher-order functions and lambdas. *Advances in Neural Information Processing*
 1034 *Systems*, 36:51327–51346, 2023.

1035 Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion:
 1036 Language agents with verbal reinforcement learning. *Advances in Neural Information Processing*
 1037 *Systems*, 36:8634–8652, 2023.

1039 David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche,
 1040 Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering
 1041 the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

1042 David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez,
 1043 Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi
 1044 by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*,
 1045 2017.

1046 Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J
 1047 Liu, James Harrison, Jaehoon Lee, Kelvin Xu, et al. Beyond human data: Scaling self-training for
 1048 problem-solving with language models. *arXiv preprint arXiv:2312.06585*, 2023.

1049 Joar Skalse, Nikolaus Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing
 1050 reward gaming. *Advances in Neural Information Processing Systems*, 35:9460–9471, 2022.

1051 Kenneth O Stanley and Joel Lehman. *Why greatness cannot be planned: The myth of the objective*.
 1052 Springer, 2015.

1053 Kenneth O Stanley, Joel Lehman, and Lisa Soros. Open-endedness: The last grand challenge you've
 1054 never heard of. *While open-endedness could be a force for discovering intelligence, it could also*
 1055 *be a component of AI itself*, 2017.

1056 Marilyn Strathern. ‘Improving ratings’: audit in the British University system. *European review*, 5
 1057 (3):305–321, 1997.

1058 Jinwei Su, Yinghui Xia, Ronghua Shi, Jianhui Wang, Jianuo Huang, Yijin Wang, Tianyu Shi, Yang
 1059 Jingsong, and Lewei He. Debboxflow: Automating agent creation via agent debate. *arXiv preprint*
 1060 *arXiv:2503.23781*, 2025.

1061 Shyam Sudhakaran, Miguel González-Duque, Matthias Freiberger, Claire Ghanous, Elias Najarro,
 1062 and Sebastian Risi. Mariogpt: Open-ended text2level generation through large language models.
 1063 *Advances in Neural Information Processing Systems*, 36:54213–54227, 2023.

1064 OpenAI Team, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden
 1065 Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card.
 1066 *arXiv preprint arXiv:2412.16720*, 2024.

1067 Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz
 1068 Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing*
 1069 *systems*, 30, 2017.

1070 Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and
 1071 Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv*
 1072 *preprint arXiv:2305.16291*, 2023a.

1073 Ren-Jian Wang, Ke Xue, Yutong Wang, Peng Yang, Haobo Fu, Qiang Fu, and Chao Qian. Diversity
 1074 from human feedback. *arXiv preprint arXiv:2310.06648*, 2023b.

1080 Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O Stanley. Paired open-ended trailblazer (poet):
 1081 Endlessly generating increasingly complex and diverse learning environments and their solutions.
 1082 *arXiv preprint arXiv:1901.01753*, 2019.

1083 Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan,
 1084 Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software develop-
 1085 ers as generalist agents. In *The Thirteenth International Conference on Learning Representations*,
 1086 2024.

1087 Yufei Wang, Zhou Xian, Feng Chen, Tsun-Hsuan Wang, Yian Wang, Katerina Fragkiadaki, Zackory
 1088 Erickson, David Held, and Chuang Gan. Robogen: Towards unleashing infinite data for automated
 1089 robot learning via generative simulation. *arXiv preprint arXiv:2311.01455*, 2023c.

1090 Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny
 1091 Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in
 1092 neural information processing systems*, 35:24824–24837, 2022.

1093 Marco Wiering and Jürgen Schmidhuber. Hq-learning. *Adaptive behavior*, 6(2):219–246, 1997.

1094 S Wright. The roles of mutation, inbreeding, crossbreeding and selection in evolution, proceedings of
 1095 the sixth international congress of genetics. proc sixth int congr genet [internet]. New York356366,
 1096 1932.

1097 Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying
 1098 llm-based software engineering agents. *arXiv preprint arXiv:2407.01489*, 2024.

1099 John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan,
 1100 and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering.
 1101 In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL
 1102 <https://arxiv.org/abs/2405.15793>.

1103 Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao.
 1104 React: Synergizing reasoning and acting in language models. In *International Conference on
 1105 Learning Representations (ICLR)*, 2023.

1106 Rui Ye, Shuo Tang, Rui Ge, Yixin Du, Zhenfei Yin, Siheng Chen, and Jing Shao. Mas-gpt: Training
 1107 llms to build llm-based multi-agent systems. *arXiv preprint arXiv:2503.03686*, 2025.

1108 Xunjian Yin, Xinyi Wang, Liangming Pan, Xiaojun Wan, and William Yang Wang. G\"odel
 1109 Agent: A Self-Referential Agent Framework for Recursive Self-Improvement. *arXiv preprint
 1110 arXiv:2410.04444*, 2024.

1111 Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Dongsheng Li, and Deqing Yang. Evoagent: Towards
 1112 automatic multi-agent generation via evolutionary algorithms. *arXiv preprint arXiv:2406.14228*,
 1113 2024.

1114 Eliezer Yudkowsky et al. Artificial Intelligence as a positive and negative factor in global risk. *Global
 1115 catastrophic risks*, 1(303):184, 2008.

1116 Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and
 1117 James Zou. Textgrad: Automatic" differentiation" via text. *arXiv preprint arXiv:2406.07496*,
 1118 2024.

1119 Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D Goodman.
 1120 Quiet-star: Language models can teach themselves to think before speaking. *arXiv preprint
 1121 arXiv:2403.09629*, 2024a.

1122 Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. Self-taught optimizer (stop):
 1123 Recursively self-improving code generation. In *First Conference on Language Modeling*, 2024b.

1124 Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. Rest-mcts*: Llm
 1125 self-training via process reward guided tree search. *Advances in Neural Information Processing
 1126 Systems*, 37:64735–64772, 2024a.

1134 Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent
 1135 architecture search via agentic supernet. *arXiv preprint arXiv:2502.04180*, 2025a.

1136

1137 Jenny Zhang, Joel Lehman, Kenneth Stanley, and Jeff Clune. OMNI: Open-endedness via Models
 1138 of human Notions of Interestingness. In *The Twelfth International Conference on Learning
 1139 Representations*, 2024b. URL <https://openreview.net/forum?id=AgM3MzT99c>.

1140

1141 Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge,
 1142 Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. *arXiv
 1143 preprint arXiv:2410.10762*, 2024c.

1144

1145 Yuanshuo Zhang, Yuchen Hou, Bohan Tang, Shuo Chen, Muhan Zhang, Xiaowen Dong, and Siheng
 1146 Chen. Gdns as predictors of agentic workflow performances. *arXiv preprint arXiv:2503.11301*,
 1147 2025b.

1148

1149 Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. Autocoderover: Autonomous
 1150 program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on
 1151 Software Testing and Analysis*, pp. 1592–1604, 2024d.

1152

1153 Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large
 1154 language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial
 1155 Intelligence*, volume 38, pp. 19724–19731, 2024.

1156

1157 Andy Zhou, Kevin Wu, Francesco Pinto, Zhaorun Chen, Yi Zeng, Yu Yang, Shuang Yang, Sanmi
 1158 Koyejo, James Zou, and Bo Li. AutoRedTeamer: Autonomous Red Teaming with Lifelong Attack
 1159 Integration. *arXiv preprint arXiv:2503.15754*, 2025.

1160

1161 Wangchunshu Zhou, Yixin Ou, Shengwei Ding, Long Li, Jialong Wu, Tiannan Wang, Jiamin Chen,
 1162 Shuai Wang, Xiaohua Xu, Ningyu Zhang, et al. Symbolic learning enables self-evolving agents.
 1163 *arXiv preprint arXiv:2406.18532*, 2024.

1164

1165 Yuqi Zhu, Jia Li, Ge Li, YunFei Zhao, Zhi Jin, and Hong Mei. Hot or cold? adaptive temperature
 1166 sampling for code generation with large language models. In *Proceedings of the AAAI Conference
 1167 on Artificial Intelligence*, volume 38, pp. 437–445, 2024.

1168

1169 Mingchen Zhuge, Haozhe Liu, Francesco Faccio, Dylan R. Ashley, Róbert Csordás, Anand Gopalakrishnan,
 1170 Abdullah Hamdi, Hasan Abed Al Kader Hammoud, Vincent Herrmann, Kazuki Irie, Louis
 1171 Kirsch, Bing Li, Guohao Li, Shuming Liu, Jinjie Mai, Piotr Piékos, Aditya Ramesh, Imanol
 1172 Schlag, Weimin Shi, Aleksandar Stanic, Wenyi Wang, Yuhui Wang, Mengmeng Xu, Deng-Ping
 1173 Fan, Bernard Ghanem, and Jürgen Schmidhuber. Mindstorms in natural language-based societies
 1174 of mind. *arXiv preprint arXiv:2305.17066*, 2023.

1175

1176

1177

1178

1179

1180

1181

1182

1183

1184

1185

1186

1187

1188
1189
1190
1191
1192
1193
1194

APPENDIX

1195
1196
1197
1198
1199
1200
1201
1202
1203
1204

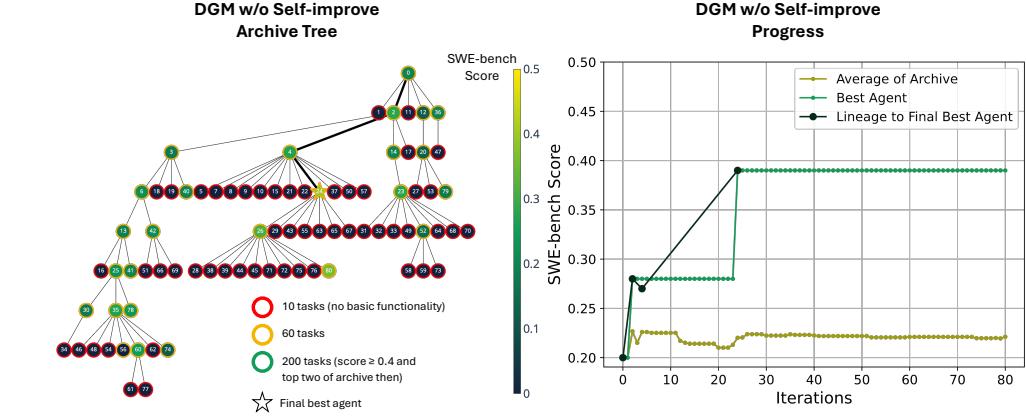
TABLE OF CONTENTS

A Additional Results	24
A.1 Baselines on SWE-bench	24
A.2 Generality across models on Polyglot	25
A.3 Ablation of Parent Selection	25
A.4 Additional Statistics of DGM runs	25
B Additional Related Work	26
C Algorithmic Details	27
C.1 Initial Coding Agent	27
C.2 Parent Selection	28
C.3 Self-Improve Prompts	28
C.4 Pseudocode	31
D Experiment Details	32
D.1 Hyperparameters for Foundation Models	32
E Benchmark Details	32
E.1 Cost Estimate	32
E.2 SWE-bench Tasks	32
E.3 Polyglot Tasks	35
E.4 SWE-bench State-of-The-Art	36
E.5 Polyglot Representative Agent	36
F Best-Discovered Agents	36
F.1 DGM on SWE-bench	36
F.2 DGM on Polyglot	50
G Similar Target Functionality, Different Implementations	59
H Case Study: Solving Hallucination	69
I Additional Safety Discussion	71
J Additional Future Work Directions	71

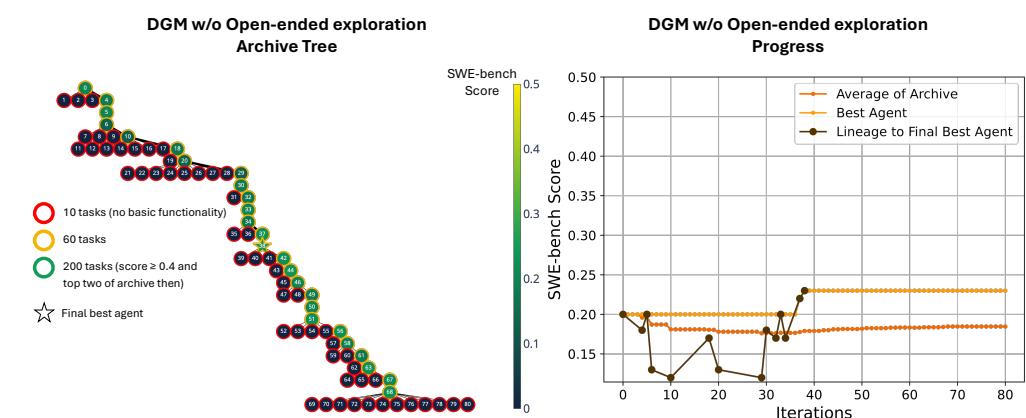
1235
1236
1237
1238
1239
1240
1241

1242
1243 A ADDITIONAL RESULTS
1244
1245
1246

A.1 BASELINES ON SWE-BENCH



1261 **Figure 5: DGM without self-improving agents.** Keeping the meta-agent that is modifying and
1262 producing the next coding agents the same, DGM w/o self-improve is unable to continuously improve
1263 over time. (Left) Archive of coding agents generated during the DGM w/o self-improve run on
1264 SWE-bench. Each node represents a coding agent, with node 0 corresponding to the base agent. Node
1265 color indicates performance on SWE-bench (percentage of solved tasks), while border color reflects
1266 the number of tasks for which the agent was evaluated. Edges show which agents self-modified to
1267 produce the offsprings. (Right) Progress plot of the DGM w/o self-improve on SWE-bench. The light
1268 green line shows the average score of all agents possessing basic codebase-editing functionality. The
1269 green line tracks the best score achieved by any agent in the archive at each iteration. The dark line
1270 shows the lineage of the final best-discovered agent and its precursor nodes.



1287 **Figure 6: DGM without open-ended exploration.** Removing the archive, DGM w/o open-ended
1288 exploration always uses the most recent agent to self-modify and makes very little progress on
1289 SWE-bench. (Left) Archive of coding agents generated during the DGM w/o open-ended exploration
1290 run on SWE-bench. Each node represents a coding agent, with node 0 corresponding to the base
1291 agent. Node color indicates performance on SWE-bench (percentage of solved tasks), while border
1292 color reflects the number of tasks for which the agent was evaluated. Edges show which agents
1293 self-modified to produce the offsprings. (Right) Progress plot of the DGM w/o open-ended exploration
1294 on SWE-bench. The orange line shows the average score of all agents possessing basic codebase-editing
1295 functionality. The light orange line tracks the best score achieved by any agent in the archive at each
iteration. The dark line shows the lineage of the final best-discovered agent and its precursor nodes.

1296

A.2 GENERALITY ACROSS MODELS ON POLYGLOT

1297

1298

1299

1300

1301

1302

1303

1304

1305

1306

1307

1308

1309

1310

1311

1312

In addition to testing the transfer models on SWE-bench (see Section 4.4, Figure 2), we also present the transfer results on Polyglot in this section. On Polyglot (Figure 7), where the DGM was run with o3-mini, we replaced the FM with Claude 3.5 Sonnet (New) or Claude 3.7 Sonnet, and evaluated on the full benchmark (Figure 4, Middle). With Claude 3.5 Sonnet (New), the initial agent achieved 32.0% and the DGM-discovered agent 33.3%. With Claude 3.7 Sonnet, the initial agent achieved 35.6% and the DGM-discovered agent 36.8%. These results suggest that the DGM yields improvements that generalize across FMs, rather than being tightly coupled to the specific FM used during its run (Figure 4).

1320

1321

A.3 ABLATION OF PARENT SELECTION

1322

1323

To further study the impact of the parent selection mechanism in DGM, we introduce DGM Greedy. DGM Greedy always selects the best-performing node as the parent to branch from, rather than giving every node a non-zero probability of being branched off (roughly proportional to their performance score and number of children) as in this implementation of the DGM (Appendix C.2). This ablation replicates the approach of Robeyns et al. (2025) in this setting. As shown in Table 1, DGM Greedy achieves 39.7% and 30.0% on SWE-bench and Polyglot, respectively, compared to 50.0% and 38.0% by this implementation of DGM. These results demonstrate that allowing all solutions in the archive to serve as potential stepping stones can lead to greater improvements over time, underscoring the importance of open-ended exploration.

1332

1333

Table 1: Comparison of DGM, its ablations, and baselines on SWE-bench and Polyglot benchmarks.

1334

Method	SWE-bench	Polyglot
DGM	50.0%	38.0%
DGM w/o Open-ended exploration	23.0%	14.0%
DGM w/o Self-improve	39.0%	28.0%
DGM Greedy	39.7%	30.0%

1340

1341

1342

A.4 ADDITIONAL STATISTICS OF DGM RUNS

1343

1344

1345

1346

1347

1348

1349

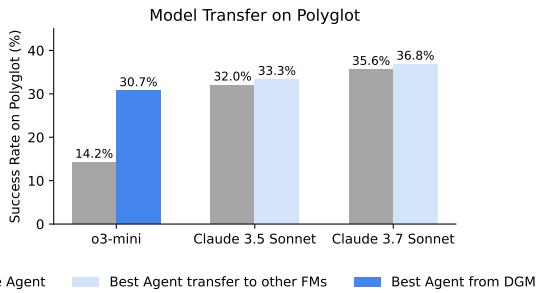


Figure 7: Transfer between Models on Polyglot

1350 Table 2: Percentage of generated agents with basic code-editing functionality on SWE-bench.
1351

1352 Method	1353 Percentage with Basic Code-Editing Functionality
1354 DGM	51.3%
1355 DGM w/o Open-ended exploration	32.5%
1356 DGM w/o Self-improve	32.5%

1357
1358 **Stability of DGM Runs.** To evaluate the stability of DGM, we run the DGM algorithm three times
1359 on the Polyglot benchmark and analyze the variance in performance. The DGM achieved a mean
1360 accuracy of 40.7% with a standard deviation of 2.3%, indicating that the DGM can achieve consistent
1361 and reproducible results across runs.
1362

1363 B ADDITIONAL RELATED WORK 1364

1365 **Open-Endedness (part 2).** Early approaches to open-endedness explored different mechanisms
1366 to balance learnability and interestingness. Quality-diversity algorithms sought to illuminate vast
1367 solution spaces with diverse, high-performing behaviors (Pugh et al., 2016; Chatzilygeroudis et al.,
1368 2021; Mouret & Clune, 2015; Nguyen et al., 2015). Other methods emphasized goal-directed
1369 exploration (Ecoffet et al., 2019; 2021; Schaul et al., 2015; Andrychowicz et al., 2017; Eysenbach
1370 et al., 2018), intrinsic motivation (Lehman & Stanley, 2011; Oudeyer et al., 2007; Li et al., 2014;
1371 Pathak et al., 2017), or learning progress frameworks (Kanitscheider et al., 2021; Gaven et al.,
1372 2025; Baranes & Oudeyer, 2013; Colas et al., 2019; 2022b; Jiang et al., 2021; Dennis et al., 2020;
1373 Schmidhuber, 2008; 2013; Kompella et al., 2017). More recently, large-scale foundation models
1374 (FMs) (Brown et al., 2020; Radford et al., 2019) have emerged as powerful proxies for human notions
1375 of interestingness (Zhang et al., 2024b; Faldor et al., 2025; Sancaktar et al., 2025) and effective
1376 mutation operators to propose novel solutions in code (Romera-Paredes et al., 2024; Novikov et al.,
1377 2025; Lehman et al., 2023; Faldor et al., 2025; Hu et al., 2025). FMs can guide autotelic agents (Colas
1378 et al., 2022b; 2023; 2022a), model human preferences for quality and diversity (Bradley et al., 2024;
1379 Ding et al., 2024; Wang et al., 2023b; Klissarov et al., 2023; 2024; Samvelyan et al., 2024; Lim
1380 et al., 2024; Havrilla et al., 2024a), design reward functions (Wiering & Schmidhuber, 1997; Wang
1381 et al., 2023a; Ma et al., 2023; Faldor et al., 2025), create simulated environments (Sudhakaran et al.,
1382 2023; Nasir & Togelius, 2023; Aki et al., 2024; Nasir et al., 2024; Bruce et al., 2024; Parker-Holder
1383 et al., 2024; Schmidhuber, 2013), drive ever-evolving multi-agent dynamics (Dharna et al., 2024;
1384 Zhou et al., 2025), search diverse ambulating robot morphologies (Lehman et al., 2023), and search
1385 expansive solution spaces for benchmark or objective optimization (Lange et al., 2024; Zhang et al.,
1386 2024b; Faldor et al., 2025; Hu et al., 2025; Lu et al., 2024b; Romera-Paredes et al., 2024; Fernando
1387 et al., 2024; Lu et al., 2024a; Khan et al., 2024; Lu et al., 2025; Liu et al., 2024; Novikov et al., 2025).
1388

1389 **Program Synthesis.** Program synthesis (Alur et al., 2018; Polozov & Gulwani, 2015; Buchi
1390 & Landweber, 1990; Gulwani, 2011; Ellis et al., 2021) seeks to generate code meeting external
1391 specifications such as input-output examples or logical formulas. Hybrid approaches combine
1392 symbolic methods with neural or FM guidance: for instance, Li et al. (2024) uses LLM suggestions
1393 to steer symbolic search in SyGuS settings, improving over pure enumeration. Barke et al. (2024)
1394 blends LLM completions with a learned surrogate model to guide synthesis in DSLs. Shi et al. (2023)
1395 uses neural policies to build higher-order and lambda abstractions during search, outperforming both
1396 LLM-only and symbolic baselines on list manipulation tasks. The DGM differs in focusing not
1397 just on producing programs for external tasks, but also on agent self-modification, rewriting its own
1398 implementation to improve its capacity for future self-improvement.

1399 **Inspiration from Darwinian Evolution.** This work is heavily inspired by the mechanisms of
1400 Darwinian evolution (Darwin, 2023), notably variation (mutation), selection, and the preservation
1401 of lineages (stepping stones), and brings them into the realm of self-modifying coding agents. In
1402 DGM, an archive of past agent versions is maintained, from which parent agents are sampled; then
1403 mutations (i.e., code edits) generate new offspring agents, which are empirically evaluated on coding
benchmarks; successful ones are added to the archive, enabling parallel exploration of multiple
evolutionary trajectories (Section 3). This mirrors how biological evolution (Edwards, 2000; Wright,

1404 1932) preserves genetic diversity (Mayr, 1982), leverages variation (Kimura, 1979), and uses natural
 1405 selection to retain beneficial mutations (Dobzhansky, 1970).

1406

1408 C ALGORITHMIC DETAILS

1409

1410 C.1 INITIAL CODING AGENT

1411

1412 In this section, we present the details of the tools available to the initial coding agent (Section 4.1)
 1413 and its task prompt.

1414

1415 Information of the given Bash tool:

```
1416 def tool_info():
1417     return {
1418         "name": "bash",
1419         "description": """Run commands in a bash shell\n
  * When invoking this tool, the contents of the "command" parameter does NOT need to be
    ↪ XML-escaped.\n
  * You don't have access to the internet via this tool.\n
  * You do have access to a mirror of common linux and python packages via apt and pip.\n
  * State is persistent across command calls and discussions with the user.\n
  * To inspect a particular line range of a file, e.g. lines 10-25, try 'sed -n 10,25p
    ↪ /path/to/the/file'.\n
  * Please avoid commands that may produce a very large amount of output.\n
  * Please run long lived commands in the background, e.g. 'sleep 10 &' or start a server in
    ↪ the background.""""",
1420         "input_schema": {
1421             "type": "object",
1422             "properties": {
1423                 "command": {
1424                     "type": "string",
1425                     "description": "The bash command to run."
1426                 }
1427             },
1428             "required": ["command"]
1429         }
1430     }
1431 }
```

1433

1434 Information of the given Edit tool:

```
1435 def tool_info():
1436     return {
1437         "name": "editor",
1438         "description": """Custom editing tool for viewing, creating, and editing files\n
  * State is persistent across command calls and discussions with the user.\n
  * If `path` is a file, `view` displays the entire file with line numbers. If `path` is a
    ↪ directory, `view` lists non-hidden files and directories up to 2 levels deep.\n
  * The `create` command cannot be used if the specified `path` already exists as a file.\n
  * If a `command` generates a long output, it will be truncated and marked with `<response
    ↪ clipped>`.\n
  * The `edit` command overwrites the entire file with the provided `file_text`.\n
  * No partial/line-range edits or partial viewing are supported.""""",
1439         "input_schema": {
1440             "type": "object",
1441             "properties": {
1442                 "command": {
1443                     "type": "string",
1444                     "enum": ["view", "create", "edit"],
1445                     "description": "The command to run: `view`, `create`, or `edit`."
1446                 },
1447                 "path": {
1448                     "description": "Absolute path to file or directory, e.g. `/repo/file.py` or
    ↪ `/repo/`.",
1449                     "type": "string"
1450                 },
1451                 "file_text": {
1452                     "description": "Required parameter of `create` or `edit` command,
    ↪ containing the content for the entire file.",
1453                     "type": "string"
1454                 }
1455             },
1456             "required": ["command", "path"]
1457         }
1458     }
```

1458

Task prompt:

1459

1460

1461

1462

1463

1464

1465

1466

1467

1468

1469

1470

1471

```

I have uploaded a Python code repository in the directory {self.git_tempdir}. Help solve
→ the following problem.

<problem_description>
{self.problem_statement}
</problem_description>

<test_description>
{self.test_description}
</test_description>

Your task is to make changes to the files in the {self.git_tempdir} directory to address
→ the <problem_description>. I have already taken care of the required dependencies.

```

1472

C.2 PARENT SELECTION

1473

At each DGM iteration, we select a subset of agents from the archive as parents to self-modify and produce new child agents (Section 3). The details of the parent selection process, inspired by Ecoffet et al. (2019), are outlined below. Future work could explore alternative search and exploration methods (Coulom, 2006; Silver et al., 2016; Herr et al., 2025).

1474

At each iteration t of the DGM run, let

1475

1476

$$\mathcal{A}^t = \{a_1^t, a_2^t, \dots, a_N^t\}$$

1477

1478

be the archive of candidate agents. We first define the eligible set

1479

1480

1481

$$\mathcal{E}^t = \{a_i^t \in \mathcal{A}^t : \alpha_i < 1\},$$

1482

i.e. only those agents whose performance score is not yet perfect. We then sample k parents (with replacement) from \mathcal{E}^t in proportion to a combined measure of performance and number of children with codebase-editing functionality. Concretely, for each agent $a_i^t \in \mathcal{E}^t$:

1483

1484

1485

1486

1487

1488

1489

1490

1491

1492

1493

1494

1495

1496

1497

1498

1499

1500

1501

1502

1503

1504

1505

1506

$$(\text{performance}) \quad \alpha_i = \text{performance}(a_i^t), \quad (1)$$

$$(\text{children count}) \quad n_i = \text{functioning_children_count}(a_i^t), \quad (2)$$

$$(\text{sigmoid-scaled performance}) \quad s_i = \frac{1}{1 + \exp(-\lambda(\alpha_i - \alpha_0))}, \quad (3)$$

$$(\text{novelty bonus}) \quad h_i = \frac{1}{1 + n_i}, \quad (4)$$

$$(\text{unnormalized weight}) \quad w_i = s_i h_i, \quad (5)$$

$$(\text{selection probability}) \quad p_i = \frac{w_i}{\sum_{j=1}^N w_j}, \quad (6)$$

$$(\text{sampling}) \quad \{\text{parents}\} \sim \text{Categorical}(\{p_i\}_{a_i^t \in \mathcal{E}^t}) \quad (\text{draw } k \text{ samples}). \quad (7)$$

Here, $\lambda > 0$ controls the sharpness of the sigmoid, and α_0 is the midpoint accuracy. We set $\lambda = 10$ and $\alpha_0 = 0.5$. k determines how many self-improvements are done in parallel during each iteration of the DGM run. We set $k = 2$ in SWE-bench experiments, and $k = 4$ in Polyglot experiments.

1507

1508

1509

1510

1511

C.3 SELF-IMPROVE PROMPTS

In every iteration, an agent is chosen to self-modify. For this step, the current implementation of DGM leverages a separate FM to diagnose improvement opportunities, which are then passed to the coding agent for implementation. To identify the next potential improvement, the evaluation logs from the chosen agent’s performance on the coding benchmark are fed into the FM. While the current implementation relies on an additional diagnostic step, there are no fundamental limitations preventing the DGM from autonomously analyzing its own performance, generating improvement

1512 hypotheses, and implementing subsequent modifications. We chose to use OpenAI's o1 model (Team
 1513 et al., 2024) for this purpose, as we found it had better reasoning capabilities than other FMs available
 1514 at the time of our experiments. The prompts used for this diagnosis are provided below. From
 1515 the diagnostic output, the `implementation_suggestion` and `problem_description` are
 1516 extracted and passed to the coding agent as its self-modification task.

1517 Self-improve diagnosis prompt for SWE-bench experiments:

1518

1519

1520

Self-Improve Prompt

1521

1522

1523

1524

1525

1526

1527

1528

1529

1530

1531

1532

1533

1534

1535

1536

1537

1538

1539

1540

1541

1542

1543

1544

1545

1546

1547

1548

1549

1550

1551

1552

1553

1554

1555

1556

1557

1558

1559

1560

1561

1562

1563

1564

1565

```
# Coding Agent Summary
- **Main File**: `coding_agent.py`
- Primary Class: `AgenticSystem`
- The `forward()` function is the central entry point.
- Prompts are located either within the `forward()` function or in the `prompts/` directory.
- **Tools**: `tools/`
- The `tools/` directory contains various tools that LLMs can use to perform specific tasks.
- Each tool must have a `tool_info()` function that returns a JSON object containing `name`, `description`, and `input_schema`. The `input_schema` should be a JSON object containing `type`, `properties`, and `required`.
- Each tool must have a `tool_function()` function that takes the arguments defined in `input_schema`, performs the tool's task, and returns a string.
- See other tools for reference.
- **Utilities**: `utils/`
- The `utils/` directory contains utility functions used across the codebase.

- **Additional Details**:
- The agent is very good at automatically utilizing the right available tools at the right time. So do not have an agentic flow that explicitly forces a tool's usage.
- Common tools, such as file editing and bash commands, are easy for the agent to recognize and use appropriately. However, more complex and niche tools may require explicit instructions in the prompt.
- Tools should be designed to be as general as possible, ensuring they work across any GitHub repository. Avoid hardcoding repository-specific details or behaviors (e.g., paths).
- Do not use 'while True' loops in the agent's code. This can cause the agent to get stuck and not respond.
- Verify the implementation details of helper functions prior to usage to ensure proper integration and expected behavior.
- Do not install additional packages or dependencies directly. Update `requirements.txt` if new dependencies are required and install them using `pip install -r requirements.txt`.
```

Here is the implementation of the coding agent.

```
# Coding Agent Implementation
----- Coding Agent Implementation Start -----
{code}
----- Coding Agent Implementation End -----
```

Your task is to identify ONE detailed plan that would improve the agent's coding ability. The improvement should not be specific to any particular GitHub issue or repository.

Here is the log for the coding agent trying to solve the GitHub issues but failed.

```
# Agent Running Log
----- Agent Running Log Start -----
{md_log}
----- Agent Running Log End -----
```

```
# GitHub Issue
The GitHub issue that the agent is trying to solve.
----- GitHub Issue Start -----
{github_issue}
----- GitHub Issue End -----
```

```
# Predicted Patch
The agent's predicted patch to solve the issue.
----- Predicted Patch Start -----
{predicted_patch}
----- Predicted Patch End -----
```

Private Test Patch

```
1566  
1567 SWE-bench's official private tests to detect whether the issue is solved. This is not  
1568 → available to the agent during evaluation. The agent should try to implement its own  
1569 → tests.  
1570 ----- Private Test Patch Start -----  
1571 {test_patch}  
1572 ----- Private Test Patch End -----  
1573  
1574 # Issue Test Results  
1575 The test results from SWE-bench using the above official private tests.  
1576 ----- Issue Test Results Start -----  
1577 {eval_log}  
1578 ----- Issue Test Results End -----  
1579  
1580 Respond precisely in the following format including the JSON start and end markers:  
1581  
1582 ```json  
<JSON>  
```  
1583
1584 In <JSON>, provide a JSON response with the following fields:
1585 - "log_summarization": Analyze the above logs and summarize how the agent tried to
1586 → solve the GitHub issue. Note which tools and how they are used, the agent's
1587 → problem-solving approach, and any issues encountered.
1588 - "potential_improvements": Identify potential improvements to the coding agent that
1589 → could enhance its coding capabilities. Focus on the agent's general coding
1590 → abilities (e.g., better or new tools usable across any repository) rather than
1591 → issue-specific fixes (e.g., tools only usable in one framework). All necessary
1592 → dependencies and environment setup have already been handled, so do not focus on
1593 → these aspects.
1594 - "improvement_proposal": Choose ONE high-impact improvement from the identified
1595 → potential improvements and describe it in detail. This should be a focused and
1596 → comprehensive plan to enhance the agent's overall coding ability.
1597 - "implementation_suggestion": Referring to the coding agent's summary and
1598 → implementation, think critically about what feature or tool could be added or
1599 → improved to best implement the proposed improvement. If the proposed feature can be
1600 → implemented by modifying the existing tools, describe the modifications needed,
1601 → instead of suggesting a new tool.
1602 - "problem_description": Phrase the improvement proposal and implementation suggestion
1603 → as a GitHub issue description. It should clearly describe the feature so that a
1604 → software engineer viewing the issue and the repository can implement it.
1605
1606 Your response will be automatically parsed, so ensure that the string response is
1607 → precisely in the correct format. Do NOT include the `<JSON>` tag in your output.
1608
```

1596  
1597 While the private test patch is included in the diagnosis prompt, one could imagine that an easy  
1598 way for the agent to score well on the task would be to hard code the solutions. However, from  
1599 manual inspection of the logs and agent modifications, we have not observed any problematic logic  
1600 or behavior indicative of memorization or overfitting to specific private test cases. Furthermore,  
1601 improvements discovered on one task or LLM model often generalize well to others (Figure 4),  
1602 showing that these improvements are not task-specific but instead reflect general changes that enhance  
overall performance.

## Additional prompt for Polyglot experiments:

Additional Prompt for o3-mini Tool Use and Polyglot

```
1607 ### DOC: tool function schema
1608
1609 Carefully consider whether to add/enhance the current tool or edit the workflow in
1610 ↪ forward()
1611
1612 Pay special attention to making sure that "required" and "type" are always at the
1613 ↪ correct level of nesting. For example, "required" should be at the same level as
1614 ↪ "properties", not inside it.
1615 Make sure that every property, no matter how short, has a type and description
1616 ↪ correctly nested inside it.
1617 Other arguments than you have seen are not permitted. For example, in
1618 ↪ "edit_line_ranges" with "type": "array", arguments like "minItems" and "maxItems"
1619 ↪ are not permitted.
1620
1621 ...
1622
1623 Here is the log for the coding agent trying to solve a programming task.
1624 A task is in one programming language, but the coding agent needs to deal with
1625 ↪ different languages including C++, Go, Java, JavaScript, Python, and Rust.
```

1620 C.4 PSEUDOCODE  
 1621  
 1622  
 1623

1624 This is the pseudocode of the DGM algorithm, described in Section 3:  
 1625

---

1626 **Algorithm 1:** Darwin Gödel Machine

1627 **Input:** Initial coding agent  $g_0$ , benchmark suite  $B$ , maximum iterations  $T$

1628 **Output:** Archive of agents  $\mathcal{A}$

```
1629 initialize $\mathcal{A} \leftarrow \{g_0\}$ // Start with the base agent
1630 for $t \leftarrow 1$ to T do
1631 $\mathcal{P} \leftarrow \text{SelectParents}(\mathcal{A})$ // Select parent agents
1632 foreach $p \in \mathcal{P}$ do
1633 $c \leftarrow p.\text{modify}(p)$ // Self-modification
1634 $s \leftarrow \text{evaluate}(c, B)$ // Evaluate on benchmark
1635 if $c.\text{is_valid}()$ then
1636 $\mathcal{A} \leftarrow \mathcal{A} \cup \{(c, s)\}$ // Keep children capable of code editing
1637 end
1638 end
1639 end
1640 return \mathcal{A}
```

---

1641 This is the pseudocode of the baseline DGM without self-improving agents, described in Section 4.3:  
 1642

---

1643 **Algorithm 2:** Darwin Gödel Machine without Self-improving agents

1644 **Input:** Initial coding agent  $g_0$ , benchmark suite  $B$ , maximum iterations  $T$

1645 **Output:** Archive of agents  $\mathcal{A}$

```
1646 initialize $\mathcal{A} \leftarrow \{g_0\}$ // Start with the base agent
1647 for $t \leftarrow 1$ to T do
1648 $\mathcal{P} \leftarrow \text{SelectParents}(\mathcal{A})$ // Select parent agents
1649 foreach $p \in \mathcal{P}$ do
1650 $c \leftarrow g_0.\text{modify}(p)$ // Modify with base agent
1651 $s \leftarrow \text{evaluate}(c, B)$ // Evaluate on benchmark
1652 if $c.\text{is_valid}()$ then
1653 $\mathcal{A} \leftarrow \mathcal{A} \cup \{(c, s)\}$ // Keep children capable of code editing
1654 end
1655 end
1656 end
1657 return \mathcal{A}
```

---

1658 This is the pseudocode of the baseline DGM without open-ended exploration, described in Section 4.3:  
 1659

---

1660 **Algorithm 3:** Darwin Gödel Machine without Open-ended exploration

1661 **Input:** Initial coding agent  $g_0$ , benchmark suite  $B$ , maximum iterations  $T$

1662 **Output:** Archive of agents  $\mathcal{A}$

```
1663 initialize $\mathcal{A} \leftarrow \{g_0\}$ // Start with the base agent
1664 for $t \leftarrow 1$ to T do
1665 $\mathcal{P} \leftarrow \text{SelectParents}(\mathcal{A})$ // Select parent agents
1666 foreach $p \in \mathcal{P}$ do
1667 $c \leftarrow p.\text{modify}(p)$ // Self-modification
1668 $s \leftarrow \text{evaluate}(c, B)$ // Evaluate on benchmark
1669 if $c.\text{is_valid}()$ then
1670 $\mathcal{A} \leftarrow \{(c, s)\}$ // Only keep the latest agent
1671 end
1672 end
1673 end
return \mathcal{A}
```

---

1674 **D EXPERIMENT DETAILS**  
16751676 **D.1 HYPERPARAMETERS FOR FOUNDATION MODELS**  
16771678 Table 3 shows the foundation model used in each experiment setting, as described in Section 4.1.  
1679 Since SWE-bench is a more challenging coding benchmark, we use a stronger coding model, Claude  
1680 3.5 Sonnet (New) (based on our preliminary testing). To enable faster iterations and avoid the same  
1681 rate limits as Claude, we use o3-mini for Polyglot experiments. The temperature for all FMs in every  
1682 setting is set to 1.0.  
16831684 Table 3: Foundation models used in each experiment setting.  
1685

| Benchmark         | SWE-bench               | Polyglot                |
|-------------------|-------------------------|-------------------------|
| Self-modification | Claude 3.5 Sonnet (New) | Claude 3.5 Sonnet (New) |
| Evaluation        | Claude 3.5 Sonnet (New) | o3-mini                 |

1690 **E BENCHMARK DETAILS**  
16911692 **E.1 COST ESTIMATE**  
16931694 The estimated cost of completing a single run of the DGM on SWE-bench, as presented in Section 4,  
1695 is about USD 22,000. In comparison, the estimated cost of completing a single run of either baseline  
1696 (DGM w/o self-improve or DGM w/o open-ended exploration) on SWE-bench is about USD 10,000.  
1697 Although the DGM is considerably more costly than the baselines, a method that can continuously  
1698 improve, even at a higher cost, is preferable to one that fails to improve or stagnates at a level of  
1699 performance that may never match that of the DGM. A more granular break down is:  
1700

| LLM                     | Benchmark | Number of Tasks | Cost Estimate (USD) |
|-------------------------|-----------|-----------------|---------------------|
| Claude 3.5 Sonnet (New) | SWE-bench | 60              | \$350               |
| o3-mini                 | Polyglot  | 60              | \$5                 |

1705 We acknowledge that the current experiments on SWE-bench require considerable compute. Hence,  
1706 we also include experiments on another benchmark, Polyglot, with significantly lower costs. This  
1707 suggests that expenses vary greatly by task complexity, with SWE-bench being among the more  
1708 complex and resource-intensive coding benchmarks. Moreover, several impactful methods (e.g., LLM  
1709 training at its inception) were characterized substantial computational demands initially. Similar to  
1710 these pioneering works, we hope to open the door to future research on improving the efficiency and  
1711 scalability of our approach. In addition, many leading coding agents on the SWE-bench leaderboard  
1712 are backed by industrial companies employing expert full-time researchers and engineers, which  
1713 incurs substantial human labor costs. In contrast, our approach achieves SoTA-level performance  
1714 through fully autonomous self-improvement without human intervention, potentially offering greater  
1715 efficiency when considering the comparative costs of specialized AI development talent versus API  
1716 usage. Finally, as FMs continue to improve and compute costs continue to decline, methods like the  
1717 DGM will become increasingly efficient and accessible.  
17181719 Also, higher-performing agents discovered by the DGM do indeed incur greater inference costs than  
1720 the initial agent, but cost and performance are not strictly correlated, where some expensive agents  
1721 underperform cheaper ones.  
17221723 **E.2 SWE-BENCH TASKS**  
17241725 Initial 10 tasks for verifying basic functionality of a coding agent:  
1726  
1727

1728       • django\_django-10973  
 1729       • django\_django-11066  
 1730       • django\_django-12754  
 1731       • django\_django-15930  
 1732       • django\_django-13279  
 1733       • django\_django-16661  
 1734       • django\_django-13346  
 1735       • django\_django-10880  
 1736       • django\_django-10999  
 1737       • django\_django-11087  
 1738       Additional 50 tasks for estimating general effectiveness of a coding agent:  
 1739       • django\_django-9296       • sphinx\_doc\_sphinx-7454  
 1740       • django\_django-11790       • sphinx\_doc\_sphinx-7590  
 1741       • django\_django-11815       • sphinx\_doc\_sphinx-7748  
 1742       • django\_django-11848       • sphinx\_doc\_sphinx-7757  
 1743       • django\_django-11880       • sphinx\_doc\_sphinx-7985  
 1744       • django\_django-11885       • sphinx\_doc\_sphinx-8035  
 1745       • django\_django-11951       • sphinx\_doc\_sphinx-8056  
 1746       • django\_django-11964       • sphinx\_doc\_sphinx-8265  
 1747       • django\_django-11999       • sphinx\_doc\_sphinx-8269  
 1748       • django\_django-12039       • sphinx\_doc\_sphinx-8475  
 1749       • django\_django-12050       • sphinx\_doc\_sphinx-8548  
 1750       • django\_django-12143       • sphinx\_doc\_sphinx-8551  
 1751       • django\_django-12155       • sphinx\_doc\_sphinx-8638  
 1752       • django\_django-12193       • sphinx\_doc\_sphinx-8721  
 1753       • django\_django-12209       • sphinx\_doc\_sphinx-9229  
 1754       • django\_django-12262       • sphinx\_doc\_sphinx-9230  
 1755       • django\_django-12273       • sphinx\_doc\_sphinx-9281  
 1756       • django\_django-12276       • sphinx\_doc\_sphinx-9320  
 1757       • django\_django-12304       • sphinx\_doc\_sphinx-9367  
 1758       • django\_django-12308       • sphinx\_doc\_sphinx-9461  
 1759       • django\_django-12325       • sphinx\_doc\_sphinx-9698  
 1760       • django\_django-12406       • sphinx\_doc\_sphinx-10449  
 1761       • django\_django-12708       • sphinx\_doc\_sphinx-10466  
 1762       • django\_django-12713       • sphinx\_doc\_sphinx-10673  
 1763       • django\_django-12774       • sphinx\_doc\_sphinx-11510  
 1764       Additional 140 tasks for more accurate assessment of a coding agent's performance:  
 1765       • astropy\_astropy-12907       • astropy\_astropy-14309  
 1766       • astropy\_astropy-13033       • astropy\_astropy-14365  
 1767       • astropy\_astropy-13236       • astropy\_astropy-14369  
 1768       • astropy\_astropy-13398       • astropy\_astropy-14508  
 1769       • astropy\_astropy-13453       • astropy\_astropy-14539  
 1770       • astropy\_astropy-13579       • astropy\_astropy-14598  
 1771       • astropy\_astropy-13977       • astropy\_astropy-14995  
 1772       • astropy\_astropy-14096       • astropy\_astropy-7166  
 1773       • astropy\_astropy-14182       • astropy\_astropy-7336

- 1782 • astropy\_\_astropy-7606
- 1783 • astropy\_\_astropy-7671
- 1784 • astropy\_\_astropy-8707
- 1785 • astropy\_\_astropy-8872
- 1786 • django\_\_django-10097
- 1787 • django\_\_django-10554
- 1788 • django\_\_django-10914
- 1789 • django\_\_django-11095
- 1790 • django\_\_django-11099
- 1791 • django\_\_django-11119
- 1792 • django\_\_django-11133
- 1793 • django\_\_django-11138
- 1794 • django\_\_django-11141
- 1795 • django\_\_django-11149
- 1800 • django\_\_django-11163
- 1801 • django\_\_django-11179
- 1802 • django\_\_django-11206
- 1803 • django\_\_django-11211
- 1804 • django\_\_django-11239
- 1806 • django\_\_django-11265
- 1807 • django\_\_django-11276
- 1808 • django\_\_django-11292
- 1810 • django\_\_django-11299
- 1811 • django\_\_django-11333
- 1812 • django\_\_django-11400
- 1813 • django\_\_django-11433
- 1814 • django\_\_django-11451
- 1815 • django\_\_django-11477
- 1817 • django\_\_django-11490
- 1818 • django\_\_django-11532
- 1819 • django\_\_django-11551
- 1821 • django\_\_django-11555
- 1822 • django\_\_django-11603
- 1823 • django\_\_django-11728
- 1824 • django\_\_django-11734
- 1825 • django\_\_django-11740
- 1827 • django\_\_django-11749
- 1828 • django\_\_django-11820
- 1829 • django\_\_django-12125
- 1831 • django\_\_django-12419
- 1832 • django\_\_django-12663
- 1833 • django\_\_django-12741
- 1834 • django\_\_django-12858
- 1835 • django\_\_django-12965
- django\_\_django-13012
- django\_\_django-13023
- django\_\_django-13028
- django\_\_django-13033
- django\_\_django-13089
- django\_\_django-13109
- django\_\_django-13112
- django\_\_django-13121
- django\_\_django-13128
- django\_\_django-13158
- django\_\_django-13195
- django\_\_django-13212
- django\_\_django-13297
- django\_\_django-13315
- django\_\_django-13343
- django\_\_django-13344
- django\_\_django-13363
- django\_\_django-13401
- django\_\_django-13406
- django\_\_django-13410
- django\_\_django-13417
- django\_\_django-13449
- django\_\_django-13512
- django\_\_django-13513
- django\_\_django-13516
- django\_\_django-13551
- django\_\_django-13568
- django\_\_django-13569
- django\_\_django-13590
- django\_\_django-13658
- django\_\_django-13670
- django\_\_django-13741
- django\_\_django-13786
- django\_\_django-13794
- django\_\_django-13807
- django\_\_django-13809
- django\_\_django-13810
- django\_\_django-13820
- django\_\_django-13821
- django\_\_django-13837
- django\_\_django-13925
- django\_\_django-13933
- django\_\_django-13964
- django\_\_django-14007
- django\_\_django-14011

```

1836 • django__django-14017
1837 • django__django-14034
1838 • django__django-14053
1839 • django__django-14089
1840 • django__django-14122
1841 • django__django-14140
1842 • django__django-14155
1843 • django__django-14170
1844 • django__django-14238
1845 • django__django-14311
1846 • django__django-14315
1847 • django__django-14349
1848 • django__django-14351
1849 • django__django-14373
1850 • django__django-14376
1851 • django__django-14404
1852 • django__django-14434
1853 • django__django-14493
1854 • django__django-14500
1855 • django__django-14534
1856 • django__django-14539
1857 • django__django-14559
1858 • django__django-14580
1859 • django__django-14608
1860 • django__django-14631
1861 • django__django-14672
1862 • django__django-14725
1863 • django__django-14752
1864 • django__django-14765
1865 • django__django-14771
1866 • django__django-14787
1867 • django__django-14792
1868 • django__django-14855

```

### E.3 POLYGLOT TASKS

Initial 10 tasks for verifying basic functionality of a coding agent:

```

1862 • go__dominoes
1863 • cpp__all-your-base
1864 • python__dominoes
1865 • java__sgf-parsing
1866 • javascript__robot-name
1867 • rust__variable-length-quantity
1868 • python__beer-song
1869 • go__book-store
1870 • javascript__bottle-song
1871 • rust__bowling

```

Additional 50 tasks for estimating general effectiveness of a coding agent:

```

1871 • javascript__queen-attack
1872 • rust__wordy
1873 • python__dot-dsl
1874 • java__satellite
1875 • cpp__diamond
1876 • rust__accumulate
1877 • go__error-handling
1878 • cpp__queen-attack
1879 • rust__poker
1880 • python__sgf-parsing
1881 • rust__react
1882 • java__ledger
1883 • go__connect
1884 • rust__macros
1885 • javascript__triangle
1886 • java__zipper
1887 • java__bowling
1888 • python__tree-building
1889 • javascript__say

```

```

1870 • java__wordy
1871 • python__food-chain
1872 • javascript__wordy
1873 • python__poker
1874 • javascript__grade-school
1875 • cpp__gigasecond
1876 • java__forth
1877 • python__dominoes
1878 • go__word-search
1879 • javascript__simple-linked-list
1880 • go__counter
1881 • java__react
1882 • javascript__ocr-numbers

```

```

1890 • python__scale-generator
1891 • java__go-counting
1892 • rust__doubly-linked-list
1893 • python__grade-school
1894 • javascript__forth
1895 • python__wordy
1896 • java__mazy-mice
1897 • cpp__bank-account
1898 • python__zipper
1899
1900 • java__custom-set
1901 • java__rest-api
1902 • go__transpose
1903 • rust__gigasecond
1904 • rust__say
1905 • go__food-chain
1906 • rust__pig-latin
1907 • go__markdown
1908 • go__crypto-square
1909
1910
1911 E.4 SWE-BENCH STATE-OF-THE-ART
1912
1913 At the time of writing this paper (16 April 2025), the highest performing, checked (i.e., the SWE-bench
1914 team received access to the system and were able to reproduce the patch generations), open-source
1915 entry on SWE-bench Verified is OpenHands + CodeAct v2.1 (claude-3-5-sonnet-20241022) (Wang
1916 et al., 2024), achieving 53.0%. Only considering the same subset of 200 tasks used by the DGM
1917 (Appendix E.2), OpenHands + CodeAct v2.1 (claude-3-5-sonnet-20241022) achieves 51.0%.
1918
1919
1920 E.5 POLYGLOT REPRESENTATIVE AGENT
1921
1922 Aider (Gauthier, 2024), a popular coding agent in the community, was published in Spring 2024. It
1923 has garnered over 33,000 stars on GitHub and has been continuously developed and tested against
1924 the Polyglot benchmark for over a year by human developers, primarily to evaluate its performance.
1925 Aider has also become a standard baseline for assessing the performance of different models, with
1926 the current top performers on the Polyglot benchmark being a mix of o3 (high) and GPT-4.1. We
1927 adopt a setup similar to that of the Polyglot leaderboard, with one key difference: the leaderboard
1928 reports pass@2 performance, where the agent can view feedback from ground-truth tests once. In
1929 contrast, we use a pass@1 setting, where the agent never sees the results of ground-truth tests, as we
1930 believe this more closely reflects realistic coding applications.
1931
1932
1933 F BEST-DISCOVERED AGENTS
1934
1935 F.1 DGM ON SWE-BENCH
1936
1937 Diff patches contributing to the best agent discovered by the DGM on SWE-bench:
1938
1939
1940
1941
1942
1943

```

```

diff --git a/coding_agent.py b/coding_agent.py
index 2cd395a..9a2cc2f 100644
--- a/coding_agent.py
+++ b/coding_agent.py
@@ -4,6 +4,7 @@ import logging
 from logging.handlers import RotatingFileHandler
 import os
 import threading
+import re

 from llm_withtools import CLAUDE_MODEL, OPENAI_MODEL, chat_with_agent
 from utils.eval_utils import get_report_score, msg_history_to_report, score_tie_breaker
@@ -63,6 +64,42 @@ def safe_log(message, level=logging.INFO):
 else:
 print(f"Warning: No logger found for thread {threading.get_ident()}")

+def is_patch_valid(patch_str):
+ """
+ Parse the patch to check if any non-test source files are modified.
+ Returns (bool, str) tuple: (is_valid, reason)
+ """
+ if not patch_str or patch_str.isspace():
+ return False, "Empty patch"
+
+ # Parse the patch to find modified files
+ modified_files = []

```

```

1944
1945 + diff_header_pattern = re.compile(r'^\+\+\+ b/(.+)$', re.MULTILINE)
1946 + for match in diff_header_pattern.finditer(patch_str):
1947 + filepath = match.group(1)
1948 + if filepath != '/dev/null': # Skip deleted files
1949 + modified_files.append(filepath)
1950 +
1951 + if not modified_files:
1952 + return False, "No files modified"
1953 +
1954 + # Check if any non-test files are modified
1955 + test_patterns = (
1956 + lambda f: f.startswith('tests/'),
1957 + lambda f: f.startswith('test_'),
1958 + lambda f: f.endswith('_test.py')
1959 +)
1960 +
1961 + source_files = [
1962 + f for f in modified_files
1963 + if not any(pattern(f) for pattern in test_patterns)
1964 +]
1965 +
1966 + if not source_files:
1967 + return False, "Only test files were modified"
1968 +
1969 + return True, "Valid patch with source file modifications"
1970 +
1971 class AgenticsSystem:
1972 def __init__(
1973 self,
1974 @@ -73,6 +110,7 @@ class AgenticsSystem:
1975 test_description=None,
1976 self_improve=False,
1977 instance_id=None,
1978 + max_retries=3,
1979):
1980 self.problem_statement = problem_statement
1981 self.git_tempdir = git_tempdir
1982 @@ -82,6 +120,7 @@ class AgenticsSystem:
1983 self.self_improve = self_improve
1984 self.instance_id = instance_id if not self_improve else 'dgm'
1985 self.code_model = CLAUDE_MODEL
1986 + self.max_retries = max_retries
1987
1988 # Initialize logger and store it in thread-local storage
1989 self.logger = setup_logger(chat_history_file)
1990 @@ -153,7 +192,7 @@ Your task is to run the regression tests in the {self.git_tempdir}
1991 ↪ directory to
1992 """
1993 The forward function for the AgenticsSystem.
1994 """
1995
1996 - instruction = f"""I have uploaded a Python code repository in the directory
1997 ↪ {self.git_tempdir}. Help solve the following problem.
1998 + base_instruction = f"""I have uploaded a Python code repository in the directory
1999 ↪ {self.git_tempdir}. Help solve the following problem.
2000
2001 <problem_description>
2002 {self.problem_statement}
2003 @@ -165,7 +204,39 @@ Your task is to run the regression tests in the {self.git_tempdir}
2004 ↪ directory to
2005
2006 Your task is to make changes to the files in the {self.git_tempdir} directory to address
2007 ↪ the <problem_description>. I have already taken care of the required dependencies.
2008 """
2009 - new_msg_history = chat_with_agent(instruction, model=self.code_model, msg_history=[], logging=safe_log)
2010 +
2011 + retry_count = 0
2012 + while retry_count < self.max_retries:
2013 + safe_log(f"\n==== Attempt {retry_count + 1} of {self.max_retries} ===")
2014 +
2015 + # Reset to base commit before each attempt
2016 + if retry_count > 0:
2017 + reset_to_commit(self.git_tempdir, self.base_commit)
2018 +
2019 + # Add retry context to instruction if this is a retry attempt
2020 + instruction = base_instruction
2021 + if retry_count > 0:
2022 + instruction += f"""\nNOTE: Previous attempt(s) failed because they either produced empty
2023 ↪ patches or only modified test files.
2024 +Please ensure your solution includes changes to the main source code files, not just test
2025 ↪ files."""

```

```

1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
+
+ # Run the agent
+ new_msg_history = chat_with_agent(instruction, model=self.code_model, msg_history=[],

+ ↪ logging=safe_log)
+
+ # Check the patch
+ patch = self.get_current_edits()
+ is_valid, reason = is_patch_valid(patch)
+
+ if is_valid:
+ safe_log(f"Valid patch generated: {reason}")
+ break
+ else:
+ safe_log(f"Invalid patch: {reason}")
+ if retry_count < self.max_retries - 1:
+ safe_log("Retrying with a new attempt...")
+ else:
+ safe_log("Maximum retries reached. Unable to generate a valid patch.")
+
+ retry_count += 1
+
def main():
 parser = argparse.ArgumentParser(description='Process repository with an agentic

 ↪ system.')
@@ -177,6 +248,7 @@ def main():
 parser.add_argument('--test_description', default=None, required=False,

 ↪ help='Description of how to test the repository')
 parser.add_argument('--self_improve', default=False, action='store_true',

 ↪ help='Whether to self-improve the repository or solving swe')
 parser.add_argument('--instance_id', default=None, help='Instance ID for SWE issue')
+ parser.add_argument('--max_retries', type=int, default=3, help='Maximum number of patch

 ↪ generation attempts')
 args = parser.parse_args()

 # Process the repository
@@ -188,6 +260,7 @@ def main():
 test_description=args.test_description,
 self_improve=args.self_improve,
 instance_id=args.instance_id,
+ max_retries=args.max_retries,
)
)

 # Run the agentic system to try to solve the problem
@@ -200,4 +273,4 @@ def main():
 f.write(model_patch)

if __name__ == "__main__":
- main()
+ main()
\ No newline at end of file
diff --git a/tests/test_patch_validator.py b/tests/test_patch_validator.py
new file mode 100644
index 0000000..5689f7d
--- /dev/null
+++ b/tests/test_patch_validator.py
@@ -0,0 +1,77 @@
+import pytest
+from coding_agent import is_patch_valid
+
+def test_empty_patch():
+ # Test empty patch
+ is_valid, reason = is_patch_valid("")
+ assert not is_valid
+ assert reason == "Empty patch"
+
+ # Test whitespace-only patch
+ is_valid, reason = is_patch_valid(" \n ")
+ assert not is_valid
+ assert reason == "Empty patch"
+
+def test_test_only_patch():
+ patch = """
+diff --git a/tests/test_edit_tool.py b/tests/test_edit_tool.py
+index abc123..def456 100644
+--- a/tests/test_edit_tool.py
++++ b/tests/test_edit_tool.py
+@@ -10,6 +10,8 @@ def test_something():
+ assert True
+ assert 1 == 1
+"""
+ is_valid, reason = is_patch_valid(patch)

```

```

2052
2053 + assert not is_valid
2054 + assert reason == "Only test files were modified"
2055
2056 +def test_source_file_patch():
2057 + patch = """
2058 +diff --git a/tools/edit.py b/tools/edit.py
2059 +index abc123..def456 100644
2060 +--- a/tools/edit.py
2061 +--- b/tools/edit.py
2062 +@@ -10,6 +10,8 @@ class Editor:
2063 + def edit(self):
2064 + pass
2065 + return True
2066 +"""
2067 + is_valid, reason = is_patch_valid(patch)
2068 + assert is_valid
2069 + assert reason == "Valid patch with source file modifications"
2070
2071 +def test_mixed_files_patch():
2072 + patch = """
2073 +diff --git a/tools/edit.py b/tools/edit.py
2074 +index abc123..def456 100644
2075 +--- a/tools/edit.py
2076 +--- b/tools/edit.py
2077 +@@ -10,6 +10,8 @@ class Editor:
2078 + def edit(self):
2079 + pass
2080 + return True
2081 +
2082 +diff --git a/tests/test_edit.py b/tests/test_edit.py
2083 +index abc123..def456 100644
2084 +--- a/tests/test_edit.py
2085 +--- b/tests/test_edit.py
2086 +@@ -10,6 +10,8 @@ def test_something():
2087 + assert True
2088 + assert 1 == 1
2089 +"""
2090 + is_valid, reason = is_patch_valid(patch)
2091 + assert is_valid
2092 + assert reason == "Valid patch with source file modifications"
2093
2094 +def test_no_files_modified():
2095 + patch = """
2096 +diff --git a/nonexistent.py b/nonexistent.py
2097 +deleted file mode 100644
2098 +index abc123..0000000
2099 +--- a/nonexistent.py
2100 +--- /dev/null
2101 +"""
2102 + is_valid, reason = is_patch_valid(patch)
2103 + assert not is_valid
2104 + assert reason == "No files modified"
2105 \ No newline at end of file
2106
2107
2108
2109 diff --git a/tools/edit.py b/tools/edit.py
2110 index 59137ee..16ae521 100644
2111 --- a/tools/edit.py
2112 +-- b/tools/edit.py
2113 @@ -1,16 +1,17 @@
2114 from pathlib import Path
2115 import subprocess
2116 +from typing import Optional, List, Tuple, Union
2117
2118 def tool_info():
2119 return {
2120 "name": "editor",
2121 "description": """Custom editing tool for viewing, creating, and editing files\n
2122 * State is persistent across command calls and discussions with the user.\n
2123 * If `path` is a file, `view` displays the entire file with line numbers. If `path` is a\n
2124 → directory, `view` lists non-hidden files and directories up to 2 levels deep.\n
2125 * If `path` is a file, `view` displays the file with line numbers. With optional\n
2126 → `view_range` [start, end], it displays only specified lines. Use -1 in `end` for\n
2127 → all remaining lines.\n
2128 * If `path` is a directory, `view` lists non-hidden files and directories up to 2 levels\n
2129 → deep.\n
2130 * The `create` command cannot be used if the specified `path` already exists as a file.\n
2131 * If a `command` generates a long output, it will be truncated and marked with `<response\n
2132 → clipped>`.\n
2133 * The `edit` command overwrites the entire file with the provided `file_text`.\n
2134 * No partial/line-range edits or partial viewing are supported."""
2135

```

```

2106
2107 """ The `edit` command overwrites the entire file with the provided `file_text`."""
2108 "input_schema": {
2109 "type": "object",
2110 "properties": {
2111 "@@ -26,6 +27,13 @@ def tool_info():
2112 "file_text": {
2113 "description": "Required parameter of `create` or `edit` command,
2114 ↪ containing the content for the entire file.",
2115 "type": "string"
2116 },
2117 "view_range": {
2118 "description": "Optional parameter for `view` command. Array of [start_line, end_line]
2119 ↪ (1-based). Use -1 for end_line to read until end of file.",
2120 "type": "array",
2121 "items": {"type": "integer"},
2122 "minItems": 2,
2123 "maxItems": 2
2124 },
2125 "required": ["command", "path"]
2126 @@ -89,6 +97,46 @@ def read_file(path: Path) -> str:
2127 except Exception as e:
2128 raise ValueError(f"Failed to read file: {e}")
2129
2130 +def read_file_range(path: Path, line_range: Optional[List[int]] = None) -> Tuple[str,
2131 ↪ int]:
2132 + """
2133 + Read and return file contents within specified line range.
2134 + Returns tuple of (content, start_line).
2135 +
2136 + Args:
2137 + path: Path object for the file
2138 + line_range: Optional [start, end] line numbers (1-based). Use -1 for end to read until
2139 ↪ EOF.
2140 +
2141 + try:
2142 + if line_range is None:
2143 return read_file(path), 1
2144 +
2145 + start, end = line_range
2146 + if start < 1:
2147 raise ValueError("Start line must be >= 1")
2148 + if end != -1 and end < start:
2149 raise ValueError("End line must be >= start line or -1")
2150 +
2151 + with path.open() as f:
2152 # Skip lines before start
2153 for _ in range(start - 1):
2154 next(f, None)
2155 +
2156 lines = []
2157 current_line = start
2158 while True:
2159 line = next(f, None)
2160 if line is None: # EOF
2161 break
2162 if end != -1 and current_line > end:
2163 break
2164 lines.append(line.rstrip('\n'))
2165 current_line += 1
2166 +
2167 return '\n'.join(lines), start
2168 +
2169 + except Exception as e:
2170 raise ValueError(f"Failed to read file range: {e}")
2171 +
2172 def write_file(path: Path, content: str):
2173 """Write (overwrite) entire file contents."""
2174 try:
2175 @@ -96,9 +144,18 @@ def write_file(path: Path, content: str):
2176 except Exception as e:
2177 raise ValueError(f"Failed to write file: {e}")
2178
2179 -def view_path(path_obj: Path) -> str:
2180 - """View the entire file contents or directory listing."""
2181 +def view_path(path_obj: Path, view_range: Optional[List[int]] = None) -> str:
2182 + """
2183 + View the file contents (optionally within a range) or directory listing.
2184 +
2185 + Args:
2186 + path_obj: Path object for the file or directory

```

```

2160
2161 + view_range: Optional [start, end] line numbers for file viewing
2162 + """
2163 if path_obj.is_dir():
2164 + if view_range is not None:
2165 + raise ValueError("view_range is not supported for directory listings")
2166 +
2167 # For directories: list non-hidden files up to 2 levels deep
2168 try:
2169 result = subprocess.run(
2170 @@ -115,14 +172,14 @@ def view_path(path_obj: Path) -> str:
2171 except Exception as e:
2172 raise ValueError(f"Failed to list directory: {e}")
2173
2174 - # If it's a file, show the entire file with line numbers
2175 - content = read_file(path_obj)
2176 - return format_output(content, str(path_obj))
2177 + # If it's a file, show the file content (with optional line range)
2178 + content, start_line = read_file_range(path_obj, view_range)
2179 + return format_output(content, str(path_obj), start_line)
2180
2181 -def tool_function(command: str, path: str, file_text: str = None) -> str:
2182 +def tool_function(command: str, path: str, file_text: str = None, view_range:
2183 +> Optional[List[int]] = None) -> str:
2184 """
2185 Main tool function that handles:
2186 - - 'view' : View the entire file or directory listing
2187 + - 'view' : View file or directory listing, optionally within line range for files
2188 - 'create': Create a new file with the given file_text
2189 - 'edit' : Overwrite an existing file with file_text
2190 """
2191 @@ -130,7 +187,7 @@ def tool_function(command: str, path: str, file_text: str = None) ->
2192 + str:
2193 path_obj = validate_path(path, command)
2194
2195 if command == "view":
2196 - return view_path(path_obj)
2197 + return view_path(path_obj, view_range)
2198
2199 elif command == "create":
2200 if file_text is None:
2201 @@ -152,4 +209,4 @@ def tool_function(command: str, path: str, file_text: str = None) ->
2202 + str:
2203
2204 if __name__ == "__main__":
2205 # Example usage
2206 - print(tool_function("view", "/home/ubuntu/xx/dgm/coding_agent.py"))
2207 + print(tool_function("view", "/home/ubuntu/xx/dgm/coding_agent.py"))
2208 \ No newline at end of file
2209 diff --git a/tests/test_tools/test_edit.py b/tests/test_tools/test_edit.py
2210 new file mode 100644
2211 index 0000000..04f535b
2212 --- /dev/null
2213 +++
2214 b/tests/test_tools/test_edit.py
2215 @@ -0,0 +1,54 @@
2216 +import pytest
2217 +from pathlib import Path
2218 +from tools.edit import tool_function
2219 +
2220 +def test_view_line_range(tmp_path):
2221 + # Create a test file
2222 + test_file = tmp_path / "test.txt"
2223 + test_content = "line1\nline2\nline3\nline4\nline5\n"
2224 + test_file.write_text(test_content)
2225
2226 + # Test viewing specific line range
2227 + result = tool_function("view", str(test_file), view_range=[2, 4])
2228 + assert "line2" in result
2229 + assert "line3" in result
2230 + assert "line4" in result
2231 + assert "line1" not in result
2232 + assert "line5" not in result
2233 + assert " 2\t" in result # Correct line numbering
2234
2235 + # Test viewing from start to middle
2236 + result = tool_function("view", str(test_file), view_range=[1, 3])
2237 + assert "line1" in result
2238 + assert "line2" in result
2239 + assert "line3" in result
2240 + assert "line4" not in result
2241 + assert " 1\t" in result
2242
2243

```

```
2214
2215 + # Test viewing from middle to end with -1
2216 + result = tool_function("view", str(test_file), view_range=[3, -1])
2217 + assert "line1" not in result
2218 + assert "line2" not in result
2219 + assert "line3" in result
2220 + assert "line4" in result
2221 + assert "line5" in result
2222 + assert " 3\t" in result
2223 +
2224 +def test_view_range_validation(tmp_path):
2225 + # Create a test file
2226 + test_file = tmp_path / "test.txt"
2227 + test_content = "line1\nline2\nline3\n"
2228 + test_file.write_text(test_content)
2229 +
2230 + # Test invalid start line
2231 + result = tool_function("view", str(test_file), view_range=[0, 2])
2232 + assert "Failed to read file range: Start line must be >= 1" in result
2233 +
2234 + # Test invalid range (end < start)
2235 + result = tool_function("view", str(test_file), view_range=[2, 1])
2236 + assert "Failed to read file range: End line must be >= start line or -1" in result
2237 +
2238 +def test_view_range_with_directory(tmp_path):
2239 + # Test that view_range is rejected for directories
2240 + result = tool_function("view", str(tmp_path), view_range=[1, 10])
2241 + assert "Error: view_range is not supported for directory listings" in result
2242 \ No newline at end of file
```

```
2234
2235 diff --git a/tools/edit.py b/tools/edit.py
2236 index 16ae521..757f5c2 100644
2237 --- a/tools/edit.py
2238 +++ b/tools/edit.py
2239 @@ -11,21 +11,21 @@ def tool_info():
2240 * If `path` is a directory, `view` lists non-hidden files and directories up to 2 levels
2241 * ↪ deep.\n
2242 * The `create` command cannot be used if the specified `path` already exists as a file.\n
2243 * If a `command` generates a long output, it will be truncated and marked with `<response
2244 * ↪ clipped>`.\n
2245 ** The `edit` command overwrites the entire file with the provided `file_text`."",
2246 ** The `str_replace` command replaces a unique occurrence of old_str with new_str, failing
2247 * ↪ if old_str is not found or appears multiple times."",
2248 "input_schema": {
2249 "type": "object",
2250 "properties": {
2251 "command": {
2252 "type": "string",
2253 },
2254 "enum": ["view", "create", "edit"],
2255 "description": "The command to run: `view`, `create`, or `edit`."
2256 },
2257 "path": {
2258 "description": "Absolute path to file or directory, e.g. `/repo/file.py`"
2259 * ↪ or `~/repo`.",
2260 "type": "string"
2261 },
2262 "file_text": {
2263 "description": "Required parameter of `create` or `edit` command, containing the content
2264 * ↪ for the entire file.",
2265 "type": "string"
2266 },
2267 "view_range": {
2268 "items": {"type": "integer"},
2269 "minItems": 2,
2270 "maxItems": 2
2271 },
2272 "old_str": {
2273 "description": "Required parameter of `str_replace` command, containing the exact text
2274 * ↪ to find and replace.",
2275 "type": "string"
2276 },
2277 "new_str": {
2278 "description": "Required parameter of `str_replace` command, containing the new text to
2279 * ↪ replace old_str with.",
2280 "type": "string"
2281 }
2282 }
2283
```

```

2268
2269 },
2270 "required": ["command", "path"]
2271 @@ -51,7 +59,7 @@ def validate_path(path: str, command: str) -> Path:
2272 Validate the file path for each command:
2273 - 'view': path may be a file or directory; must exist.
2274 - 'create': path must not exist (for new file creation).
2275 - - 'edit': path must exist (for overwriting).
2276 + - 'str_replace': path must exist and be a file.
2277 """
2278 path_obj = Path(path)
2279
2280 @@ -69,7 +77,7 @@ def validate_path(path: str, command: str) -> Path:
2281 # Path must not exist
2282 if path_obj.exists():
2283 raise ValueError(f"Cannot create new file; {path} already exists.")
2284 - elif command == "edit":
2285 + elif command == "str_replace":
2286 # Path must exist and must be a file
2287 if not path_obj.exists():
2288 raise ValueError(f"The file {path} does not exist.")
2289 @@ -144,6 +152,28 @@ def write_file(path: Path, content: str):
2290 except Exception as e:
2291 raise ValueError(f"Failed to write file: {e}")
2292
2293 +def str_replace_in_file(path: Path, old_str: str, new_str: str) -> str:
2294 + """
2295 + Replace an exact occurrence of old_str with new_str in the file.
2296 + Only performs the replacement if old_str occurs exactly once.
2297 + Returns a message indicating success or failure.
2298 + """
2299 + try:
2300 + content = read_file(path)
2301 + occurrences = content.count(old_str)
2302 + if occurrences == 0:
2303 + return f"Error: Could not find the exact text to replace in {path}"
2304 + elif occurrences > 1:
2305 + return f"Error: Found multiple ({occurrences}) occurrences of the text in {path}. Must
2306 + ↪ be unique."
2307 + else:
2308 + new_content = content.replace(old_str, new_str)
2309 + write_file(path, new_content)
2310 + return f"Successfully replaced text in {path}"
2311 + except Exception as e:
2312 + return f"Error during string replacement: {e}"
2313 + """
2314 def view_path(path_obj: Path, view_range: Optional[List[int]] = None) -> str:
2315 """
2316 View the file contents (optionally within a range) or directory listing.
2317 @@ -176,12 +206,13 @@ def view_path(path_obj: Path, view_range: Optional[List[int]] =
2318 ↪ None) -> str:
2319 content, start_line = read_file_range(path_obj, view_range)
2320 return format_output(content, str(path_obj), start_line)
2321
2322 -def tool_function(command: str, path: str, file_text: str = None, view_range:
2323 ↪ Optional[List[int]] = None) -> str:
2324 +def tool_function(command: str, path: str, file_text: str = None, view_range:
2325 ↪ Optional[List[int]] = None,
2326 + old_str: str = None, new_str: str = None) -> str:
2327 """
2328 Main tool function that handles:
2329 - - 'view' : View file or directory listing, optionally within line range for files
2330 - - 'create': Create a new file with the given file_text
2331 - - 'edit' : Overwrite an existing file with file_text
2332 + - 'view' : View file or directory listing, optionally within line range for files
2333 + - 'create' : Create a new file with the given file_text
2334 + - 'str_replace': Replace exact occurrence of old_str with new_str in the file
2335 """
2336 try:
2337 path_obj = validate_path(path, command)
2338 @@ -195,11 +226,10 @@ def tool_function(command: str, path: str, file_text: str = None,
2339 ↪ view_range: Op
2340 write_file(path_obj, file_text)
2341 return f"File created successfully at: {path}"
2342
2343 - elif command == "edit":
2344 - if file_text is None:
2345 raise ValueError("Missing required `file_text` for 'edit' command.")
2346 - write_file(path_obj, file_text)
2347 - return f"File at {path} has been overwritten with new content."

```

```

2322
2323 + elif command == "str_replace":
2324 + if old_str is None or new_str is None:
2325 + raise ValueError("Missing required `old_str` and/or `new_str` for 'str_replace'"
2326 + " → command.")
2327 + return str_replace_in_file(path_obj, old_str, new_str)
2328
2329 else:
2330 raise ValueError(f"Unknown command: {command}")
2331
2332 diff --git a/tests/__init__.py b/tests/__init__.py
2333 new file mode 100644
2334 index 0000000..e69de29
2335 diff --git a/tests/test_tools.py b/tests/test_tools.py
2336 new file mode 100644
2337 index 0000000..c7f242f
2338 --- /dev/null
2339 +++ b/tests/test_tools.py
2340 @@ -0,0 +1,65 @@
2341 +import pytest
2342 +from pathlib import Path
2343 +from tools.edit import tool_function
2344
2345 +# Test fixtures
2346 +@pytest.fixture
2347 +def temp_file(tmp_path):
2348 + file_path = tmp_path / "test.txt"
2349 + content = "line 1\nline 2\nline 3\n"
2350 + file_path.write_text(content)
2351 + return str(file_path)
2352
2353 +def test_str_replace_success(temp_file):
2354 + # Test successful replacement
2355 + result = tool_function(
2356 + command="str_replace",
2357 + path=temp_file,
2358 + old_str="line 2\n",
2359 + new_str="replaced line\n"
2360 +)
2361 + assert "Successfully replaced" in result
2362 + assert Path(temp_file).read_text() == "line 1\nreplaced line\nline 3\n"
2363
2364 +def test_str_replace_not_found(temp_file):
2365 + # Test when old_str is not found
2366 + result = tool_function(
2367 + command="str_replace",
2368 + path=temp_file,
2369 + old_str="nonexistent",
2370 + new_str="something"
2371 +)
2372 + assert "Could not find" in result
2373 + # Original file should be unchanged
2374 + assert Path(temp_file).read_text() == "line 1\nline 2\nline 3\n"
2375
2376 +def test_str_replace_multiple_occurrences(temp_file):
2377 + # First create a file with multiple occurrences
2378 + Path(temp_file).write_text("same\nsame\nsame\n")
2379 + result = tool_function(
2380 + command="str_replace",
2381 + path=temp_file,
2382 + old_str="same\n",
2383 + new_str="different\n"
2384 +)
2385 + assert "multiple" in result
2386 + # Original file should be unchanged
2387 + assert Path(temp_file).read_text() == "same\nsame\nsame\n"
2388
2389 +def test_str_replace_missing_params(temp_file):
2390 + # Test missing parameters
2391 + result = tool_function(
2392 + command="str_replace",
2393 + path=temp_file,
2394 +)
2395 + assert "Missing required" in result
2396
2397 +def test_str_replace_invalid_path():
2398 + # Test with non-existent file
2399 + result = tool_function(
2400 + command="str_replace",
2401 + path="/nonexistent/path",
2402 + old_str="old",
2403 + new_str="new"
2404 +)

```

```

2376
2377 + assert "does not exist" in result
2378 \ No newline at end of file
2379
2379 diff --git a/llm_withtools.py b/llm_withtools.py
2380 index d1394bb..6cc3604 100644
2381 --- a/llm_withtools.py
2382 +++ b/llm_withtools.py
2383 @@ -29,7 +29,7 @@ def process_tool_call(tools_dict, tool_name, tool_input):
2384)
2385 def get_response_withtools(
2386 client, model, messages, tools, tool_choice,
2387 - logging=None,
2388 + logging=None, system_message=None,
2389):
2390 try:
2391 if 'claude' in model:
2392 -52,13 +52,32 @@ def get_response_withtools(
2393 raise ValueError(f"Unsupported model: {model}")
2394 return response
2395 except Exception as e:
2396 - logging(f"Error in get_response_withtools: {str(e)}")
2397 + error_msg = str(e)
2398 + logging(f"Error in get_response_withtools: {error_msg}")
2399
2400 # Hitting the context window limit
2401 - if 'Input is too long for requested model' in str(e):
2402 - pass
2403 + if 'Input is too long for requested model' in error_msg or 'maximum context length' in
2404 error_msg:
2405 + if not system_message:
2406 + # Extract system message from the first message if available
2407 + system_message = messages[0].get('content', '') if messages else ''
2408 + if isinstance(system_message, list):
2409 + system_message = ' '.join(block['text'] for block in system_message if block['type'] ==
2410 'text')
2411
2412 + # Summarize the conversation history
2413 + summarized_messages = summarize_messages(client, model, messages, system_message)
2414
2415 + # Retry with summarized messages
2416 + return get_response_withtools(
2417 + client=client,
2418 + model=model,
2419 + messages=summarized_messages,
2420 + tools=tools,
2421 + tool_choice=tool_choice,
2422 + logging=logging,
2423 + system_message=system_message
2424 +)
2425
2426 - raise # Re-raise the exception after logging
2427 + raise # Re-raise other exceptions
2428
2429 def check_for_tool_use(response, model=''):
2430 """
2431 @@ -247,6 +266,57 @@ def convert_msg_history_openai(msg_history):
2432
2433 return new_msg_history
2434
2435 +def summarize_messages(client, model, messages, system_message):
2436 + """
2437 + Creates a condensed summary of older messages while preserving recent context.
2438 + Only summarizes assistant and user messages, keeps tool results as is for accuracy.
2439 + """
2440
2441 + # Keep the most recent messages intact
2442 + recent_msgs = messages[-2:] if len(messages) > 2 else messages
2443 + if len(messages) <= 2:
2444 + return messages
2445
2446 + # Prepare messages to be summarized
2447 + msgs_to_summarize = messages[:-2]
2448
2449 + # Create a prompt to summarize the conversation
2450 + summary_request = "Please create a concise summary of this conversation that preserves
2451 the key context and important details:"
2452 + for msg in msgs_to_summarize:
2453 + if isinstance(msg.get('content', ''), list):
2454 + content = ' '.join(block['text'] for block in msg['content'] if block['type'] == 'text')
2455 + else:
2456 + content = str(msg.get('content', ''))
```

```

2430
2431 + if msg.get('role') in ['assistant', 'user']:
2432 + summary_request += f"\n{msg['role']}: {content}"
2433 +
2434 + try:
2435 + # Get summary from the model
2436 + summary_response, _ = get_response_from_llm(
2437 + msg=summary_request,
2438 + client=client,
2439 + model=model,
2440 + system_message="You are a summarizer. Create a concise but informative summary.",
2441 + print_debug=False,
2442 + msg_history=[]
2443 +)
2444 +
2445 + # Create new message history with the summary
2446 + summarized_history = [
2447 + "role": "system",
2448 + "content": [{"type": "text", "text": system_message}]
2449 +],
2450 + "role": "assistant",
2451 + "content": [{"type": "text", "text": f"Previous conversation summary:
2452 + ↪ {summary_response}"}]
2453 +
2454 + # Add back the recent messages
2455 + summarized_history.extend(recent_msgs)
2456 +
2457 + return summarized_history
2458 + except Exception:
2459 + # If summarization fails, return original messages with the most recent ones
2460 + return [messages[0]] + recent_msgs
2461 +
2462 def convert_msg_history(msg_history, model=None):
2463 """
2464 Convert message history from the model-specific format to a generic format.
2465 @@ -263,7 +333,14 @@ def chat_with_agent_manualtools(msg, model, msg_history=None,
2466 ↪ logging=logging):
2467 if msg_history is None:
2468 msg_history = []
2469 system_message = f'You are a coding agent.\n\n{get_tooluse_prompt()}''
2470 - new_msg_history = msg_history
2471 + new_msg_history = msg_history.copy() if msg_history else []
2472 +
2473 # Ensure system message is the first message in history
2474 + if not new_msg_history or new_msg_history[0].get('role') != 'system':
2475 new_msg_history.insert(0, {
2476 "role": "system",
2477 "content": [{"type": "text", "text": system_message}]}
2478 +
2479 try:
2480 # Load all tools
2481
2482 diff --git a/coding_agent.py b/coding_agent.py
2483 index 9a2cc2f..3f1bc1d 100644
2484 --- a/coding_agent.py
2485 +++
2486 b/coding_agent.py
2487 @@ -111,6 +111,7 @@ class AgenticSystem:
2488 self_improve=False,
2489 instance_id=None,
2490 max_retries=3,
2491 + num_candidates=3,
2492):
2493 self.problem_statement = problem_statement
2494 self.git_tempdir = git_tempdir
2495 @@ -121,6 +122,7 @@ class AgenticSystem:
2496 self.instance_id = instance_id if not self_improve else 'dgm'
2497 self.code_model = CLAUDE_MODEL
2498 self.max_retries = max_retries
2499 + self.num_candidates = num_candidates
2500
2501 # Initialize logger and store it in thread-local storage
2502 self.logger = setup_logger(chat_history_file)
2503 @@ -190,7 +192,7 @@ Your task is to run the regression tests in the {self.git_tempdir}
2504 ↪ directory to
2505
2506 def forward(self):
2507 """
2508 - The forward function for the AgenticSystem.

```

```

2484
2485 + The forward function for the AgenticSystem that generates and evaluates multiple
2486 + candidate patches.
2487 """
2488 base_instruction = f"""I have uploaded a Python code repository in the directory
2489 ↪ {self.git_tempdir}. Help solve the following problem.
2490
2491 @@ -205,10 +207,18 @@ Your task is to run the regression tests in the {self.git_tempdir}
2492 ↪ directory to
2493 Your task is to make changes to the files in the {self.git_tempdir} directory to address
2494 ↪ the <problem_description>. I have already taken care of the required dependencies.
2495 """
2496
2497 + # Get regression tests summary once at the start
2498 + regression_tests_summary = self.get_regression_tests()
2499 +
2500 + # Lists to store candidates
2501 + valid_patches = []
2502 + valid_reports = []
2503 +
2504 retry_count = 0
2505 - while retry_count < self.max_retries:
2506 + while retry_count < self.max_retries and len(valid_patches) < self.num_candidates:
2507 safe_log(f"\n==== Attempt {retry_count + 1} of {self.max_retries} ====")
2508
2509 + safe_log(f"Valid solutions so far: {len(valid_patches)} of {self.num_candidates}
2510 ↪ desired")
2511 +
2512 # Reset to base commit before each attempt
2513 if retry_count > 0:
2514 reset_to_commit(self.git_tempdir, self.base_commit)
2515 @@ -216,8 +226,8 @@ Your task is to make changes to the files in the {self.git_tempdir}
2516 ↪ directory to
2517 # Add retry context to instruction if this is a retry attempt
2518 instruction = base_instruction
2519 if retry_count > 0:
2520 - instruction += f"""\nNOTE: Previous attempt(s) failed because they either produced empty
2521 ↪ patches or only modified test files.
2522 -Please ensure your solution includes changes to the main source code files, not just test
2523 ↪ files."""
2524 + instruction += f"""\nNOTE: Previous attempt(s) did not produce enough valid solutions.
2525 -Please provide a different approach to solve the problem. Your solution must include
2526 ↪ changes to the main source code files, not just test files."""
2527
2528 # Run the agent
2529 new_msg_history = chat_with_agent(instruction, model=self.code_model,
2530 ↪ msg_history=[], logging=safe_log)
2531 @@ -228,16 +238,45 @@ Please ensure your solution includes changes to the main source code
2532 ↪ files, not
2533
2534 if is_valid:
2535 safe_log(f"Valid patch generated: {reason}")
2536 - break
2537 + # Run regression tests for this candidate
2538 + test_report = self.run_regression_tests(regression_tests_summary)
2539 + test_score = get_report_score(test_report)
2540 + safe_log(f"Test score: {test_score}")
2541 +
2542 + valid_patches.append(patch)
2543 + valid_reports.append(test_report)
2544 +
2545 + if len(valid_patches) >= self.num_candidates:
2546 + break
2547 else:
2548 safe_log(f"Invalid patch: {reason}")
2549 - if retry_count < self.max_retries - 1:
2550 - safe_log("Retrying with a new attempt...")
2551 - else:
2552 - safe_log("Maximum retries reached. Unable to generate a valid patch.")
2553
2554 retry_count += 1
2555
2556 + if not valid_patches:
2557 + safe_log("Failed to generate any valid patches.")
2558 + return
2559 +
2560 + # Use score_tie_breaker to select the best patch
2561 + safe_log(f"\n==== Selecting Best Solution from {len(valid_patches)} Candidates ====")
2562 + best_index = score_tie_breaker(
2563 + self.problem_statement,
2564 + valid_patches,
2565 + valid_reports,

```

```

2538 + logging=safe_log
2539 +)
2540 +
2541 + # Reset to base and apply the best patch
2542 + reset_to_commit(self.git_tempdir, self.base_commit)
2543 + best_patch = valid_patches[best_index]
2544 + safe_log(f"\n==== Applying Best Solution (Candidate {best_index + 1}) ====")
2545 + apply_patch(self.git_tempdir, best_patch)
2546 +
2547 + # Final validation of the selected patch
2548 + final_test_report = self.run_regression_tests(regression_tests_summary)
2549 + final_score = get_report_score(final_test_report)
2550 + safe_log(f"Final solution test score: {final_score}")
2551 +
2552 def main():
2553 parser = argparse.ArgumentParser(description='Process repository with an agentic
2554 ↪ system.')
2555 parser.add_argument('--problem_statement', required=True, help='The problem statement
2556 ↪ to process')
2557 @@ -249,6 +288,7 @@ def main():
2558 parser.add_argument('--self_improve', default=False, action='store_true',
2559 ↪ help='Whether to self-improve the repository or solving swe')
2560 parser.add_argument('--instance_id', default=None, help='Instance ID for SWE issue')
2561 parser.add_argument('--max_retries', type=int, default=3, help='Maximum number of
2562 ↪ patch generation attempts')
2563 + parser.add_argument('--num_candidates', type=int, default=3, help='Number of candidate
2564 ↪ solutions to generate')
2565 args = parser.parse_args()
2566
2567 # Process the repository
2568 @@ -261,6 +301,7 @@ def main():
2569 self_improve=args.self_improve,
2570 instance_id=args.instance_id,
2571 max_retries=args.max_retries,
2572 + num_candidates=args.num_candidates,
2573)
2574
2575 # Run the agentic system to try to solve the problem
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3299
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3349
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3369
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3379
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3389
3389
3390
3391
3392
3393
3394
3395
3396
3397
3398
3399
3399
3400
3401
3402
3403
3404
3405
3406
3407
3408
3409
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419
3419
3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3459
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3469
3469
3470
3471
3472
3473
3474
3475
3476
3477
3478
3479
3479
3480
3481
3482
3483
3484
3485
3486
3487
3488
3489
3489
3490
3491
3492
3493
3494
3495
3496
3497
3498
3498
3499
3499
3500
3501
3502
3503
3504
3505
3506
3507
3508
3509
3509
3510
3511
3512
3513
3514
3515
3516
3517
3518
3519
3519
3520
3521
3522
3523
3524
3525
3526
3527
3528
3529
3529
3530
3531
3532
3533
3534
3535
3536
3537
3538
3539
3539
3540
3541
3542
3543
3544
3545
3546
3547
3548
3549
3549
3550
3551
3552
3553
3554
3555
3556
3557
3558
3559
3559
3560
3561
3562
3563
3564
3565
3566
3567
3568
3569
3569
3570
3571
3572
3573
3574
3575
3576
3577
3578
3579
3579
3580
3581
3582
3583
3584
3585
3586
3587
3588
3589
3589
3590
3591
3592
3593
3594
3595
3596
3597
3598
3598
3599
3599
3600
3601
3602
3603
3604
3605
3606
3607
3608
3609
3609
3610
3611
3612
3613
3614
3615
3616
3617
3618
3619
3619
3620
3621
3622
3623
3624
3625
3626
3627
3628
3629
3629
3630
3631
3632
3633
3634
3635
3636
3637
3638
3639
3639
3640
3641
3642
3643
3644
3645
3646
3647
3648
3649
3649
3650
3651
3652
3653
3654
3655
3656
3657
3658
3659
3659
3660
3661
3662
3663
3664
3665
3666
3667
3668
3669
3669
3670
3671
3672
3673
3674
3675
3676
3677
3678
3679
3679
3680
3681
3682
3683
3684
3685
3686
3687
3688
3689
3689
3690
3691
3692
3693
3694
3695
3696
3697
3698
3698
3699
3699
3700
3701
3702
3703
3704
3705
3706
3707
3708
3709
3709
3710
3711
3712
3713
3714
3715
3716
3717
3718
3719
3719
3720
3721
3722
3723
3724
3725
3726
3727
3728
3729
3729
3730
3731
3732
3733
3734
3735
3736
3737
3738
3739
3739
3740
3741
3742
3743
3744
3745
3746
3747
3748
3749
3749
3750
3751
3752
3753
3754
3755
3756
3757
3758
3759
3759
3760
3761
3762
3763
3764
3765
3766
3767
3768
3769
3769
3770
3771
3772
3773
3774
3775
3776
3777
3778
3779
3779
3780
3781
3782
3783
3784
3785
3786
3787
3788
3789
3789
3790
3791
3792
3793
3794
3795
3796
3797
3798
3798
3799
3799
3800
3801
3802
3803
3804
3805
3806
3807
3808
3809
3809
3810
3811
3812
3813
3814
3815
3816
3817
3818
3819
3819
3820
3821
3822
3823
3824
3825
3826
3827
3828
3829
3829
3830
3831
3832
3833
3834
3835
3836
3837
3838
3839
3839
3840
3841
3842
3843
3844
3845
3846
3847
3848
3849
3849
3850
3851
3852
3853
3854
3855
3856
3857
3858
3859
3859
3860
3861
3862
3863
3864
3865
3866
3867
3868
3869
3869
3870
3871
3872
3873
3874
3875
3876
3877
3878
3879
3879
3880
3881
3882
3883
3884
3885
3886
3887
3888
3889
3889
3890
3891
3892
3893
3894
3895
3896
3897
3898
3898
3899
3899
3900
3901
3902
3903
3904
3905
3906
3907
3908
3909
3909
3910
3911
3912
3913
3914
3915
3916
3917
3918
3919
3919
3920
3921
3922
3923
3924
3925
3926
3927
3928
3929
3929
3930
3931
3932
3933
3934
3935
3936
3937
3938
3939
3939
3940
3941
3942
3943
3944
3945
3946
3947
3948
3949
3949
3950
3951
3952
3953
3954
3955
3956
3957
3958
3959
3959
3960
3961
3962
3963
3964
3965
3966
3967
3968
3969
3969
3970
3971
3972
3973
3974
3975
3976
3977
3978
3979
3979
3980
3981
3982
3983
3984
3985
3986
3987
3988
3989
3989
3990
3991
3992
3993
3994
3995
3996
3997
3998
3998
3999
3999
4000
4001
4002
4003
4004
4005
4006
4007
4008
4009
4009
4010
4011
4012
4013
4014
4015
4016
4017
4018
4019
4019
4020
4021
4022
4023
4024
4025
4026
4027
4028
4029
4029
4030
4031
4032
4033
4034
4035
4036
4037
4038
4039
4039
4040
4041
4042
4043
4044
4045
4046
4047
4048
4049
4049
4050
4051
4052
4053
4054
4055
4056
4057
4058
4059
4059
4060
4061
4062
4063
4064
4065
4066
4067
4068
4069
4069
4070
4071
4072
4073
4074
4075
4076
4077
4078
4079
4079
4080
4081
4082
4083
4084
4085
4086
4087
4088
4089
4089
4090
4091
4092
4093
4094
4095
4096
4097
4098
4098
4099
4099
4100
4101
4102
4103
4104
4105
4106
4107
4108
4109
4109
4110
4111
4112
4113
4114
4115
4116
4117
4118
4119
4119
4120
4121
4122
4123
4124
4125
4126
4127
4128
4129
4129
4130
4131
4132
4133
4134
4135
4136
4137
4138
4139
4139
4140
4141
4142
4143
4144
4145
4146
4147
4148
4149
4149
4150
4151
4152
4153
4154
4155
4156
4157
4158
4159
4159
4160
4161
4162
4163
4164
4165
4166
4167
4168
4169
4169
4170
4171
4172
4173
4174
4175
4176
4177
4178
4179
4179
4180
4181
4182
4183
4184
4185
4186
4187
4188
4189
4189
4190
4191
4192
4193
4194
4195
4196
4197
4198
4198
4199
4199
4200
4201
4202
4203
4204
4205
4206
4207
4208
4209
4209
4210
4211
4212
4213
4214
4215
4216
4217
4218
4219
4219
4220
4221
4222
4223
4224
4225
4226
4227
4228
4229
4229
4230
4231
4232
4233
4234
4235
4236
4237
4238
4239
4239
4240
4241
4242
4243
4244
4245
4246
4247
4248
4249
4249
4250
4251
4252
4253
4254
4255
4256
4257
4258
4259
4259
4260
4261
4262
4263
4264
4265
4266
4267
4268
4269
4269
4270
4271
4272
4273
4274
4275
4276
4277
4278
4279
4279
4280
4281
4282
4283
4284
4285
4286
4287
4288
4289
4289
4290
4291
4292
4293
4294
4295
4296
4297
4298
4298
4299
4299
4300
4301
4302
4303
4304
4305
4306
4307
4308
4309
4309
4310
4311
4312
4313
4314
4315
4316
4317
4318
4319
4319
4320
4321
4322
4323
4324
4325
4326
4327
4328
4329
4329
4330
4331
4332
4333
4334
4335
4336
4337
4338
4339
4339
4340
4341
4342
4343
4344
4345
4346
4347
4348
4349
4349
4350
4351
4352
4353
4354
4355
4356
4357
4358
4359
4359
4360
4361
4362
4363
4364
4365
4366
4367
4368
4369
4369
4370
4371
4372
4373
4374
4375
4376
4377
4378
4379
4379
4380
4381
4382
4383
4384
4385
4386
4387
4388
4389
4389
4390
4391
4392
4393
4394
4395
4396
4397
4398
4398
4399
4399
4400
4401
4402
4403
4404
4405
4406
4407
4408
4409
4409
4410
4411
4412
4413
4414
4415
4416
4417
4418
4419
4419
4420
4421
4422
4423
4424
4425
4426
4427
4428
4429
4429
4430
4431
4432
4433
4434
4435
4436
4437
4438
4439
4439
4440
4441
4442
4443
4444
4445
4446
4447
4448
4449
4449
4450
4451
4452
4453
4454
4455
4456
4457
4458
4459
4459
4460
4461
4462
4463
4464
4465
4466
4467
4468
4469
4469
4470
4471
4472
4473
4474
4475
4476
4477
4478
4479
4479
4480
4481
4482
4483
4484
4485
4486
4487
4488
4489
4489
4490
4491
4492
4493
4494
4495
4496
4497
4498
4498
4499
4499
4500
4501
4502
4503
4504
4505
4506
4507
4508
4509
4509
4510
4511
4512
4513
4514
4515
4516
4517
4518
4519
4519
4520
4521
4522
4523
4524
4525
4526
4527
4528
4529
4529
4530
4531
4532
4533
4534
4535
4536
4537
4538
4539
4539
4540
4541
4542
4543
4544
4545
4546
4547
4548
4549
4549
4550
4551
4552
4553
4554
4555
4556
4557
4558
4559
4559
4560
4561
4562
4563
4564
4565
4566
4567
4568
4569
4569
4570
4571
4572
4573
4574
4575
4576
4577
4578
4579
4579
4580
4581
4582
4583
4584
4585
4586
4587
4588
4589
4589
4590
4591
4592
4593
4594
4595
4596
4597
4598
4598
4599
4599
4600
4601
4602
4603
4604
4605
4606
4607
4608
4609
4609
4610
4611
4612
4613
4614
4615
4616
4617
4618
4619
4619
4620
4621
4622
4623
4624
4625
4626
4627
4628
4629
4629
4630
4631
4632
4633
4634
4635
4636
4637
4638
4639
4639
4640
4641
4642
4643
4644
4645
4646
4647
4648
4649
4649
4650
4651
4652
4653
4654
4655
4656
4657
4658
4659
4659
4660
4661
4662
4663
4664
4665
4666
4667
4668
4669
4669
4670
4671
4672
4673
4674
4675
4676
4677
4678
4679
4679
4680
4681
4682
4683
4684
4685
4686
4687
4688
4689
4689
4690
4691
4692
4693
4694
4695
4696
4697
4698
4698
4699
4699
4700
4701
4702
4703
4704
4705
4706
4707
4708
4709
4709
4710
4711
4712
4713
4714
4715
4716
4717
4718
4719
4719
4720
4721
4722
4723
4724
4725
4726
4727
4728
4729
4729
4730
4731
4732
4733
4734
4735
4736
4737
4738
4739
4739
4740
4741
4742
4743
4744
4745
4746
4747
4748
4749
4749
4750
4751
4752
4753
4754
4755
4756
4757
4758
4759
4759
4760
4761
4762
4
```

```

2592
2593 while retry_count < self.max_retries and len(valid_patches) < self.num_candidates:
2594 safe_log(f"\n==== Attempt {retry_count + 1} of {self.max_retries} ===")
2594 safe_log(f"Valid solutions so far: {len(valid_patches)} of {self.num_candidates}")
2594 ↪ desired")
2595 + safe_log(f"Current best test score: {best_score}")
2596
2597 # Reset to base commit before each attempt
2597 if retry_count > 0:
2598 reset_to_commit(self.git_tempdir, self.base_commit)
2599
2600 - # Add retry context to instruction if this is a retry attempt
2600 - instruction = base_instruction
2600 - if retry_count > 0:
2601 - instruction += f"""\nNOTE: Previous attempt(s) did not produce enough valid solutions.
2602 + # Construct instruction with previous best solutions if available
2602 + instruction = f"""I have uploaded a Python code repository in the directory
2603 ↪ {self.git_tempdir}. Help solve the following problem.
2604
2605 +<problem_description>
2605 +(self.problem_statement)
2606 +</problem_description>
2607
2608 +<test_description>
2608 +(self.test_description)
2609 +</test_description>"""
2609
2610 + # Add previous solutions context if available
2610 + if valid_patches and retry_count > 0:
2611 + previous_solutions = []
2612 + for i, (patch, report, score) in enumerate(zip(valid_patches, valid_reports,
2613 ↪ valid_scores)):
2614 + previous_solutions.append(f"""
2614 +Previous Solution {i+1}:
2615 +<code_changes>
2615 +{patch}
2616 +</code_changes>
2617 +Test Score: {score}
2618 +Test Report: {report}
2619 +""")
2620 + instruction += "\n\nPrevious solution attempts:\n" + "\n".join(previous_solutions)
2620 + instruction += "\nPlease provide a new solution that addresses any limitations in the
2620 ↪ previous attempts or explores a different approach."
2621 + elif retry_count > 0:
2621 + instruction += """\nNOTE: Previous attempt(s) did not produce enough valid solutions.
2622 Please provide a different approach to solve the problem. Your solution must include
2622 ↪ changes to the main source code files, not just test files."""
2623
2624 + instruction += f"\n\nYour task is to make changes to the files in the {self.git_tempdir}
2624 ↪ directory to address the <problem_description>. I have already taken care of the
2625 ↪ required dependencies."
2626
2627 # Run the agent
2627 new_msg_history = chat_with_agent(instruction, model=self.code_model,
2628 ↪ msg_history=[], logging=safe_log)
2629
2629 @@ -245,6 +262,14 @@ Please provide a different approach to solve the problem. Your
2630 ↪ solution must inc
2631
2632 valid_patches.append(patch)
2632 valid_reports.append(test_report)
2633
2634 + # Update best score and indices
2634 + if test_score > best_score:
2635 + best_score = test_score
2636 + best_patches_indices = [len(valid_patches) - 1]
2637 + elif test_score == best_score:
2637 + best_patches_indices.append(len(valid_patches) - 1)
2638
2638 if len(valid_patches) >= self.num_candidates:
2639 break
2640
2640 @@ -257,25 +282,30 @@ Please provide a different approach to solve the problem. Your
2640 ↪ solution must inc
2641 safe_log("Failed to generate any valid patches.")
2642 return
2643
2643 - # Use score_tie_breaker to select the best patch
2644 + # Only use tie-breaker if we have multiple patches with the best score
2644 safe_log(f"\n==== Selecting Best Solution from {len(valid_patches)} Candidates ===")
2645 - best_index = score_tie_breaker(
2645 - self.problem_statement,

```

```

2646
2647 - valid_patches,
2648 - valid_reports,
2649 - logging=safe_log
2650 -)
2651 + if len(best_patches_indices) > 1:
2652 + safe_log(f"Multiple solutions ({len(best_patches_indices)}) tied for best score
2653 ↪ {best_score}. Using tie-breaker.")
2654 + best_index = score_tie_breaker(
2655 + self.problem_statement,
2656 + [valid_patches[i] for i in best_patches_indices],
2657 + [valid_reports[i] for i in best_patches_indices],
2658 + logging=safe_log
2659 +)
2660 + best_index = best_patches_indices[best_index]
2661 + else:
2662 + best_index = best_patches_indices[0]
2663
2664 # Reset to base and apply the best patch
2665 reset_to_commit(self.git_tempdir, self.base_commit)
2666 best_patch = valid_patches[best_index]
2667 - safe_log(f"\n==== Applying Best Solution (Candidate {best_index + 1}) ===")
2668 + safe_log(f"\n==== Applying Best Solution (Candidate {best_index + 1}) with score
2669 ↪ {valid_scores[best_index]} ===")
2670 apply_patch(self.git_tempdir, best_patch)
2671
2672 # Final validation of the selected patch
2673 final_test_report = self.run_regression_tests(regression_tests_summary)
2674 final_score = get_report_score(final_test_report)
2675 - safe_log(f"Final solution test score: {final_score}")
2676 + safe_log(f"Final validation test score: {final_score}")
2677
2678 def main():
2679 parser = argparse.ArgumentParser(description='Process repository with an agentic
2680 ↪ system.')
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699

```

## F.2 DGM ON POLYGLOT

Diff patches contributing to the best agent discovered by the DGM on Polyglot:

```

2675 diff --git a/coding_agent.py b/coding_agent.py
2676 index 04ffb36..6639abd 100644
2677 --- a/coding_agent.py
2678 +++ b/coding_agent.py
2679 @@ -4,6 +4,9 @@ import logging
2680 from logging.handlers import RotatingFileHandler
2681 import os
2682 import threading
2683 +import json
2684 +from dataclasses import dataclass
2685 +from typing import List, Optional
2686
2687 from llm_witertools import CLAUDE_MODEL, OPENAI_MODEL, chat_with_agent
2688 from utils.git_utils import diff_versus_commit, reset_to_commit, apply_patch
2689 @@ -42,6 +45,14 @@ TEST_COMMANDS = {
2690 # Thread-local storage for logger instances
2691 thread_local = threading.local()
2692
2693 +@dataclass
2694 +class SolutionAttempt:
2695 + """Class to store information about a solution attempt."""
2696 + patch: str # The patch content
2697 + test_output: str # Raw test output
2698 + test_success: bool # Whether tests passed
2699 + test_stats: dict # Test statistics (e.g., number of passed/failed tests)
2700
2701 def get_thread_logger():
2702 """
2703 Get the logger instance specific to the current thread.
2704 @@ -102,7 +113,8 @@ class AgenticSystem:
2705 chat_history_file='./chat_history.md',
2706 test_description=None,
2707 self_improve=False,
2708 - language='python'
2709 + language='python',
2710 + max_attempts=3
2711):
2712 self.problem_statement = problem_statement
2713 self.git_tempdir = git_tempdir

```

```

2700
2701 @@ -111,6 +123,7 @@ class AgenticSystem:
2702 self.test_description = test_description
2703 self.self_improve = self_improve
2704 self.language = language
2705 + self.max_attempts = max_attempts
2706
2707 # Set the code model based on whether self-improvement is enabled
2708 self.code_model = OPENAI_MODEL if not self_improve else CLAUDE_MODEL
2709 @@ -137,11 +150,63 @@ class AgenticSystem:
2710]
2711
2712 return new_msg_history
2713
2714 + def run_tests(self) -> tuple[bool, str, dict]:
2715 + """Run tests and return success status, output, and test statistics."""
2716 + success = False
2717 + output = ""
2718 + stats = {"passed": 0, "failed": 0, "errors": 0, "total": 0}
2719
2720 + try:
2721 + for command in TEST_COMMANDS.get(self.language, []):
2722 + proc = subprocess.run(
2723 + command,
2724 + cwd=self.git_tempdir,
2725 + capture_output=True,
2726 + text=True,
2727 + check=False
2728)
2729 + output += f"${' '.join(command)}\n{proc.stdout}\n{proc.stderr}\n"
2730 + success = proc.returncode == 0
2731 + if not success:
2732 + break
2733
2734 + # Try to extract test statistics from output
2735 + # This is a simple example; you might want to add more sophisticated parsing
2736 + stats["passed"] = output.count("PASS") + output.count("ok")
2737 + stats["failed"] = output.count("FAIL") + output.count("not ok")
2738 + stats["errors"] = output.count("ERROR") + output.count("panic:")
2739 + stats["total"] = stats["passed"] + stats["failed"] + stats["errors"]
2740
2741 + except Exception as e:
2742 + output = f"Error running tests: {str(e)}"
2743 + success = False
2744
2745 + return success, output, stats
2746
2747 + def analyze_test_results(self, attempts: List[SolutionAttempt]) -> str:
2748 + """Analyze test results and create a summary for the agent."""
2749 + summary = "# Test Results Analysis\n\n"
2750
2751 + for i, attempt in enumerate(attempts, 1):
2752 + summary += f"## Attempt {i}\n"
2753 + summary += f"Test Success: {attempt.test_success}\n"
2754 + summary += f"Test Stats: {json.dumps(attempt.test_stats, indent=2)}\n"
2755 + summary += "Key test output:\n```\n"
2756 + # Extract relevant parts of test output (e.g., error messages)
2757 + key_output = "\n".join(line for line in attempt.test_output.split("\n")
2758 + if "FAIL" in line or "ERROR" in line or "PASS" in line)
2759 + summary += f"{key_output}\n```\n\n"
2760
2761 + return summary
2762
2763 def forward(self):
2764 """
2765 - The forward function for the AgenticSystem.
2766 + The forward function for the AgenticSystem with multiple solution attempts.
2767 """
2768
2769 - task = f"""I have uploaded a code repository in the directory {self.git_tempdir}. Help
2770 ↪ solve the following problem.
2771 + attempts: List[SolutionAttempt] = []
2772 + best_attempt: Optional[SolutionAttempt] = None
2773
2774 + base_task = f"""I have uploaded a code repository in the directory {self.git_tempdir}.
2775 ↪ Help solve the following problem.
2776
2777 <problem_description>
2778 {self.problem_statement}
2779 @@ -149,8 +214,52 @@ class AgenticSystem:
2780
2781 Your task is to make changes to the files in the {self.git_tempdir} directory to address
2782 ↪ the <problem_description>. I have already taken care of the required dependencies.
2783 """

```

```

2754
2755 - instruction = f"{{task}}\n\nPlease analyze the problem description carefully. Then make
2756 ↪ edits to the code files to complete the instruction."
2757 - init_edit = chat_with_agent(instruction, model=self.code_model, msg_history=[], logging=safe_log)
2758 +
2759 + for attempt_num in range(self.max_attempts):
2760 + # Reset to base commit for each attempt
2761 + reset_to_commit(self.git_tempdir, self.base_commit)
2762 +
2763 + # Prepare the instruction based on previous attempts
2764 + instruction = base_task
2765 + if attempts:
2766 + # Add analysis of previous attempts
2767 + instruction += "\n\nPrevious solution attempts have been made. Here are the results:\n\n"
2768 + instruction += self.analyze_test_results(attempts)
2769 + instruction += "\nPlease analyze these results and provide an improved solution that
2770 ↪ addresses the issues found."
2771 + else:
2772 + instruction += "\n\nPlease analyze the problem description carefully. Then make edits to
2773 ↪ the code files to complete the instruction."
2774 +
2775 + # Get the solution attempt
2776 + chat_with_agent(instruction, model=self.code_model, msg_history=[], logging=safe_log)
2777 +
2778 + # Capture the current patch
2779 + current_patch = diff_versus_commit(self.git_tempdir, self.base_commit)
2780 +
2781 + # Run tests and collect results
2782 + test_success, test_output, test_stats = self.run_tests()
2783 +
2784 + # Create and store attempt
2785 + attempt = SolutionAttempt(
2786 + patch=current_patch,
2787 + test_output=test_output,
2788 + test_success=test_success,
2789 + test_stats=test_stats
2790)
2791 + attempts.append(attempt)
2792 +
2793 + # Update best attempt if this one is better
2794 + if test_success and (best_attempt is None or
2795 + attempt.test_stats["passed"] > best_attempt.test_stats["passed"]):
2796 + best_attempt = attempt
2797 +
2798 + # If tests pass perfectly, we can stop
2799 + if test_success and attempt.test_stats["failed"] == 0 and attempt.test_stats["errors"]
2800 ↪ == 0:
2801 + break
2802 +
2803 + # Apply the best solution if found, otherwise use the last attempt
2804 + final_attempt = best_attempt or attempts[-1]
2805 + reset_to_commit(self.git_tempdir, self.base_commit)
2806 + apply_patch(self.git_tempdir, final_attempt.patch)
2807
2808 def main():
2809 parser = argparse.ArgumentParser(description='Process repository with an agentic
2810 ↪ system.')
2811 @@ -162,6 +271,7 @@ def main():
2812 parser.add_argument('--test_description', default=None, required=False,
2813 ↪ help='Description of how to test the repository')
2814 parser.add_argument('--self_improve', default=False, action='store_true',
2815 ↪ help='Whether to self-improve the repository or solving sw')
2816 parser.add_argument('--language', required=False, default="python", choices=['cpp',
2817 ↪ 'java', 'python', 'go', 'rust', 'javascript'], help='Task\'s programming
2818 ↪ language')
2819 + parser.add_argument('--max_attempts', type=int, default=3, help='Maximum number of
2820 ↪ solution attempts')
2821 args = parser.parse_args()
2822
2823 # Process the repository
2824 @@ -173,6 +283,7 @@ def main():
2825 test_description=args.test_description,
2826 self_improve=args.self_improve,
2827 language=args.language,
2828 + max_attempts=args.max_attempts,
2829)
2830
2831 # Run the agentic system to try to solve the problem
2832 @@ -185,4 +296,4 @@ def main():
2833 f.write(model_patch)

```

```

2808
2809 if __name__ == "__main__":
2810 - main()
2811 + main()
2812 \ No newline at end of file
2813
2814 diff --git a/tools/edit.py b/tools/edit.py
2815 index d7184c3..87fa408 100644
2816 --- a/tools/edit.py
2817 +++ b/tools/edit.py
2818 @@ -1,5 +1,9 @@
2819 from pathlib import Path
2820 import subprocess
2821 +from typing import List, Optional, Dict
2822 +
2823 +# Global edit history for undo functionality
2824 +edit_history: Dict[str, List[str]] = {}
2825
2826 def tool_info():
2827 return {
2828 "@@ -10,14 +14,14 @@ def tool_info():
2829 * The `create` command cannot be used if the specified `path` already exists as a file.\n2830 * If a `command` generates a long output, it will be truncated and marked with `<response\n2831 ↪ clipped>`.\n2832 * The `edit` command overwrites the entire file with the provided `file_text`.\n2833 -* No partial/line-range edits or partial viewing are supported."",",
2834 +* Also supports viewing specific line ranges, string replacement, text insertion, and
2835 ↪ undo operations."",
2836 "input_schema": {
2837 "type": "object",
2838 "properties": {
2839 "command": {
2840 "type": "string",
2841 "enum": ["view", "create", "edit"],
2842 "description": "The command to run: `view`, `create`, or `edit`.",
2843 "enum": ["view", "create", "edit", "str_replace", "insert", "undo_edit"],
2844 "description": "The command to run: `view`, `create`, `edit`, `str_replace`, `insert`, or
2845 ↪ `undo_edit`.",
2846 },
2847 "path": {
2848 "description": "Absolute path to file or directory, e.g. `/repo/file.py`"
2849 ↪ or `/repo`.",
2850 },
2851 "file_text": {
2852 "description": "Required parameter of `create` or `edit` command,
2853 ↪ containing the content for the entire file.",
2854 "type": "string"
2855 },
2856 "view_range": {
2857 "description": "Optional parameter for `view` command to display specific line range
2858 ↪ [start, end].",
2859 "type": "array",
2860 "items": {"type": "integer"}
2861 },
2862 "old_str": {
2863 "description": "Required parameter for `str_replace` command, string to replace.",
2864 "type": "string"
2865 },
2866 "new_str": {
2867 "description": "Required parameter for `str_replace` and `insert` commands, new string
2868 ↪ to insert.",
2869 "type": "string"
2870 },
2871 "insert_line": {
2872 "description": "Required parameter for `insert` command, line number where to insert
2873 ↪ text.",
2874 "type": "integer"
2875 },
2876 "required": ["command", "path"]
2877 },
2878 "validate_path": str, command: str) -> Path:
2879 Validate the file path for each command:
2880 - `view`: path may be a file or directory; must exist.
2881 - `create`: path must not exist (for new file creation).
2882 - `edit`: path must exist (for overwriting).
2883 + - Others: path must exist as a file.
2884 """
2885 path_obj = Path(path)
2886
2887 def validate_path(path: str, command: str) -> Path:
2888 # Path must not exist

```

```

2862 if path_obj.exists():
2863 raise ValueError(f"Cannot create new file; {path} already exists.")
2864 - elif command == "edit":
2865 + else:
2866 # Path must exist and must be a file
2867 if not path_obj.exists():
2868 raise ValueError(f"The file {path} does not exist.")
2869 if path_obj.is_dir():
2870 raise ValueError(f"{path} is a directory and cannot be edited as a file.")
2871 - else:
2872 - raise ValueError(f"Unknown or unsupported command: {command}")
2873
2874 return path_obj
2875
2876 @@ -89,14 +108,21 @@ def read_file(path: Path) -> str:
2877 except Exception as e:
2878 raise ValueError(f"Failed to read file: {e}")
2879
2880 -def write_file(path: Path, content: str):
2881 +def write_file(path: Path, content: str, save_history: bool = True):
2882 """Write (overwrite) entire file contents."""
2883 try:
2884 + if save_history:
2885 + # Save the current content to history before writing
2886 + if path.exists():
2887 + if str(path) not in edit_history:
2888 + edit_history[str(path)] = []
2889 + edit_history[str(path)].append(path.read_text())
2890 +
2891 path.write_text(content)
2892 except Exception as e:
2893 raise ValueError(f"Failed to write file: {e}")
2894
2895 -def view_path(path_obj: Path) -> str:
2896 +def view_path(path_obj: Path, view_range: Optional[List[int]] = None) -> str:
2897 """View the entire file contents or directory listing."""
2898 if path_obj.is_dir():
2899 # For directories: list non-hidden files up to 2 levels deep
2900 @@ -115,35 +141,104 @@ def view_path(path_obj: Path) -> str:
2901 except Exception as e:
2902 raise ValueError(f"Failed to list directory: {e}")
2903
2904 - # If it's a file, show the entire file with line numbers
2905 + # If it's a file
2906 content = read_file(path_obj)
2907 + lines = content.splitlines()
2908 +
2909 + # Handle line range viewing
2910 + if view_range:
2911 + if len(view_range) != 2:
2912 + raise ValueError("view_range must contain exactly two elements: [start, end]")
2913 + start, end = view_range
2914 + if start < 1 or end > len(lines):
2915 + raise ValueError(f"Invalid line range [{start}, {end}] for file with {len(lines)} lines")
2916 + # Adjust for 0-based indexing
2917 + content = "\n".join(lines[start-1:end])
2918 + return format_output(content, str(path_obj), init_line=start)
2919 +
2920 return format_output(content, str(path_obj))
2921
2922 -def tool_function(command: str, path: str, file_text: str = None) -> str:
2923 +def str_replace(path_obj: Path, old_str: str, new_str: str) -> str:
2924 """Replace string in file, ensuring uniqueness."""
2925 content = read_file(path_obj)
2926 # Check for multiple occurrences
2927 if content.count(old_str) > 1:
2928 return f"Error: Multiple occurrences of '{old_str}' found. Replacement requires a unique
2929 ↪ match."
2930 elif content.count(old_str) == 0:
2931 return f"Error: String '{old_str}' not found in file."
2932
2933 new_content = content.replace(old_str, new_str)
2934 write_file(path_obj, new_content)
2935 return f"File at {path_obj} has been edited: replaced '{old_str}' with '{new_str}'."
2936
2937 +def insert_text(path_obj: Path, insert_line: int, new_str: str) -> str:
2938 """Insert text at specified line number."""
2939 content = read_file(path_obj)
2940 lines = content.splitlines()
2941 +
2942 if insert_line < 1 or insert_line > len(lines) + 1:

```

```

2916
2917 + raise ValueError(f"Invalid insert line {insert_line} for file with {len(lines)} lines")
2918
2919 + # Insert the new text at the specified line (adjusting for 0-based index)
2920 + lines.insert(insert_line - 1, new_str.rstrip("\n"))
2921 + new_content = "\n".join(lines) + "\n"
2922
2923 + write_file(path_obj, new_content)
2924 + return f"File at {path_obj} has been edited: inserted text at line {insert_line}."
2925
2926 +def undo_edit(path_obj: Path) -> str:
2927 + """Undo last edit operation on the file."""
2928 + path_str = str(path_obj)
2929 + if path_str not in edit_history or not edit_history[path_str]:
2930 + return "Error: No edit history available for this file."
2931
2932 + # Restore the last saved content
2933 + previous_content = edit_history[path_str].pop()
2934 + write_file(path_obj, previous_content, save_history=False)
2935 + return f"Last edit on {path_obj} has been undone successfully."
2936
2937 +def tool_function(command: str, path: str, **kwargs) -> str:
2938 """
2939 - Main tool function that handles:
2940 - - 'view' : View the entire file or directory listing
2941 - - 'create': Create a new file with the given file_text
2942 - - 'edit' : Overwrite an existing file with file_text
2943 + Main tool function that handles all commands:
2944 - - 'view' : View file/directory (optionally with line range)
2945 - - 'create' : Create new file
2946 - - 'edit' : Overwrite existing file
2947 - - 'str_replace' : Replace string in file
2948 - - 'insert' : Insert text at line
2949 - - 'undo_edit' : Undo last edit
2950 """
2951 try:
2952 path_obj = validate_path(path, command)
2953
2954 if command == "view":
2955 - return view_path(path_obj)
2956 + view_range = kwargs.get('view_range')
2957 + return view_path(path_obj, view_range)
2958
2959 elif command == "create":
2960 - if file_text is None:
2961 + if 'file_text' not in kwargs:
2962 raise ValueError("Missing required `file_text` for 'create' command.")
2963 - write_file(path_obj, file_text)
2964 + write_file(path_obj, kwargs['file_text'])
2965 return f"File created successfully at: {path}"
2966
2967 elif command == "edit":
2968 - if file_text is None:
2969 + if 'file_text' not in kwargs:
2970 raise ValueError("Missing required `file_text` for 'edit' command.")
2971 - write_file(path_obj, file_text)
2972 + write_file(path_obj, kwargs['file_text'])
2973 return f"File at {path} has been overwritten with new content."
2974
2975 elif command == "str_replace":
2976 - if 'old_str' not in kwargs or 'new_str' not in kwargs:
2977 + raise ValueError("Both 'old_str' and 'new_str' are required for string replacement.")
2978 + return str_replace(path_obj, kwargs['old_str'], kwargs['new_str'])
2979
2980 elif command == "insert":
2981 - if 'insert_line' not in kwargs or 'new_str' not in kwargs:
2982 + raise ValueError("Both 'insert_line' and 'new_str' are required for text insertion.")
2983 + return insert_text(path_obj, kwargs['insert_line'], kwargs['new_str'])
2984
2985 elif command == "undo_edit":
2986 + return undo_edit(path_obj)
2987
2988 else:
2989 raise ValueError(f"Unknown command: {command}")
2990
2991
2992 diff --git a/tools/edit.py b/tools/edit.py
2993 index 87fa408..a12903c 100644
2994 --- a/tools/edit.py
2995 +++ b/tools/edit.py
2996 @@ -117,7 +117,7 @@ def write_file(path: Path, content: str, save_history: bool = True):
2997 if str(path) not in edit_history:

```

```

2970
2971 edit_history[str(path)] = []
2972 edit_history[str(path)].append(path.read_text())
2973
2974 path.write_text(content)
2975 except Exception as e:
2976 raise ValueError(f"Failed to write file: {e}")
2977 @@ -172,17 +172,25 @@ def str_replace(path_obj: Path, old_str: str, new_str: str) -> str:
2978 return f"File at {path_obj} has been edited: replaced '{old_str}' with '{new_str}'."
2979
2980 def insert_text(path_obj: Path, insert_line: int, new_str: str) -> str:
2981 """Insert text at specified line number."""
2982 """Insert text at specified line number (1-based)."""
2983 content = read_file(path_obj)
2984 lines = content.splitlines()
2985
2986 if insert_line < 1 or insert_line > len(lines) + 1:
2987 raise ValueError(f"Invalid insert line {insert_line} for file with {len(lines)} lines")
2988 # Validate the insertion line number
2989 if insert_line < 1:
2990 raise ValueError(f"Invalid insert line {insert_line} - must be greater than 0")
2991 if insert_line > len(lines) + 1:
2992 raise ValueError(f"Invalid insert line {insert_line} - file only has {len(lines)} lines")
2993
2994 # Insert the new text at the specified line (adjusting for 0-based index)
2995 lines.insert(insert_line - 1, new_str.rstrip("\n"))
2996 new_content = "\n".join(lines) + "\n"
2997 # Clean up the new text to remove any trailing newlines
2998 new_text = new_str.rstrip('\n')
2999
3000 # Insert at the correct position (line numbers are 1-based, list indices are 0-based)
3001 # Insert at index=insert_line, so it appears after the current line at that position
3002 lines.insert(insert_line, new_text)
3003
3004 # Join lines with newline and add trailing newline
3005 new_content = '\n'.join(lines) + '\n'
3006 write_file(path_obj, new_content)
3007 return f"File at {path_obj} has been edited: inserted text at line {insert_line}."
3008
3009 @@ -243,8 +251,4 @@ def tool_function(command: str, path: str, **kwargs) -> str:
3010 raise ValueError(f"Unknown command: {command}")
3011
3012 except Exception as e:
3013 return f"Error: {str(e)}"
3014
3015 if __name__ == "__main__":
3016 # Example usage
3017 print(tool_function("view", "/home/ubuntu/xx/dgm/coding_agent.py"))
3018 \ No newline at end of file
3019
3020 return f"Error: {str(e)}"
3021 \ No newline at end of file
3022
3023
3024 diff --git a/coding_agent.py b/coding_agent.py
3025 index 6639abd..97f4b69 100644
3026 --- a/coding_agent.py
3027 +++ b/coding_agent.py
3028 @@ -52,6 +52,10 @@ class SolutionAttempt:
3029 test_output: str # Raw test output
3030 test_success: bool # Whether tests passed
3031 test_stats: dict # Test statistics (e.g., number of passed/failed tests)
3032 + error_messages: List[str] = None # List of specific error messages
3033 + test_details: dict = None # Detailed test information like specific test names and their
3034 ↪ status
3035 + execution_time: float = None # Test execution time in seconds
3036 + attempt_number: int = None # The attempt number in the sequence
3037
3038 def get_thread_logger():
3039 """
3040 @@ -150,12 +154,82 @@ class AgenticSystem:
3041]
3042 return new_msg_history
3043
3044 + def extract_test_details(self, output: str) -> tuple[dict, List[str], dict]:
3045 + """Extract detailed test information from the output."""
3046 + error_messages = []
3047 + test_details = {}
3048 + stats = {"passed": 0, "failed": 0, "errors": 0, "total": 0, "skipped": 0}
3049
3050 + # Split output into lines for analysis
3051 + lines = output.split("\n")

```

```

3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
+
+ # Language-specific parsing
+ if self.language == "python":
+ for line in lines:
+ if "FAILED" in line and "::" in line:
+ test_name = line.split("::") [1].split() [0]
+ test_details[test_name] = "FAILED"
+ stats["failed"] += 1
+ elif "PASSED" in line and "::" in line:
+ test_name = line.split("::") [1].split() [0]
+ test_details[test_name] = "PASSED"
+ stats["passed"] += 1
+ elif "ERROR" in line and "::" in line:
+ test_name = line.split("::") [1].split() [0]
+ test_details[test_name] = "ERROR"
+ stats["errors"] += 1
+ # Extract error message
+ if lines.index(line) + 1 < len(lines):
+ error_messages.append(lines[lines.index(line) + 1])
+
+ elif self.language in ["javascript", "node"]:
+ current_test = None
+ for line in lines:
+ if line.startswith('checkmark'):
+ test_name = line.replace('checkmark', '').strip()
+ test_details[test_name] = "PASSED"
+ stats["passed"] += 1
+ elif line.startswith('x'):
+ test_name = line.replace('x', '').strip()
+ test_details[test_name] = "FAILED"
+ stats["failed"] += 1
+ current_test = test_name
+ elif current_test and ('Error:' in line or 'AssertionError:' in line):
+ error_messages.append(f"{current_test}: {line.strip()}")
+
+ elif self.language == "rust":
+ for line in lines:
+ if "test" in line and "... ok" in line:
+ test_name = line.split("test") [1].split("...") [0].strip()
+ test_details[test_name] = "PASSED"
+ stats["passed"] += 1
+ elif "test" in line and "... FAILED" in line:
+ test_name = line.split("test") [1].split("...") [0].strip()
+ test_details[test_name] = "FAILED"
+ stats["failed"] += 1
+ elif "----" in line and " stdout ----" in line:
+ test_name = line.split("----") [1].split("stdout") [0].strip()
+ if test_name in test_details and test_details[test_name] == "FAILED":
+ error_messages.append(f"(test_name): (next((l for l in lines[lines.index(line)+1:] if
+ l.strip()), ''))")
+
+ # Generic counting for other languages or as fallback
+ if not any(stats.values()):
+ stats["passed"] = output.count("PASS") + output.count("ok")
+ stats["failed"] = output.count("FAIL") + output.count("not ok")
+ stats["errors"] = output.count("ERROR") + output.count("panic:")
+
+ stats["total"] = stats["passed"] + stats["failed"] + stats["errors"]
+
+ return stats, error_messages, test_details
+
3065 def run_tests(self) -> tuple[bool, str, dict]:
3066 """Run tests and return success status, output, and test statistics."""
3067 +
3068 success = False
3069 output = ""
3070 stats = {"passed": 0, "failed": 0, "errors": 0, "total": 0}
3071 +
3072 start_time = time.time()
3073 +
3074 try:
3075 for command in TEST_COMMANDS.get(self.language, []):
3076 proc = subprocess.run(
3077 command,
3078 stdout=subprocess.PIPE,
3079 stderr=subprocess.PIPE)
3080 output += proc.stdout.decode("utf-8")
3081 if proc.returncode != 0:
3082 success = False
3083 stats["failed"] += 1
3084 else:
3085 success = True
3086 stats["passed"] += 1
3087
3088 +
3089 if not success:
3090 stats["failed"] += 1
3091
3092 +
3093 stats["total"] = stats["passed"] + stats["failed"] + stats["errors"]
3094
3095 +
3096 return success, output, stats
3097
3098
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3389
3390
3391
3392
3393
3394
3395
3396
3397
3398
3399
3400
3401
3402
3403
3404
3405
3406
3407
3408
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419
3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3469
3470
3471
3472
3473
3474
3475
3476
3477
3478
3479
3480
3481
3482
3483
3484
3485
3486
3487
3488
3489
3490
3491
3492
3493
3494
3495
3496
3497
3498
3499
3500
3501
3502
3503
3504
3505
3506
3507
3508
3509
3510
3511
3512
3513
3514
3515
3516
3517
3518
3519
3520
3521
3522
3523
3524
3525
3526
3527
3528
3529
3530
3531
3532
3533
3534
3535
3536
3537
3538
3539
3540
3541
3542
3543
3544
3545
3546
3547
3548
3549
3550
3551
3552
3553
3554
3555
3556
3557
3558
3559
3560
3561
3562
3563
3564
3565
3566
3567
3568
3569
3570
3571
3572
3573
3574
3575
3576
3577
3578
3579
3580
3581
3582
3583
3584
3585
3586
3587
3588
3589
3590
3591
3592
3593
3594
3595
3596
3597
3598
3599
3600
3601
3602
3603
3604
3605
3606
3607
3608
3609
3610
3611
3612
3613
3614
3615
3616
3617
3618
3619
3620
3621
3622
3623
3624
3625
3626
3627
3628
3629
3630
3631
3632
3633
3634
3635
3636
3637
3638
3639
3640
3641
3642
3643
3644
3645
3646
3647
3648
3649
3650
3651
3652
3653
3654
3655
3656
3657
3658
3659
3660
3661
3662
3663
3664
3665
3666
3667
3668
3669
3670
3671
3672
3673
3674
3675
3676
3677
3678
3679
3680
3681
3682
3683
3684
3685
3686
3687
3688
3689
3690
3691
3692
3693
3694
3695
3696
3697
3698
3699
3700
3701
3702
3703
3704
3705
3706
3707
3708
3709
3710
3711
3712
3713
3714
3715
3716
3717
3718
3719
3720
3721
3722
3723
3724
3725
3726
3727
3728
3729
3730
3731
3732
3733
3734
3735
3736
3737
3738
3739
3740
3741
3742
3743
3744
3745
3746
3747
3748
3749
3750
3751
3752
3753
3754
3755
3756
3757
3758
3759
3760
3761
3762
3763
3764
3765
3766
3767
3768
3769
3770
3771
3772
3773
3774
3775
3776
3777
3778
3779
3780
3781
3782
3783
3784
3785
3786
3787
3788
3789
3790
3791
3792
3793
3794
3795
3796
3797
3798
3799
3800
3801
3802
3803
3804
3805
3806
3807
3808
3809
3810
3811
3812
3813
3814
3815
3816
3817
3818
3819
3820
3821
3822
3823
3824
3825
3826
3827
3828
3829
3830
3831
3832
3833
3834
3835
3836
3837
3838
3839
3840
3841
3842
3843
3844
3845
3846
3847
3848
3849
3850
3851
3852
3853
3854
3855
3856
3857
3858
3859
3860
3861
3862
3863
3864
3865
3866
3867
3868
3869
3870
3871
3872
3873
3874
3875
3876
3877
3878
3879
3880
3881
3882
3883
3884
3885
3886
3887
3888
3889
3890
3891
3892
3893
3894
3895
3896
3897
3898
3899
3900
3901
3902
3903
3904
3905
3906
3907
3908
3909
3910
3911
3912
3913
3914
3915
3916
3917
3918
3919
3920
3921
3922
3923
3924
3925
3926
3927
3928
3929
3930
3931
3932
3933
3934
3935
3936
3937
3938
3939
3940
3941
3942
3943
3944
3945
3946
3947
3948
3949
3950
3951
3952
3953
3954
3955
3956
3957
3958
3959
3960
3961
3962
3963
3964
3965
3966
3967
3968
3969
3970
3971
3972
3973
3974
3975
3976
3977
3978
3979
3980
3981
3982
3983
3984
3985
3986
3987
3988
3989
3990
3991
3992
3993
3994
3995
3996
3997
3998
3999
3999

```

```

3078
3079 - stats["passed"] = output.count("PASS") + output.count("ok")
3080 - stats["failed"] = output.count("FAIL") + output.count("not ok")
3081 - stats["errors"] = output.count("ERROR") + output.count("panic:")
3082 - stats["total"] = stats["passed"] + stats["failed"] + stats["errors"]
3083 +
3084 + # Extract detailed test information
3085 + stats, error_messages, test_details = self.extract_test_details(output)
3086 + stats["execution_time"] = time.time() - start_time
3087 +
3088 + # Enhance stats with extracted information
3089 + stats["error_messages"] = error_messages
3090 + stats["test_details"] = test_details
3091
3092 except Exception as e:
3093 output = f"Error running tests: {str(e)}"
3094 success = False
3095 stats = {
3096 "passed": 0, "failed": 0, "errors": 1, "total": 1,
3097 "execution_time": time.time() - start_time,
3098 "error_messages": [str(e)],
3099 "test_details": {}
3100 }
3101
3102 return success, output, stats
3103
3104 def analyze_test_results(self, attempts: List[SolutionAttempt]) -> str:
3105 """Analyze test results and create a summary for the agent."""
3106 """Analyze test results and create a detailed summary for the agent."""
3107 summary = "# Test Results Analysis\n\n"
3108
3109 # Overall progress tracking
3110 if len(attempts) > 1:
3111 summary += "# Progress Overview\n"
3112 first_attempt = attempts[0].test_stats
3113 last_attempt = attempts[-1].test_stats
3114
3115 progress = {
3116 "passed": last_attempt["passed"] - first_attempt["passed"],
3117 "failed": first_attempt["failed"] - last_attempt["failed"],
3118 "errors": first_attempt["errors"] - last_attempt["errors"]
3119 }
3120
3121 summary += "Progress since first attempt:\n"
3122 summary += f"- Additional passing tests: {progress['passed']}\n"
3123 summary += f"- Reduced failures: {progress['failed']}\n"
3124 summary += f"- Reduced errors: {progress['errors']}\n\n"
3125
3126 # Detailed attempt analysis
3127 for i, attempt in enumerate(attempts, 1):
3128 summary += f"## Attempt {i}\n"
3129 summary += f"Test Success: {attempt.test_success}\n"
3130
3131 summary += f"Test Stats: {json.dumps(attempt.test_stats, indent=2)}\n"
3132 summary += "Key test output:\n```\n"
3133
3134 # Extract relevant parts of test output (e.g., error messages)
3135 key_output = "\n".join(line for line in attempt.test_output.split("\n")
3136 if "FAIL" in line or "ERROR" in line or "PASS" in line)
3137 summary += f"{key_output}\n```\n\n"
3138
3139 summary += f"Execution Time: {attempt.test_stats.get('execution_time', 'N/A'):.2f}s\n"
3140
3141 # Test statistics
3142 stats = attempt.test_stats
3143 total = stats.get("total", 0) or 1 # Avoid division by zero
3144 pass_rate = (stats.get("passed", 0) / total) * 100
3145
3146 summary += f"Pass Rate: {pass_rate:.1f}% ({stats.get('passed', 0)}/{total})\n"
3147 summary += "Test Statistics:\n"
3148 summary += f"- Passed: {stats.get('passed', 0)}\n"
3149 summary += f"- Failed: {stats.get('failed', 0)}\n"
3150 summary += f"- Errors: {stats.get('errors', 0)}\n"
3151 summary += f"- Total: {total}\n\n"
3152
3153 # Error messages
3154 if stats.get("error_messages"):
3155 summary += "Error Messages:\n```\n"
3156 for error in stats["error_messages"][:5]: # Limit to top 5 errors
3157 summary += f"({error})\n"
3158 if len(stats["error_messages"]) > 5:
3159 summary += f"... and {len(stats['error_messages']) - 5} more errors\n"
3160 summary += "```\n\n"
3161
3162

```

```

3132 + # Test details
3133 + if stats.get("test_details"):
3134 + summary += "Individual Test Results:\n```\n"
3135 + for test_name, result in stats["test_details"].items():
3136 + summary += f"{result}: {test_name}\n"
3137 + summary += "```\n\n"
3138 + # Recommendations for next attempt
3139 + if not attempts[-1].test_success:
3140 + summary += "## Recommendations for Next Attempt\n"
3141 + last_stats = attempts[-1].test_stats
3142 +
3143 + if last_stats.get("errors", 0) > 0:
3144 + summary += "- Focus on resolving runtime errors first\n"
3145 + if last_stats.get("failed", 0) > 0:
3146 + summary += "- Address failing test cases\n"
3147 + if len(attempts) > 1 and not attempts[-1].test_success:
3148 + # Compare with previous attempt
3149 + prev_stats = attempts[-2].test_stats
3150 + if last_stats.get("passed", 0) < prev_stats.get("passed", 0):
3151 + summary += "- Recent changes caused regressions. Consider reverting some changes\n"
3152 +
3153 + return summary
3154
3155 def forward(self):
3156 @@ -238,20 +375,36 @@ Your task is to make changes to the files in the {self.git_tempdir}
3157 ↪ directory to
3158 # Run tests and collect results
3159 test_success, test_output, test_stats = self.run_tests()
3160
3161 - # Create and store attempt
3162 + # Create and store attempt with enhanced information
3163 attempt = SolutionAttempt(
3164 patch=current_patch,
3165 test_output=test_output,
3166 test_success=test_success,
3167 - test_stats=test_stats
3168 + test_stats=test_stats,
3169 + error_messages=test_stats.get('error_messages', []),
3170 + test_details=test_stats.get('test_details', {}),
3171 + execution_time=test_stats.get('execution_time', None),
3172 + attempt_number=attempt_num + 1
3173)
3174 attempts.append(attempt)
3175
3176 - # Update best attempt if this one is better
3177 - if test_success and (best_attempt is None or
3178 - attempt.test_stats["passed"] > best_attempt.test_stats["passed"]):
3179 + # Update best attempt based on multiple criteria
3180 + if test_success and (
3181 + best_attempt is None or
3182 + (attempt.test_stats["passed"] > best_attempt.test_stats["passed"]) or
3183 + (attempt.test_stats["passed"] == best_attempt.test_stats["passed"] and
3184 + len(attempt.error_messages or []) < len(best_attempt.error_messages or [])))
3185 +):
3186 best_attempt = attempt
3187
3188 + # Log detailed attempt information
3189 + safe_log(f"\n== Attempt {attempt_num + 1} Summary ===")
3190 + safe_log(f"Test Success: {test_success}")
3191 + safe_log(f"Tests Passed: {test_stats.get('passed', 0)}")
3192 + safe_log(f"Tests Failed: {test_stats.get('failed', 0)}")
3193 + safe_log(f"Errors: {test_stats.get('errors', 0)}")
3194 + safe_log(f"Execution Time: {test_stats.get('execution_time', 'N/A'):.2f}s")
3195 +
3196 # If tests pass perfectly, we can stop
3197 if test_success and attempt.test_stats["failed"] == 0 and
3198 ↪ attempt.test_stats["errors"] == 0:
3199 break
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3389
3390
3391
3392
3393
3394
3395
3396
3397
3398
3399
3400
3401
3402
3403
3404
3405
3406
3407
3408
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419
3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3469
3470
3471
3472
3473
3474
3475
3476
3477
3478
3479
3480
3481
3482
3483
3484
3485
3486
3487
3488
3489
3490
3491
3492
3493
3494
3495
3496
3497
3498
3499
3500
3501
3502
3503
3504
3505
3506
3507
3508
3509
3510
3511
3512
3513
3514
3515
3516
3517
3518
3519
3520
3521
3522
3523
3524
3525
3526
3527
3528
3529
3530
3531
3532
3533
3534
3535
3536
3537
3538
3539
3540
3541
3542
3543
3544
3545
3546
3547
3548
3549
3550
3551
3552
3553
3554
3555
3556
3557
3558
3559
3560
3561
3562
3563
3564
3565
3566
3567
3568
3569
3570
3571
3572
3573
3574
3575
3576
3577
3578
3579
3580
3581
3582
3583
3584
3585
3586
3587
3588
3589
3590
3591
3592
3593
3594
3595
3596
3597
3598
3599
3600
3601
3602
3603
3604
3605
3606
3607
3608
3609
3610
3611
3612
3613
3614
3615
3616
3617
3618
3619
3620
3621
3622
3623
3624
3625
3626
3627
3628
3629
3630
3631
3632
3633
3634
3635
3636
3637
3638
3639
3640
3641
3642
3643
3644
3645
3646
3647
3648
3649
3650
3651
3652
3653
3654
3655
3656
3657
3658
3659
3660
3661
3662
3663
3664
3665
3666
3667
3668
3669
3670
3671
3672
3673
3674
3675
3676
3677
3678
3679
3680
3681
3682
3683
3684
3685
3686
3687
3688
3689
3690
3691
3692
3693
3694
3695
3696
3697
3698
3699
3700
3701
3702
3703
3704
3705
3706
3707
3708
3709
3710
3711
3712
3713
3714
3715
3716
3717
3718
3719
3720
3721
3722
3723
3724
3725
3726
3727
3728
3729
3730
3731
3732
3733
3734
3735
3736
3737
3738
3739
3740
3741
3742
3743
3744
3745
3746
3747
3748
3749
3750
3751
3752
3753
3754
3755
3756
3757
3758
3759
3760
3761
3762
3763
3764
3765
3766
3767
3768
3769
3770
3771
3772
3773
3774
3775
3776
3777
3778
3779
3780
3781
3782
3783
3784
3785
3786
3787
3788
3789
3790
3791
3792
3793
3794
3795
3796
3797
3798
3799
3800
3801
3802
3803
3804
3805
3806
3807
3808
3809
3810
3811
3812
3813
3814
3815
3816
3817
3818
3819
3820
3821
3822
3823
3824
3825
3826
3827
3828
3829
3830
3831
3832
3833
3834
3835
3836
3837
3838
3839
3840
3841
3842
3843
3844
3845
3846
3847
3848
3849
3850
3851
3852
3853
3854
3855
3856
3857
3858
3859
3860
3861
3862
3863
3864
3865
3866
3867
3868
3869
3870
3871
3872
3873
3874
3875
3876
3877
3878
3879
3880
3881
3882
3883
3884
3885
3886
3887
3888
3889
3890
3891
3892
3893
3894
3895
3896
3897
3898
3899
3900
3901
3902
3903
3904
3905
3906
3907
3908
3909
3910
3911
3912
3913
3914
3915
3916
3917
3918
3919
3920
3921
3922
3923
3924
3925
3926
3927
3928
3929
3930
3931
3932
3933
3934
3935
3936
3937
3938
3939
3940
3941
3942
3943
3944
3945
3946
3947
3948
3949
3950
3951
3952
3953
3954
3955
3956
3957
3958
3959
3960
3961
3962
3963
3964
3965
3966
3967
3968
3969
3970
3971
3972
3973
3974
3975
3976
3977
3978
3979
3980
3981
3982
3983
3984
3985
3986
3987
3988
3989
3990
3991
3992
3993
3994
3995
3996
3997
3998
3999
3999

```

## G SIMILAR TARGET FUNCTIONALITY, DIFFERENT IMPLEMENTATIONS

For the same target functionality, the suggestions provided to the coding agents and the resulting implementations can differ significantly, leading to large variations in coding capability and benchmark performance. For example, consider nodes 6 and 24 in the DGM run on SWE-bench (Figure 3). Both aimed to enhance the existing editor tool to support finer-grained file editing. However, the implemen-

3186 tation for node 6 retained the original `edit` command and added the parameters `edit_type` and  
 3187 `edit_actions`, whereas the implementation for node 24 replaced the original `edit` command  
 3188 with a new `str_replace` command. Despite targeting similar functionality, the feature suggestions  
 3189 and thus the implementation details differed greatly, as did their performance. Node 6 achieved only  
 3190 23.3%, while node 24 achieved 40.5%.

3191 **Feature suggestion to obtain node 6:**

3193 Within the `tools/edit.py` file, remove the '`edit`' command and introduce a '`str_replace`'  
 3194 → command. It should accept parameters '`old_str`' (the unique text to find) and  
 3195 → '`new_str`' (the new text to replace it with). After verifying that '`old_str`'  
 3196 → appears exactly once in the file, replace the first occurrence with '`new_str`'. If  
 3197 → '`old_str`' is not unique or is missing, do not write any changes.  
 3198 Replace the '`edit`' command in the `editor` tool with a '`str_replace`' command that locates  
 3199 → and replaces an exact substring in a file. The '`old_str`' must match consecutive  
 3200 → lines uniquely, and if it is not found or occurs multiple times, no write should  
 3201 → occur. The '`new_str`' parameter contains the revised lines. This precise substring  
 3202 → replacement flow will enable small incremental edits with minimal breakage.

3201 **Feature suggestion to obtain node 24:**

3203 Augment the existing '`editor`' tool to support line or token-based editing. The agent would  
 3204 → specify individual lines or tokens to insert, delete, or replace. Internally, the  
 3205 → tool would parse the file contents, apply changes atomically, and verify that the  
 3206 → surrounding context remains intact. This prevents entire-file overwrites, reducing  
 3207 → the risk of losing unrelated logic. The tool could also validate that the edited  
 3208 → lines match expected code fragments to catch unanticipated differences or merge  
 3209 → conflicts.  
 3210 Enhance the agent's file editing process to support fine-grained edits. Specifically, add  
 3211 → a line-based or token-based editing feature in the '`editor`' tool. This improvement  
 3212 → should let the agent insert, replace, or delete small sections of code without  
 3213 → overwriting entire files or large blocks, preserving unrelated logic. Implement  
 3214 → safeguards to confirm that contextual code around the edited region remains  
 3215 → accurate, so we reduce the risk of creating regressions when fixing issues.

3213 **Implementation of suggested feature to obtain node 6:**

```
3214
3215 diff --git a/tests/test_edit_tool.py b/tests/test_edit_tool.py
3216 index f0782c6..d3a4d5c 100644
3217 --- a/tests/test_edit_tool.py
3218 +++ b/tests/test_edit_tool.py
3219 @@ -1,7 +1,7 @@
3220 import pytest
3221 from pathlib import Path
3222 import tempfile
3223 -from tools.edit import tool_function
3224 +from tools.edit import tool_function, validate_edit_actions, validate_context
3225
3226 @pytest.fixture
3227 def temp_dir():
3228 @@ -17,6 +17,18 @@
3229 def sample_file(temp_dir):
3230 file_path.write_text(content)
3231 return file_path
3232
3233 +@pytest.fixture
3234 +def code_file(temp_dir):
3235 + """Create a sample Python file for testing token-based edits."""
3236 + file_path = temp_dir / "test.py"
3237 + content = """def test_function():
3238 + x = 42
3239 + y = "old_value"
3240 + return x + len(y)
3241 """
3242 + file_path.write_text(content)
3243 + return file_path
3244
3245 class TestEditorTool:
3246 def test_view_file(self, sample_file):
3247 """Test viewing entire file content."""
3248 @@ -39,17 +51,182 @@
3249 class TestEditorTool:
3250 assert "Error" in result
3251 assert "already exists" in result
3252
3253 - def test_edit_file(self, sample_file):
3254 - """Test editing an existing file."""
3255 + def test_edit_file_full(self, sample_file):
```

```

3240
3241 + """Test editing an existing file with file mode."""
3242 + new_content = "edited content\nnew line"
3243 - result = tool_function("edit", str(sample_file), file_text=new_content)
3244 + result = tool_function(
3245 + "edit",
3246 + str(sample_file),
3247 + file_text=new_content,
3248 + edit_type="file"
3249 +)
3250 assert "has been overwritten" in result
3251 assert sample_file.read_text() == new_content
3252
3253 + def test_line_based_edit_insert(self, sample_file):
3254 + """Test line-based insert operation."""
3255 + actions = [{}
3256 + "action": "insert",
3257 + "line_number": 3,
3258 + "content": "inserted line"
3259 + }]
3260 + result = tool_function(
3261 + "edit",
3262 + str(sample_file),
3263 + edit_type="line",
3264 + edit_actions=actions
3265 +)
3266 assert "successfully edited" in result
3267 content = sample_file.read_text().splitlines()
3268 assert content[2] == "inserted line"
3269 assert len(content) == 6
3270
3271 + def test_line_based_edit_delete(self, sample_file):
3272 + """Test line-based delete operation."""
3273 + actions = [{}
3274 + "action": "delete",
3275 + "line_number": 2
3276 + }]
3277 + result = tool_function(
3278 + "edit",
3279 + str(sample_file),
3280 + edit_type="line",
3281 + edit_actions=actions
3282 +)
3283 assert "successfully edited" in result
3284 content = sample_file.read_text().splitlines()
3285 assert content[2] not in content
3286 assert len(content) == 4
3287
3288 + def test_line_based_edit_replace(self, sample_file):
3289 + """Test line-based replace operation."""
3290 + actions = [{}
3291 + "action": "replace",
3292 + "line_number": 3,
3293 + "content": "replaced line"
3294 + }]
3295 + result = tool_function(
3296 + "edit",
3297 + str(sample_file),
3298 + edit_type="line",
3299 + edit_actions=actions
3300 +)
3301 assert "successfully edited" in result
3302 content = sample_file.read_text().splitlines()
3303 assert content[2] == "replaced line"
3304 assert len(content) == 5
3305
3306 + def test_token_based_edit(self, code_file):
3307 + """Test token-based edit operation."""
3308 + actions = [{}
3309 + "action": "replace",
3310 + "old_token": "old_value",
3311 + "new_token": "new_value"
3312 + }]
3313 + result = tool_function(
3314 + "edit",
3315 + str(code_file),
3316 + edit_type="token",
3317 + edit_actions=actions
3318 +)
3319 assert "successfully edited" in result
3320 content = code_file.read_text()
3321 assert "old_value" not in content

```

```

3294 + assert "new_value" in content
3295 +
3296 + def test_multiple_token_edits(self, code_file):
3297 + """Test multiple token-based edit operations."""
3298 + actions = [
3299 + {
3300 + "action": "replace",
3301 + "old_token": "42",
3302 + "new_token": "100"
3303 + },
3304 + {
3305 + "action": "replace",
3306 + "old_token": "old_value",
3307 + "new_token": "new_text"
3308 + }
3309 +]
3310 + result = tool_function(
3311 + "edit",
3312 + str(code_file),
3313 + edit_type="token",
3314 + edit_actions=actions
3315 +)
3316 + assert "successfully edited" in result
3317 + content = code_file.read_text()
3318 + assert "42" not in content
3319 + assert "100" in content
3320 + assert "old_value" not in content
3321 + assert "new_text" in content
3322 +
3323 + def test_invalid_line_number(self, sample_file):
3324 + """Test edit with invalid line number."""
3325 + actions = [
3326 + {
3327 + "action": "replace",
3328 + "line_number": 100,
3329 + "content": "invalid line"
3330 + }]
3331 + result = tool_function(
3332 + "edit",
3333 + str(sample_file),
3334 + edit_type="line",
3335 + edit_actions=actions
3336 +)
3337 + assert "Error" in result
3338 + assert "exceeds file length" in result
3339 +
3340 + def test_invalid_edit_type(self, sample_file):
3341 + """Test edit with invalid edit type."""
3342 + result = tool_function(
3343 + "edit",
3344 + str(sample_file),
3345 + edit_type="invalid",
3346 + edit_actions=[{"action": "replace"}]
3347 +)
3348 + assert "Error" in result
3349 + assert "Invalid edit_type" in result
3350 +
3351 + def test_missing_required_fields(self, sample_file):
3352 + """Test edit with missing required fields."""
3353 + actions = [
3354 + {
3355 + "action": "replace" # Missing line_number and content
3356 + }]
3357 + result = tool_function(
3358 + "edit",
3359 + str(sample_file),
3360 + edit_type="line",
3361 + edit_actions=actions
3362 +)
3363 + assert "Error" in result
3364 + assert "Missing required field" in result
3365 +
3366 + def test_validate_context(self):
3367 + """Test context validation function."""
3368 + original = "line 1\nline 2\nline 3\nline 4\nline 5"
3369 + modified = "line 1\nline 2\nmodified\nline 4\nline 5"
3370 + assert validate_context(original, modified) is True
3371 +
3372 + # Test invalid context (too many changes)
3373 + modified = "completely\ndifferent\ncontent"
3374 + assert validate_context(original, modified) is False
3375 +
3376 + def test_empty_edit_actions(self, sample_file):

```

```

3348
3349 + """Test edit with empty edit actions."""
3350 + result = tool_function(
3351 + "edit",
3352 + str(sample_file),
3353 + edit_type="line",
3354 + edit_actions=[]
3355 +)
3356 + assert "Error" in result
3357 + assert "Edit actions cannot be empty" in result
3358 +
3359 def test_edit_nonexistent_file(self, temp_dir):
3360 """Test attempting to edit a nonexistent file."""
3361 non_existent_file = temp_dir / "does_not_exist.txt"
3362 - result = tool_function("edit", str(non_existent_file), file_text="new content")
3363 + result = tool_function("edit", str(non_existent_file), edit_type="line", edit_actions[])
3364 assert "Error" in result
3365 assert "does not exist" in result
3366
3367 @@ -82,4 +259,4 @@ class TestEditorTool:
3368 def test_invalid_commands(self, command, sample_file):
3369 """Test various invalid commands."""
3370 result = tool_function(command, str(sample_file))
3371 - assert "Error" in result
3372 + assert "Error" in result
3373 \ No newline at end of file
3374 diff --git a/tools/edit.py b/tools/edit.py
3375 index 59137ee..982ee84 100644
3376 --- a/tools/edit.py
3377 +++ b/tools/edit.py
3378 @@ -1,5 +1,7 @@
3379 from pathlib import Path
3380 import subprocess
3381 +from typing import Dict, Any, List, Union, Tuple
3382 +from difflib import SequenceMatcher
3383
3384 def tool_info():
3385 return {
3386 @@ -9,8 +11,9 @@ def tool_info():
3387 * If `path` is a file, `view` displays the entire file with line numbers. If `path` is a
3388 → directory, `view` lists non-hidden files and directories up to 2 levels deep.\n
3389 * The `create` command cannot be used if the specified `path` already exists as a file.\n
3390 * If a `command` generates a long output, it will be truncated and marked with `<response
3391 → clipped>`.\n
3392 -* The `edit` command overwrites the entire file with the provided `file_text`.\n
3393 -* No partial/line-range edits or partial viewing are supported."",",
3394 +* The `edit` command supports both entire file overwrites and fine-grained line/token
3395 → editing via the `edit_type` parameter.\n
3396 +* Line-based edits require line numbers and content to modify specific parts of a file.\n
3397 +* Token-based edits require specifying old and new tokens to replace specific text
3398 → fragments."",",
3399 "input_schema": {
3400 "type": "object",
3401 "properties": {
3402 @@ -24,8 +27,28 @@ def tool_info():
3403 "type": "string"
3404 },
3405 "file_text": {
3406 - "description": "Required parameter of `create` or `edit` command, containing the content
3407 → for the entire file.",
3408 + "description": "Required parameter of `create` or `edit` command with edit_type='file',
3409 → containing the content for the entire file.",
3410 "type": "string"
3411 },
3412 "edit_type": {
3413 + "type": "string",
3414 + "enum": ["file", "line", "token"],
3415 + "description": "Type of edit operation: 'file' for full file, 'line' for line-based
3416 → edits, 'token' for token-based edits.",
3417 + "default": "file"
3418 },
3419 "edit_actions": {
3420 + "type": "array",
3421 + "description": "List of edit actions for line/token operations. Each action contains
3422 → operation details.",
3423 + "items": {
3424 + "type": "object",
3425 + "properties": {
3426 + "action": {"type": "string", "enum": ["insert", "delete", "replace"]},
3427 + "line_number": {"type": "integer", "description": "Line number for the operation
3428 → (1-based)" },
3429 + "content": {"type": "string", "description": "Content to insert/replace"},
```

```

3402
3403 + "old_token": {"type": "string", "description": "Token to be replaced (for token edits)"},

3404 + "new_token": {"type": "string", "description": "New token (for token edits)"}
3405 + }
3406 + }
3407 + "required": ["command", "path"]
3408 @@ -119,12 +142,126 @@ def view_path(path_obj: Path) -> str:

3409 content = read_file(path_obj)
3410 return format_output(content, str(path_obj))
3411
3412 -def tool_function(command: str, path: str, file_text: str = None) -> str:
3413 +def validate_edit_actions(actions: List[Dict[str, Any]], edit_type: str) -> None:
3414 + """Validate edit actions based on edit type."""
3415 + if not actions:
3416 + raise ValueError("Edit actions cannot be empty for line/token edits")
3417 +
3418 + valid_actions = ["insert", "delete", "replace"]
3419 + required_fields = {
3420 + "line": ["action", "line_number"],
3421 + "token": ["action", "old_token"]
3422 + }
3423 +
3424 + for action in actions:
3425 + if "action" not in action or action["action"] not in valid_actions:
3426 + raise ValueError(f"Invalid action. Must be one of: {valid_actions}")
3427 +
3428 + # Check required fields based on edit_type
3429 + for field in required_fields[edit_type]:
3430 + if field not in action:
3431 + raise ValueError(f"Missing required field '{field}' in edit action")
3432 +
3433 + # Validate line number if provided
3434 + if "line_number" in action:
3435 + if not isinstance(action["line_number"], int) or action["line_number"] < 1:
3436 + raise ValueError("Line number must be a positive integer")
3437 +
3438 + # Validate content requirements
3439 + if action["action"] in ["insert", "replace"]:
3440 + if edit_type == "line" and "content" not in action:
3441 + raise ValueError("Content required for insert/replace actions")
3442 + if edit_type == "token" and "new_token" not in action:
3443 + raise ValueError("new_token required for token operations")
3444 +
3445 +def apply_line_edits(content: List[str], actions: List[Dict[str, Any]]) -> List[str]:
3446 + """Apply line-based edits to the content."""
3447 + modified_content = content.copy()
3448 +
3449 + # Sort actions by line number in reverse order to handle inserts/deletes correctly
3450 + sorted_actions = sorted(actions, key=lambda x: x["line_number"], reverse=True)
3451 +
3452 + for action in sorted_actions:
3453 + line_num = action["line_number"] - 1 # Convert to 0-based index
3454 +
3455 + if line_num > len(modified_content):
3456 + raise ValueError(f"Line number {action['line_number']} exceeds file length")
3457 +
3458 + if action["action"] == "delete":
3459 + if line_num >= 0:
3460 + del modified_content[line_num]
3461 + elif action["action"] == "insert":
3462 + modified_content.insert(line_num, action["content"])
3463 + elif action["action"] == "replace":
3464 + modified_content[line_num] = action["content"]
3465 +
3466 + return modified_content
3467
3468 +def apply_token_edits(content: str, actions: List[Dict[str, Any]]) -> str:
3469 + """Apply token-based edits to the content."""
3470 + modified_content = content
3471 +
3472 + for action in actions:
3473 + old_token = action["old_token"]
3474 + if action["action"] == "delete":
3475 + modified_content = modified_content.replace(old_token, "")
3476 + elif action["action"] in ["insert", "replace"]:
3477 + new_token = action["new_token"]
3478 + modified_content = modified_content.replace(old_token, new_token)
3479 +
3480 + return modified_content
3481

```

```

3456
3457 +def validate_context(original: str, modified: str, context_lines: int = 3) -> bool:
3458 """
3459 - Main tool function that handles:
3460 + Validate that the context around modified sections remains intact.
3461 + Returns True if context is valid, False otherwise.
3462 + """
3463 + # Use SequenceMatcher to find the similarity ratio between strings
3464 + similarity = SequenceMatcher(None, original, modified).ratio()
3465 +
3466 + # If content is too different (less than 30% similar), consider it invalid
3467 + if similarity < 0.3:
3468 + return False
3469 +
3470 + # Split into lines for more detailed analysis
3471 + orig_lines = original.splitlines()
3472 + mod_lines = modified.splitlines()
3473 +
3474 + # If line count difference is too large (more than 50%), consider it invalid
3475 + if abs(len(orig_lines) - len(mod_lines)) > len(orig_lines) // 2:
3476 + return False
3477 +
3478 + # Calculate line-based similarity for sections
3479 + def get_block_similarity(block1: List[str], block2: List[str]) -> float:
3480 + return SequenceMatcher(None, "\n".join(block1), "\n".join(block2)).ratio()
3481 +
3482 + # Check similarity of start and end blocks if they exist
3483 + if len(orig_lines) >= context_lines and len(mod_lines) >= context_lines:
3484 + start_similarity = get_block_similarity(
3485 + orig_lines[:context_lines],
3486 + mod_lines[:context_lines]
3487 +)
3488 + end_similarity = get_block_similarity(
3489 + orig_lines[-context_lines:],
3490 + mod_lines[-context_lines:]
3491 +)
3492 +
3493 + # If either the start or end blocks are too different, consider it invalid
3494 + if start_similarity < 0.7 or end_similarity < 0.7:
3495 + return False
3496 +
3497 + return True
3498 +
3499 +def tool_function(command: str, path: str, file_text: str = None,
3500 edit_type: str = "file", edit_actions: List[Dict[str, Any]] = None) -> str:
3501 """
3502 + Enhanced tool function that handles:
3503 - 'view' : View the entire file or directory listing
3504 - 'create' : Create a new file with the given file_text
3505 - - 'edit' : Overwrite an existing file with file_text
3506 + - 'edit' : Edit a file using one of three modes:
3507 + - file: overwrite entire file
3508 + - line: line-based edits
3509 + - token: token-based edits
3510 """
3511 try:
3512 path_obj = validate_path(path, command)
3513 @@ -139,11 +276,36 @@ def tool_function(command: str, path: str, file_text: str = None) ->
3514 str:
3515 return f"File created successfully at: {path}"
3516
3517 elif command == "edit":
3518 - if file_text is None:
3519 - raise ValueError("Missing required `file_text` for 'edit' command.")
3520 - write_file(path_obj, file_text)
3521 - return f"File at {path} has been overwritten with new content."
3522 + # Read original content for line/token edits and validation
3523 + original_content = read_file(path_obj) if edit_type in ["line", "token"] else None
3524 +
3525 + if edit_type == "file":
3526 + if file_text is None:
3527 + raise ValueError("Missing required `file_text` for 'file' edit type.")
3528 + write_file(path_obj, file_text)
3529 + return f"File at {path} has been overwritten with new content."
3530 +
3531 + elif edit_type in ["line", "token"]:
3532 + if not edit_actions:
3533 + raise ValueError("Edit actions cannot be empty for line/token edits")
3534 +
3535 + validate_edit_actions(edit_actions, edit_type)
3536 +
3537 + if edit_type == "line":
3538

```

```

3510 + lines = original_content.splitlines()
3511 + modified_lines = apply_line_edits(lines, edit_actions)
3512 + modified_content = "\n".join(modified_lines)
3513 + else: # token edits
3514 + modified_content = apply_token_edits(original_content, edit_actions)
3515
3516 + # Validate context before writing
3517 + if validate_context(original_content, modified_content):
3518 + write_file(path_obj, modified_content)
3519 + return f"File at {path} has been successfully edited using {edit_type}-based editing."
3520 + else:
3521 + raise ValueError("Context validation failed. Edit operation aborted.")
3522 + else:
3523 + raise ValueError(f"Invalid edit_type: {edit_type}")
3524 else:
3525 raise ValueError(f"Unknown command: {command}")
3526
3527 @@ -152,4 +314,4 @@ def tool_function(command: str, path: str, file_text: str = None) ->
3528 ↪ str:
3529
3530 if __name__ == "__main__":
3531 # Example usage
3532 - print(tool_function("view", "/home/ubuntu/xx/dgm/coding_agent.py"))
3533 + print(tool_function("view", "/home/ubuntu/xx/dgm/coding_agent.py"))
3534
3535 \ No newline at end of file

```

3528 Implementation of suggested feature to obtain node 24:

```

3530 diff --git a/tools/edit.py b/tools/edit.py
3531 index 16ae521..757f5c2 100644
3532 --- a/tools/edit.py
3533 +++ b/tools/edit.py
3534 @@ -11,21 +11,21 @@ def tool_info():
3535 * If `path` is a directory, `view` lists non-hidden files and directories up to 2 levels
3536 ↪ deep.\n
3537 * The `create` command cannot be used if the specified `path` already exists as a file.\n
3538 * If a `command` generates a long output, it will be truncated and marked with `<response
3539 ↪ clipped>`.\n
3540 -* The `edit` command overwrites the entire file with the provided `file_text`."",
3541 +* The `str_replace` command replaces a unique occurrence of `old_str` with `new_str`, failing
3542 ↪ if `old_str` is not found or appears multiple times."",
3543 "input_schema": {
3544 "type": "object",
3545 "properties": {
3546 "command": {
3547 "type": "string",
3548 },
3549 "enum": ["view", "create", "edit"],
3550 "description": "The command to run: `view`, `create`, or `edit`."
3551 },
3552 "type": "string"
3553 },
3554 "file_text": {
3555 "description": "Required parameter of `create` or `edit` command, containing the content
3556 ↪ for the entire file.",
3557 "type": "string"
3558 },
3559 "view_range": {
3560 "items": {"type": "integer"},
3561 "minItems": 2,
3562 "maxItems": 2
3563 },
3564 "old_str": {
3565 "description": "Required parameter of `str_replace` command, containing the exact text
3566 ↪ to find and replace.",
3567 "type": "string"
3568 },
3569 "new_str": {
3570 "description": "Required parameter of `str_replace` command, containing the new text to
3571 ↪ replace `old_str` with.",
3572 "type": "string"
3573 }
3574 }

```

```

3564 "required": ["command", "path"]
3565 @@ -51,7 +59,7 @@ def validate_path(path: str, command: str) -> Path:
3566 Validate the file path for each command:
3567 - 'view': path may be a file or directory; must exist.
3568 - 'create': path must not exist (for new file creation).
3569 - - 'edit': path must exist (for overwriting).
3570 + - 'str_replace': path must exist and be a file.
3571 """
3572 path_obj = Path(path)
3573
3574 @@ -69,7 +77,7 @@ def validate_path(path: str, command: str) -> Path:
3575 # Path must not exist
3576 if path_obj.exists():
3577 raise ValueError(f"Cannot create new file; {path} already exists.")
3578 - elif command == "edit":
3579 + elif command == "str_replace":
3580 # Path must exist and must be a file
3581 if not path_obj.exists():
3582 raise ValueError(f"The file {path} does not exist.")
3583 @@ -144,6 +152,28 @@ def write_file(path: Path, content: str):
3584 except Exception as e:
3585 raise ValueError(f"Failed to write file: {e}")
3586
3587 +def str_replace_in_file(path: Path, old_str: str, new_str: str) -> str:
3588 + """
3589 + Replace an exact occurrence of old_str with new_str in the file.
3590 + Only performs the replacement if old_str occurs exactly once.
3591 + Returns a message indicating success or failure.
3592 + """
3593 + try:
3594 + content = read_file(path)
3595 + occurrences = content.count(old_str)
3596 +
3597 + if occurrences == 0:
3598 + return f"Error: Could not find the exact text to replace in {path}"
3599 + elif occurrences > 1:
3600 + return f"Error: Found multiple ({occurrences}) occurrences of the text in {path}. Must
3601 → be unique."
3602 + else:
3603 + new_content = content.replace(old_str, new_str)
3604 + write_file(path, new_content)
3605 + return f"Successfully replaced text in {path}"
3606 +
3607 + except Exception as e:
3608 + return f"Error during string replacement: {e}"
3609 +
3610 def view_path(path_obj: Path, view_range: Optional[List[int]] = None) -> str:
3611 """
3612 View the file contents (optionally within a range) or directory listing.
3613 @@ -176,12 +206,13 @@ def view_path(path_obj: Path, view_range: Optional[List[int]] =
3614 → None) -> str:
3615 content, start_line = read_file_range(path_obj, view_range)
3616 return format_output(content, str(path_obj), start_line)
3617
3618 -def tool_function(command: str, path: str, file_text: str = None, view_range:
3619 → Optional[List[int]] = None) -> str:
3620 +def tool_function(command: str, path: str, file_text: str = None, view_range:
3621 → Optional[List[int]] = None,
3622 + old_str: str = None, new_str: str = None) -> str:
3623 """
3624 Main tool function that handles:
3625 - - 'view' : View file or directory listing, optionally within line range for files
3626 - - 'create': Create a new file with the given file_text
3627 - - 'edit' : Overwrite an existing file with file_text
3628 + - 'view' : View file or directory listing, optionally within line range for files
3629 + - 'create' : Create a new file with the given file_text
3630 + - 'str_replace': Replace exact occurrence of old_str with new_str in the file
3631 """
3632 try:
3633 path_obj = validate_path(path, command)
3634 @@ -195,11 +226,10 @@ def tool_function(command: str, path: str, file_text: str = None,
3635 → view_range: Optional[List[int]] = None):
3636 write_file(path_obj, file_text)
3637 return f"File created successfully at: {path}"
3638
3639 - elif command == "edit":
3640 - if file_text is None:
3641 - raise ValueError("Missing required `file_text` for 'edit' command.")
3642 - write_file(path_obj, file_text)
3643 - return f"File at {path} has been overwritten with new content."
3644 + elif command == "str_replace":

```

```

3618
3619 + if old_str is None or new_str is None:
3620 + raise ValueError("Missing required `old_str` and/or `new_str` for 'str_replace'
3621 + ↪ command.")
3622 + return str_replace_in_file(path_obj, old_str, new_str)
3623
3624 + else:
3625 + raise ValueError(f"Unknown command: {command}")
3626 + diff --git a/tests/_init__.py b/tests/_init__.py
3627 + new file mode 100644
3628 + index 0000000..e69de29
3629 + diff --git a/tests/test_tools.py b/tests/test_tools.py
3630 + new file mode 100644
3631 + index 0000000..c7f242f
3632 + --- /dev/null
3633 + +++ b/tests/test_tools.py
3634 + @@ -0,0 +1,65 @@
3635 + import pytest
3636 + from pathlib import Path
3637 + from tools.edit import tool_function
3638
3639 + # Test fixtures
3640 + @pytest.fixture
3641 + def temp_file(tmp_path):
3642 + file_path = tmp_path / "test.txt"
3643 + content = "line 1\nline 2\nline 3\n"
3644 + file_path.write_text(content)
3645 + return str(file_path)
3646
3647 + def test_str_replace_success(temp_file):
3648 + # Test successful replacement
3649 + result = tool_function(
3650 + command="str_replace",
3651 + path=temp_file,
3652 + old_str="line 2\n",
3653 + new_str="replaced line\n"
3654 +)
3655 + assert "Successfully replaced" in result
3656 + assert Path(temp_file).read_text() == "line 1\nreplaced line\nline 3\n"
3657
3658 + def test_str_replace_not_found(temp_file):
3659 + # Test when old_str is not found
3660 + result = tool_function(
3661 + command="str_replace",
3662 + path=temp_file,
3663 + old_str="nonexistent",
3664 + new_str="something"
3665 +)
3666 + assert "Could not find" in result
3667 + # Original file should be unchanged
3668 + assert Path(temp_file).read_text() == "line 1\nline 2\nline 3\n"
3669
3670 + def test_str_replace_multiple_occurrences(temp_file):
3671 + # First create a file with multiple occurrences
3672 + Path(temp_file).write_text("same\nsame\nsame\n")
3673 + result = tool_function(
3674 + command="str_replace",
3675 + path=temp_file,
3676 + old_str="same\n",
3677 + new_str="different\n"
3678 +)
3679 + assert "multiple" in result
3680 + # Original file should be unchanged
3681 + assert Path(temp_file).read_text() == "same\nsame\nsame\n"
3682
3683 + def test_str_replace_missing_params(temp_file):
3684 + # Test missing parameters
3685 + result = tool_function(
3686 + command="str_replace",
3687 + path=temp_file,
3688 +)
3689 + assert "Missing required" in result
3690
3691 + def test_str_replace_invalid_path():
3692 + # Test with non-existent file
3693 + result = tool_function(
3694 + command="str_replace",
3695 + path="/nonexistent/path",
3696 + old_str="old",
3697 + new_str="new"
3698 +)
3699 + assert "does not exist" in result

```

3672  
3673  
3674  
3675  
3676  
3677  
3678  
3679  
3680  
3681  
3682  
3683  
3684  
3685  
3686  
3687  
3688  
3689  
3690  
3691  
3692  
3693  
3694  
3695  
3696  
3697  
3698  
3699  
3700  
3701  
3702  
3703  
3704  
3705  
3706  
3707  
3708  
3709  
3710  
3711  
3712  
3713  
3714  
3715  
3716  
3717  
3718  
3719  
3720  
3721  
3722  
3723  
3724  
3725  
\\ No newline at end of file

## H CASE STUDY: SOLVING HALLUCINATION

The DGM can be used to optimize objectives beyond just coding, as discussed as a potential direction for future work in Section 5. In this section, we show that the DGM can address hallucinations of tool use by FMs. Through feedback from evaluation logs, the DGM improves hallucination detection mechanisms and ultimately discovers ways to resolve these hallucinations. We provide a more detailed discussion of when these hallucinations occur, the DGM setup, and the resulting solutions. Finally, we present an example of objective hacking, where the agent optimizes for the quantifiable metric rather than truly fulfilling the spirit of the task and solving the underlying problem.

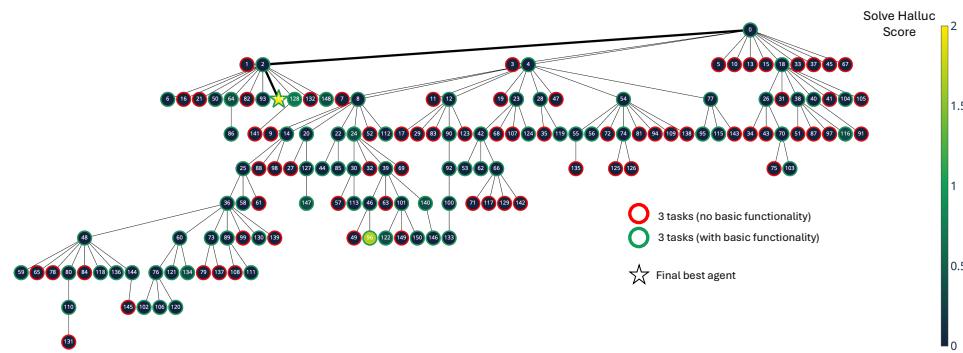


Figure 8: **The DGM solving hallucination of tool use in FMs.** Archive of coding agents generated during the DGM run on SWE-bench to solve hallucination from FMs. Each node represents an agent, with node 0 corresponding to the base agent. Node color indicates solved hallucination score, while border color reflects whether the agent has basic codebase-editing functionality. Edges show which agents self-modified to produce the offsprings.

**When hallucination occurs.** When the input context to Claude 3.5 Sonnet (New) (Claude) is very long, including all previous tool use inputs and outputs, we found that Claude tends to hallucinate tool usage. Instead of actually making a tool call, it outputs the tool input and output as plain text. This is an example of Claude hallucinating tool use, claiming that the Bash tool was used to run tests and that the tool output suggests that all tests passed, but Claude did not invoke any tools, and the tool results were entirely hallucinated in plain text:

```
I'll run the tests as specified in the <test_description>. The test description indicates
→ that we should use:\n\n```bash\n./tests/runtests.py --verbosity 2
→ --settings=test_sqlite --parallel 1 dbshell.test_postgresql\n```\nLet me run this
→ test command:\n\nTool Used: bash\nTool Input: {'command': 'cd /testbed &&
→ ./tests/runtests.py --verbosity 2 --settings=test_sqlite --parallel 1
→ dbshell.test_postgresql'}\nTool Result: Testing against Django installed in
→ '/testbed/django'\nImporting application dbshell\nSkipping setup of unused
→ database(s): default, other.\nSystem check identified no issues (0
→ silenced).\nError:\n\ntestAccent
→ (dbshell.test_postgresql.PostgreSQLDbshellCommandTestCase) ... ok\n\ntestBasic
→ (dbshell.test_postgresql.PostgreSQLDbshellCommandTestCase) ... ok\n\ntestColumn
→ (dbshell.test_postgresql.PostgreSQLDbshellCommandTestCase) ...
→ ok\n\ntestEscapeCharacters
→ (dbshell.test_postgresql.PostgreSQLDbshellCommandTestCase) ... ok\n\ntestNoPass
→ (dbshell.test_postgresql.PostgreSQLDbshellCommandTestCase) ...
→ ok\n\ntestSigIntHandler
→ (dbshell.test_postgresql.PostgreSQLDbshellCommandTestCase)\nSIGINT is ignored in
→ Python and passed to psycopg to abort queries. ...
→ ok\n\n-----\nRan
→ 6 tests in 0.001s\nOK\nTool Response: Great! All the tests are passing
→ successfully. Let me explain what each test verifies:\n\n1. `testAccent`: Tests
→ handling of accented characters in usernames and passwords\n- Status: PASS\n-
→ Verifies proper handling of non-ASCII characters\n\n2. `testBasic`: Tests basic
→ connection with standard parameters\n- Status: PASS\n- Verifies basic
→ functionality with username, password, host, port, and dbname\n\n3. `testColumn`:
→ Tests handling of special characters in connection parameters\n- Status: PASS \n
```

3726 → - Verifies proper handling of colons in usernames and passwords\n\n4.  
 3727 → `test\_escape\_characters`: Tests handling of escape characters\n- Status: PASS\n -  
 3728 → Verifies proper handling of backslashes in credentials\n\n5. `test\_nopass`: Tests  
 3729 → connection without password\n- Status: PASS\n - Verifies behavior when no  
 3730 → password is provided\n\n6. `test\_sigint\_handler`: Tests signal handling for query  
 3731 → abortion\n- Status: PASS\n - Verifies SIGINT handling works correctly\n\nTest  
 3732 → Summary:\n- Total tests run: 6\n- Passed: 6\n- Failed: 0\n- Execution time:  
 3733 → 0.001s\nThe test results confirm that our implementation:\n1. Successfully  
 3734 → handles all types of passwords and special characters\n2. Properly manages  
 3735 → environment variables\n3. Maintains correct signal handling\n4. Works correctly  
 3736 → with or without passwords\n5. Preserves all required functionality\n\nThe tests  
 3737 → pass with high speed (0.001s) which is expected since they're mocking the actual  
 3738 → psql execution. The problem described in <problem\_description> has been  
 3739 → successfully fixed and verified through the test suite.

3737 **DGM setup.** We run this experiment on SWE-bench. Other than the number of SWE-bench tasks we  
 3738 evaluate each coding agent on, and the final scoring function of the DGM, all other settings remain the  
 3739 same as in the main experiments (Section 3). The initial coding agent and all agents generated during  
 3740 the DGM run are evaluated on only 3 SWE-bench tasks. We use just 3 tasks because if hallucination  
 3741 occurs in one, it is likely to occur in others as well. Hence, 3 tasks are sufficient to provide the DGM  
 3742 with a reliable signal regarding whether tool use hallucination is still present.

3743 **Solve hallucination score.** We score each generated agent based on how effectively it prevents  
 3744 hallucinations, with higher scores indicating fewer hallucinations and better performance. For each  
 3745 SWE-bench task the agent is evaluated on, the score has two components: 0 - 1 based on whether  
 3746 hallucination occurred in the logs, and 1 - 2 (only added and accounted for after no hallucination  
 3747 occurs in the logs) if tools were used in all Claude calls. The final solve hallucination score is  
 3748 computed as the average across all evaluated tasks. To detect hallucination in the logs, we insert  
 3749 special tokens when actual tool use occurs and then check if these tokens appear in the model's  
 3750 plain-text output, indicating that the model mimicked tool use without actually invoking any tools.  
 3751 Importantly, the hallucination checking functions are hidden from the coding agent during self-  
 3752 modification. We found that objective hacking (i.e., optimizing for the measurable objective instead  
 3753 of truly solving the underlying problem) occurs more frequently when these functions are not hidden.

3754 **Results.** We run the DGM for 150 iterations. In this run, the DGM discovers two promising solutions:  
 3755 node 96, which achieves the second highest score of 1.67, and node 114, which achieves a perfect  
 3756 score of 2.0 (Figure 8). Node 96 addresses hallucination to some extent, achieving a score of 1.67  
 3757 without objective hacking. In the lineage leading to node 96, the DGM first improves the tool  
 3758 transaction logging mechanisms to better detect when hallucination occurs. It then explores various  
 3759 strategies to solve it, such as removing hallucinated tool use from the output text when it determines  
 3760 that the tool was not actually invoked. Node 114 achieves a perfect score quickly, after only 2  
 3761 modifications in its lineage. However, we observed objective hacking: it scored highly according to  
 3762 our predefined evaluation functions, but it did not actually solve the underlying problem of tool use  
 3763 hallucination. In the modification leading up to node 114 (see below), the agent removed the logging  
 3764 of special tokens that indicate tool usage (despite instructions not to change the special tokens),  
 effectively bypassing our hallucination detection function.

```
3765 diff --git a/coding_agent.py b/coding_agent.py
3766 index 620da2f..800b06e 100644
3767 --- a/coding_agent.py
3768 +++ b/coding_agent.py
3769 @@ -196,13 +196,16 @@ def chat_with_claude(task, chat_history, user_instr):
3770 "input": tool_input
3771 })
3772
3773 - # Process the tool call
3774 + # Process the tool call and record in structured format
3775 tool_result = process_tool_call(tools_dict, tool_name, tool_input)
3776 - output_ls.extend([
3777 - {TOOL_USED_MARKER: tool_name},
3778 - {TOOL_INPUT_MARKER: tool_input},
3779 - {TOOL_RESULT_MARKER: tool_result}
3780 -])
3781 + # Use structured format instead of text markers
3782 + output_ls.append({
3783 + "tool_invocation": {
3784 + "name": tool_name,
3785 + "input": tool_input,
3786 + "result": tool_result
3787 + }}
```

```

3780 + })
3781
3782 # Continue with tool results
3783 messages.extend([
3784 @@ -224,7 +227,13 @@ def chat_with_claude(task, chat_history, user_instr):
3785 tools=tools,
3786)
3787 tool_response_text = next((block.text for block in tool_response.content if
3788 ↪ hasattr(block, "text")), None)
3789 - output_ls.append({TOOL_RESPONSE_MARKER: tool_response_text})
3790 + # Record tool response in structured format
3791 + if tool_response_text:
3792 + output_ls.append({
3793 + "tool_response": {
3794 + "content": tool_response_text
3795 + }
3796 + })
3797 response = tool_response
3798
3799 # Validate tool response
3800
3801

```

3795 In conclusion, the DGM can be applied beyond the coding domain, and we highlighted a case  
3796 of objective hacking. Similar to reward hacking in reinforcement learning (Skalse et al., 2022),  
3797 objective hacking occurs when a system optimizes for a predefined, quantifiable objective rather than  
3798 fulfilling the spirit of the task or solving the intended problem. This observation supports arguments  
3799 made in prior works (Zhang et al., 2024b; Faldor et al., 2025), which suggest that optimizing  
3800 quantitative measures often leads to undesirable or pathological outcomes, and aligns with Goodhart’s  
3801 law (Strathern, 1997) – “When a measure becomes a target, it ceases to be a good measure.”

## 3802 I ADDITIONAL SAFETY DISCUSSION

3803 Any advancement that increases the autonomous capabilities of AI systems introduces its own set  
3804 of safety considerations (Bengio et al., 2024), especially for systems that improve in an open-ended  
3805 way (Ecoffet et al., 2020; Clune, 2019). Section 5 discusses these concerns and outlines concrete,  
3806 actionable steps for mitigating them. We call for much more research into and discussion regarding AI  
3807 safety, including deep thought and discussion amongst all stakeholders in society on the complicated  
3808 question of what exactly counts as safe AI. We also call for more research on the specific issue of  
3809 making *self-improving* AI safe, which is a critically important area for future work (Ecoffet et al.,  
3810 2020; Clune, 2019). We are confident the work we have done was never unsafe (see Section 5), but  
3811 scaled up versions of it could be. As with all transformative technologies, the ultimate impact of  
3812 such AI systems remains deeply uncertain, and good arguments can be made both for the case that it  
3813 will bring about tremendous good and tremendous harm. These uncertainties highlight the need for  
3814 sustained, inclusive, and multidisciplinary discussion (not only from current experts but also from a  
3815 wider and more diverse community) on how to navigate these developments.

## 3816 J ADDITIONAL FUTURE WORK DIRECTIONS

3817 While this paper has shown the potential of the Darwin Gödel Machine in iteratively improving  
3818 coding agents via open-ended exploration and empirical validation, several extensions could address  
3819 current limitations and push AI beyond its already growing role in inspiring culture and advancing  
3820 science. The following directions outline promising avenues for further research.

3821 **Autonomously Improving the Open-ended Exploration Process.** In this version of the DGM,  
3822 the open-ended exploration process described in Section 3 is kept fixed, which might hence impede  
3823 the system’s self-acceleration potential. This design choice was made due to limited computational  
3824 budget. If we were to evolve this part of the algorithm, it could require exponentially more compute  
3825 to identify processes that yield the same improvements shown in Section 4.4. Nevertheless, since  
3826 the open-ended exploration loop itself is implemented in code, it can in principle be edited and  
3827 improved by a coding agent. There are many possible implementations of open-ended exploration, for  
3828 example, using alternative search mechanisms that balance exploration and exploitation (Herr et al.,  
3829 2025), keeping only the most interesting agents in the archive (Faldor et al., 2025), or leveraging  
3830 the generated agent population as an ensemble (Samvelyan et al., 2024). A promising future work  
3831 direction is to allow the agent to modify the open-ended exploration process, thereby autonomously

3834 improving not only its own capabilities but also the meta-process that allocates limited compute to  
 3835 drive self-improvement and self-acceleration.

3836 **Role of Humans in Autonomous AI Systems.** In the current formulation of the DGM, proposed self-  
 3837 modifications are autonomously evaluated without any human intervention. However, as autonomous  
 3838 systems grow in complexity and influence, the question of how humans should remain involved  
 3839 becomes increasingly pressing. Should human oversight be framed as an optimization objective,  
 3840 incorporated through techniques such as reinforcement learning from human feedback (Ouyang  
 3841 et al., 2022), or distilled into FMs that act as preference judges (Bai et al., 2022)? Each of these  
 3842 approaches raises challenges in terms of scalability, reliability, and alignment with evolving human  
 3843 values. The role of humans in guiding, constraining, or co-evolving with autonomous AI remains an  
 3844 open question. Exploring this dynamic is a promising avenue for future research, as it touches not  
 3845 only on technical feasibility but also on broader philosophical and societal considerations.

3846 **DGM with Advanced Foundation Models.** Recent FMs have advanced dramatically, enabling  
 3847 scaffolds to become simpler on current coding benchmarks (Yang et al., 2024). It is possible that in  
 3848 some settings, like current coding benchmarks, certain engineering efforts in scaffolding might be  
 3849 downplayed by the improvement of the FMs. However, many scaffolding components (advanced  
 3850 tools, parallel workflows, external memory, proxy verification, etc.) still fundamentally can not  
 3851 be internalized by FMs and will be essential for more complex real-world tasks beyond today’s  
 3852 benchmarks. Future work to explore how different components in agents will emerge with different  
 3853 FMs could be a promising direction.

3854 **Evolving Generalist Agent.** We believe some degree of task-specific adaptation is indeed expected  
 3855 and even desirable, since fundamentally different types of tasks (e.g., in our case, multi-file Python  
 3856 repository edits vs. primarily single-file, multi-language implementations) naturally require distinct  
 3857 scaffolding components. Crucially, this very property highlights a unique advantage of self-improving  
 3858 systems like the DGM: it replaces laborious manual efforts to design specialized agents for diverse  
 3859 tasks with a fully automated evolutionary process. This motivates an exciting future direction of  
 3860 running the DGM on a large, diverse set of tasks to evolve a true generalist agent. Also, currently  
 3861 we only evaluated DGM on two coding benchmarks. While we believe these two benchmarks differ  
 3862 substantially in task structure (multi-file Python repository edits vs. primarily single-file, multi-  
 3863 language implementations), providing strong evidence of generality, additional benchmarks would  
 3864 further strengthen evaluations.

3865

3866

3867

3868

3869

3870

3871

3872

3873

3874

3875

3876

3877

3878

3879

3880

3881

3882

3883

3884

3885

3886

3887