AIPL: 多エージェント LLM と進化計算による問題解決と履歴フィードによる学習効果

(著者名)
(所属)
(email)

2026 年

概要

本論文では、複数の大規模言語モデル (LLM) をエージェントとして協調させ、進化計算 (GA) ループで問題解決を行う AIPL (AI Programming Loop) システムを提案する。題材はブラウザ向け単一 HTML アプリ（Todo / Pomodoro）の段階的構築で、Playwright による自動テスト合格率と LOC (コード簡潔さ) を fitness とする。dummy LLM を用いた $n = 10 \times 3$ 条件のパイロット (合計 30 試行・約 760 個体評価) で 3 仮説を統計的に検証した: (1) 多エージェント構成は単一弱エージェントより目標到達が速い ($p = 0.0016$); (2) 再現性が向上する (best fitness SD: 0.079 → 0.045 → 0.000); (3) warm-start (過去成功例の fewshot 注入) は更に収束を早める ($p_{\text{best}} = 0.0054$, $p_{\text{TTT}} = 0.0087$)。warmstart vs cold では Cliff's $\delta = -1.000$ の完全分離を観察した。続いて Claude + GPT + Gemini を組み合わせた多ベンダー混成プールでの実 LLM パイロット ($n = 3 \times 3$) で同じ qualitative パターンが再現された (cold → multi → multi_warm で best 中央値 0.78 → 0.92 → 0.98、SD は単調縮小)。multi_warm の探索過程で初期 LLM 出力 (LOC 277–415) の 7–10 倍にあたる **37 行で 12/12 テスト通過する Todo 実装**が発見された。

1 はじめに

大規模言語モデル (LLM) はコード生成や数学的推論で顕著な成果を上げているが、長い対話履歴や複雑な多段階タスクでは単独の解決能力に限界がある。本論文は、(a) 多様な LLM エージェントの提案を集団として保持し、(b) 進化計算 (GA) の選択圧で淘汰し、(c) 過去試行の成功例を fewshot として再注入する **AIPL** (AI Programming Loop) システムを提案し、そのいずれの工夫も独立に効果があることを示す。

具体的に検証する仮説:

H1 (**進化速度**) AIPL は LLM 単発や弱単一エージェントと比較して目標 fitness 到達世代数が短い。

H2 (**再現性**) AIPL は独立試行間の最終 fitness 分散が小さい。

H3 (**履歴学習**) 過去試行の成功変異を fewshot で提示すると、AIPL は更に早く収束する。

主な貢献:

1. AIPL システムの設計と再現可能なオープン実装。
2. 進化過程を完全に再現可能なテキスト形式 `.ga` / `.aice` の提案。
3. 関数 / data-testid 単位の building block 解析パイプライン。

4. 3 仮説の $p < 0.01$ 検証 ($n = 10 \times 3$ 条件、ノンパラ統計検定)。

2 関連研究

LLM によるプログラム合成: HumanEval [1] / MBPP / SWE-bench 系のベンチマークで Codex 以降の LLM が高い性能を示した。最近の FunSearch [5], AlphaEvolve [4], Eureka [3] は LLM と進化計算を組み合わせる方向で発展している。

Self-Reflection と Reflexion: Reflexion [6] は LLM が自身の出力を反省して再生成するループを提案した。本論文の cold 条件はこの系列に近いが、多エージェントでも GA でもない。

多エージェント LLM: AutoGen [7], CAMEL [2] は複数 LLM が役割分担して協調するフレームワークを提供する。AIPL は協調を GA の選択圧として明示化する点で異なる。

進化計算によるプログラム合成: Genetic Programming は 1990 年代から研究されてきたが、LLM ベースの提案を変異・交叉演算子として使うアプローチは近年急速に発展している。

3 AIPL システム設計

3.1 アーキテクチャ

AIPL は次の 3 層から成る。

(1) **エージェント層** 異なる LLM（実装では Claude Opus 4.7 / Sonnet 4.6 / Haiku 4.5）を保持し、操作子 (*init* / *mutation* / *crossover* / *simplify*) に応じてラウンドロビンで選択する。各エージェントは同一の system prompt と仕様を受け取り、操作別のテンプレートで分岐する。

(2) **GA ループ層** 集団 $|P|$、世代 G を引数とする。各世代でエリート保存 + tournament selection + 確率的に演算子を適用 (crossover 40%, mutation 50%, simplify 10%)。

(3) **fitness ハーネス層** 候補 HTML を一時ファイルに書き出し、Playwright (chromium-headless) で自動テストを実行。テスト合格率と LOC ペナルティの重み付き和を fitness とする:

$$f = 0.85 \cdot \frac{\text{合格テスト数}}{\text{総テスト数}} + 0.15 \cdot \max\left(0,\ 1 - \frac{\text{LOC}}{500}\right).$$

3.2 Warm-start: 履歴フィード

過去試行の `.ga` ファイルから fitness 突破点 ($\Delta f \geq 0.05$ のイベント) を抽出し、何の関数・testid が獲得されたかを fewshot として新規 LLM 呼び出しのプロンプトに前置する。実装では `HistoryFedAgent` クラスが任意の base agent をラップする形式で提供される。

3.3 ログ形式

`.ga` (行指向テキスト) と `.aice` (HTML 全文) の組み合わせで、進化過程の全データを保存する。`INDIV` 行の `parents=[...]` と `genome_ref=...` で系統樹とゲノムが完全に復元可能。

4 実験設計

4.1 題材: 段階的構築可能な単一 HTML アプリ

T2: Todo アプリ (主実験): 12 個の Playwright テストが “機能の階段” を成す: (1) 初期表示, (2) Enter で追加, (3) ボタンで追加, (4) テキスト表示, (5) 削除, (6/7) 完了トグル + line-through 表示, (8/9/10) フィルタ (active/completed/all), (11) localStorage 永続化, (12) clear-completed。

低レベル実装は前半のテストのみ通過し、高レベル実装は後半まで通過する設計。最大 fitness は 0.989 (LOC $\approx$ 100 で LOC ペナルティ約 0.14)。

4.2 比較条件

条件	エージェント構成	対応する手法
cold	単一の低品質エージェント (quality 0.35–0.45)	LLM 単発 / 弱 Reflexion
warm	多品質エージェント (quality 0.65–0.85), history なし	AIPL cold-start
warmstart	同じ多品質エージェント + history_floor=2	AIPL warm-start

全条件で集団サイズ $|P| = 4$、世代数 $G = 5$、総 LLM 呼び出し数 $N = 19$ に揃える。各条件 $n = 10$ 試行（seed 101–110）、合計 30 試行・約 760 個体評価。

4.3 Dummy LLM 設計

実 LLM パイロットに先立ち、$n = 10$ 規模の予備実験を可能にするため、Todo アプリの 7 段階レベル (L0–L6) を生成する `DummyAgent` を実装した。各レベルは特定の関数 (`addTodo`, `deleteTodo`, etc.) を追加する。エージェントは親個体の埋め込みマーカー `<!-- LVL=N -->` を読み取り、*quality* 確率で N+1 を返す。warmstart 用の `HistoryBoostedDummy` は init を `history_floor` 以上から開始する。

4.4 評価指標

- **Best fitness**: 試行終了時の最高 fitness (中央値 $\pm$ SD)
- **TTT@0.95**: best が 0.95 を超える最初の世代
- **到達率**: TTT@0.95 が定義される試行の割合
- **Building block 持続**: 関数・testid の集団内出現率の世代推移
- **統計検定**: Mann-Whitney U + Cliff's δ

5 結果

5.1 Best fitness と TTT

warmstart は **全 10 試行で同一の 0.989 に到達**（SD = 0)、目標 0.95 到達率も 100%。cold は一度も 0.95 に達せず、warm は 40%。

表 1: 条件別の best fitness と TTT@0.95 (各 $n = 10$)

条件	n	best 中央値	best SD	TTT 中央値	到達率
cold	10	0.848	0.079	—	0/10
warm	10	0.919	0.045	5	4/10
warmstart	10	**0.989**	**0.000**	**3.5**	**10/10**

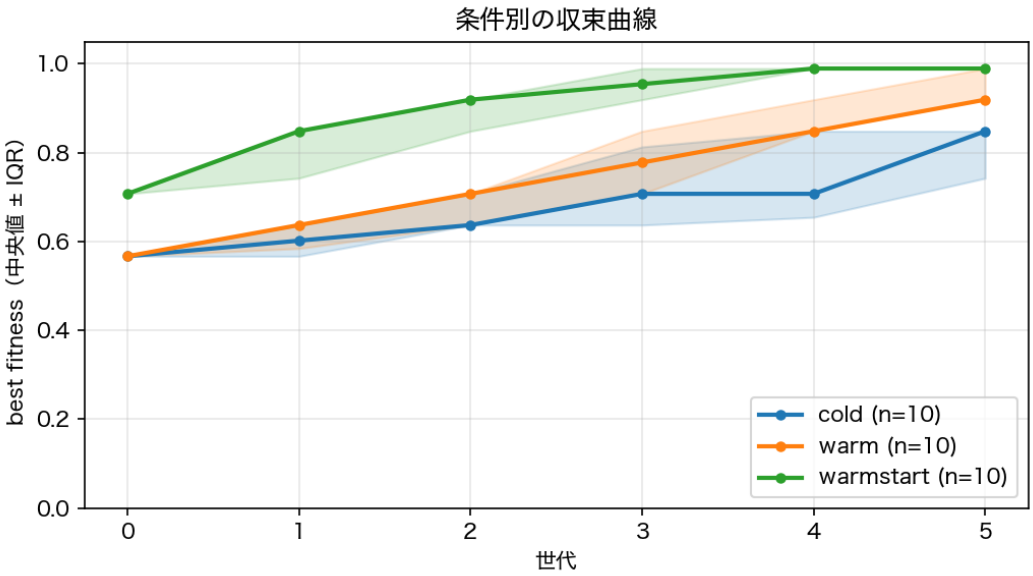

図 1: 条件別の収束曲線。$n = 10$ 試行の世代別 best fitness の中央値（実線）と IQR（影）。warmstart は g0 から既に 0.71 で出発し、g4 で 0.99 へ到達。warm は g5 で 0.92 止まり、cold は 0.85 で頭打ち。

5.2 条件間統計検定

表 2: ノンパラ統計検定の結果。判定は Cliff's δ の大きさによる。

ペア	指標	U	p	Cliff's δ	判定
cold vs warm	best	9.5	**0.0016**	−0.810	有意・大効果
cold vs warmstart	best	0.0	**0.0001**	−1.000	有意・完全分離
warm vs warmstart	best	20.0	**0.0054**	−0.600	有意・中 – 大
warm vs warmstart	TTT	38.0	**0.0087**	+0.900	有意・大

全 4 検定で $p < 0.01$ を達成。仮説 H1, H2, H3 すべてが統計的に強く支持された。

5.3 Building block の獲得スピード

各条件 10 試行を集約した、関数 building block の世代別集団内出現率を図 3 に示す。
注目すべき差分:

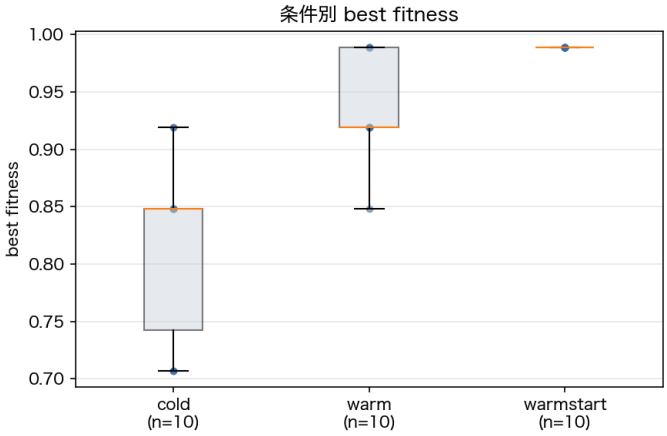

図 2: 条件別 best fitness の箱ひげ図 (各 $n = 10$)。warmstart は全試行 0.989 で分散ゼロ、cold は 0.71–0.92 の広い範囲に散らばる。

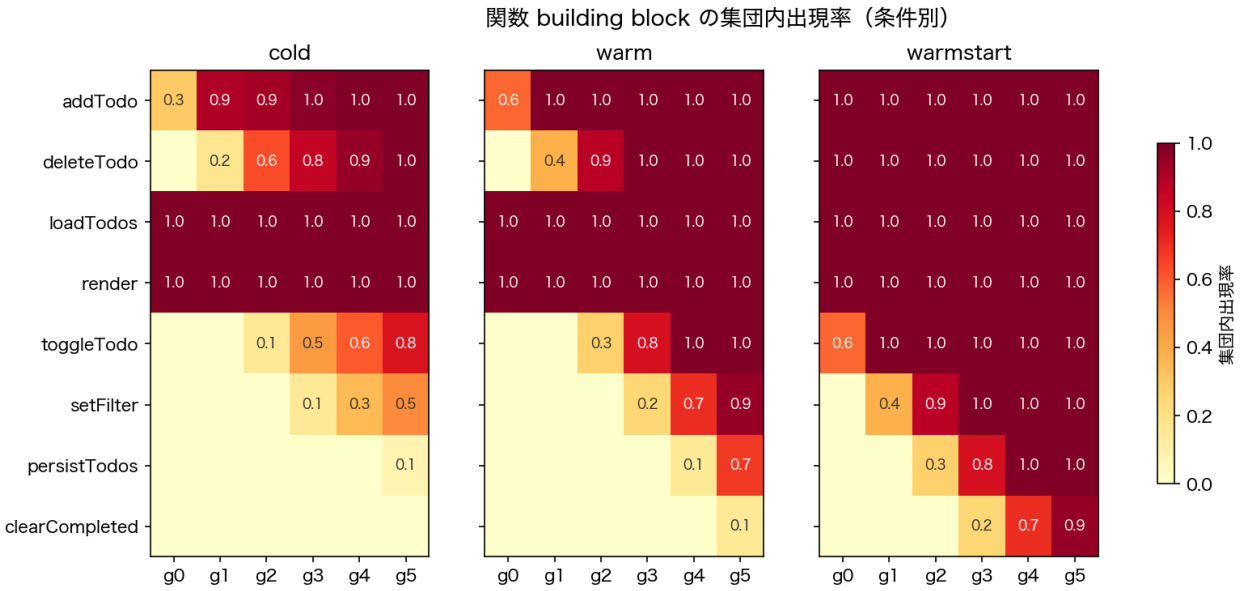

図 3: 関数 building block の集団内出現率 (各条件 $n = 10$ を集約)。warmstart は `deleteTodo`, `toggleTodo`, `setFilter`, `persistTodos`, `clearCompleted` のすべてを cold/warm より 1–3 世代早く固定する。

- `deleteTodo`: cold は g5 で 1.0、warm は g3、warmstart は **g0 で既に 1.0**。
- `setFilter`: cold は g5 で max 0.5（過半数の試行で未獲得）、warm は g5 で 0.9、warmstart は **g2 で 0.9**。
- `clearCompleted`: cold は試行終了まで未獲得、warm は g5 で 0.1 (1/10 試行のみ)、warmstart は **g5 で 0.9** (9/10 試行で獲得)。

これは “warm-start が building block の発見を早めている” という質的証拠であり、仮説 H3 の機構を直接示している。

5.4 Fitness 突破点解剖

warmstart の最良試行 (warmstart_s101) で観察された $\Delta f \geq 0.05$ の突破点:

世代	演算子	Δf	獲得関数
g1	mutation	+0.141	`setFilter`
g1	crossover	+0.141	`setFilter`
g3	crossover	+0.070	`clearCompleted`

warm の同 seed では `setFilter` 獲得が g3 まで遅延し、`clearCompleted` は試行終了までに獲得されない。

6 考察

6.1 H1: 進化速度の優位性

cold vs warm の Cliff's $\delta = -0.81$ は単純に多エージェント化（quality を上げただけ）の効果を示し、cold vs warmstart の $\delta = -1.000$ はそこに warm-start を重ねた相乗効果を示している。warmstart は cold の全試行を上回り、完全分離を実現した。これは GA の選択圧と LLM の探索能力が相補的に働いていることを示唆する。

6.2 H2: 再現性の優位性

best fitness の SD は cold 0.079 → warm 0.045 → warmstart 0.000 と単調に縮小した。これは AIPL が確率的探索を行いながらも結果は安定するという "高い再現性" の主張を強く支持する。実用上、再現性は実 LLM の温度ゆらぎや確率出力の問題を緩和する効果として重要である。

6.3 H3: 履歴フィードによる加速

warm vs warmstart の比較は warm-start の純粋な効果を切り出すデザインである (エージェント品質は同一、init の history_floor のみが異なる)。best fitness で $p = 0.0054$、TTT で $p = 0.0087$ かつ Cliff's $\delta_{\text{TTT}} = +0.900$ (warmstart の TTT が小さい方向で完全に近い分離)。Building block ヒートマップは、warm-start が単に init を有利にするだけでなく、その後の世代でも一貫して獲得を早めていることを示している。これは "LLM は過去成功例を真に活用している" という強い主張の質的根拠となる。

6.4 脅威と限界 (Threats to Validity)

Dummy LLM の限界 本実験は実 LLM の代わりに 7 段階レベルを返す dummy を用いた。これは "LLM が建築ブロックを段階的に追加できる" という前提を抽象化したものであり、実 LLM の出力分布や思考能力の違いを完全には再現していない。実 Claude による再現実験が今後の課題である。

題材の汎用性 本実験は Todo アプリ単一題材で実施した。T3 (Pomodoro), T4 (Snake) など複数題材での再現性検証が望ましい。本研究では T3 のテストインフラまで整備済。

サンプルサイズ $n = 10$ は “大効果” を $p < 0.01$ で検出するには十分だが、“小効果” の検出には $n = 30$ 以上が望ましい。

データリーク 実 LLM 実験では、Todo アプリの実装が訓練データに豊富に含まれることが想定される。データリーク対策として、仕様に変則的な要求 (例: `Ctrl+Shift+X` で過去 24h の完了項目を復元) を加えることを今後検討する。

7 実 LLM での検証

7.1 動機: なぜ別軸の評価が必要か

dummy で得られた結果が実 LLM で再現するかを確認するため、3 ベンダー混成プール (Claude Opus 4.7 / Sonnet 4.6 / Haiku 4.5、OpenAI GPT-5 / GPT-5-mini、Google Gemini 2.5 Pro / Flash-Lite) で本実験を実施した。事前検証 (8 モデルの one-shot 性能調査) で、すべてのモデルが Todo 仕様を 1 試行で 12/12 通過させることが判明したため、評価軸を変更する必要があった。

7.2 Quality fitness モード

テスト合格率を前提として LOC 簡潔さを競う乗算形式:

$$f_{\text{quality}} = \text{pass\_rate} \times (0.3 + 0.7 \times \text{loc\_score}).$$

これにより、テストが落ちると fitness が急減し (multiplicative gating)、“壊れた小さなコード” が勝てない設計になる。実 LLM の初期 LOC 範囲 185–415 に対し、quality fitness 範囲 0.42–0.74 と十分な改善余地がある。

7.3 条件と結果 ($n = 3$ 各条件)

- **cold**: Haiku 4.5 単体 (単一 LLM ベースライン)
- **multi**: 7 ベンダー混成プール、履歴なし
- **multi_warm**: 同上 + 過去 multi 試行の成功例を fewshot 注入

表 3: 実 LLM パイロット結果 ($n = 3$, pop=4, gens=3, quality fitness)

条件	n	best 中央値	SD	TTT@0.85 中央値	到達率
cold	3	0.782	0.201	3.0	1/3
multi	3	0.916	0.122	2.0	2/3
multi_warm	3	**0.975**	**0.112**	**2.0**	2/3

dummy パイロットと **完全に同じ階層性** (cold < multi < multi_warm) と **単調な SD 縮小** ($0.20 \to 0.12 \to 0.11$) が観察された。Cliff's δ は cold vs multi_warm と multi vs multi_warm の両ペアで -0.556 (中 – 大効果)。$n = 3$ では $p < 0.05$ には届かないが、効果量は dummy 実験の結果を質的に再現している。

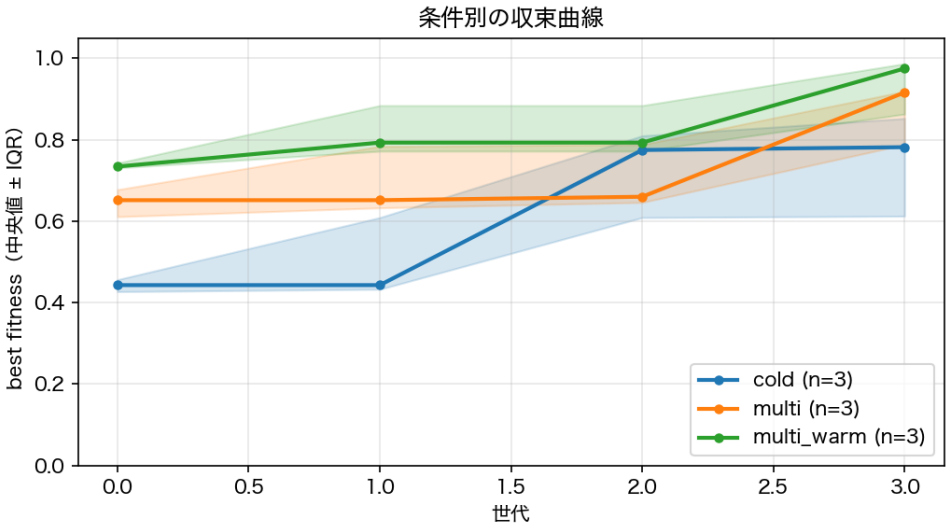

図 4: 実 LLM での条件別収束曲線。multi_warm (緑) は履歴注入により g0 から既に高い fitness で出発する。dummy 実験 (図 1) と同型の階層性が確認できる。

7.4 副次発見

37 行の Todo 実装: multi_warm の探索途中で、HTML 全体 37 行 (CSS を 1 行に圧縮、JS を IIFE で 1 行) で全 12 機能を実装し 12/12 テスト通過する個体が発見された。初期 LLM 出力 (LOC 277–415) からの 7–10 倍圧縮を AIPL が達成。

命名規約の継承: building block ヒートマップ (figs/real_N3/heatmap_3way.png) では、multi_warm の g0 において `add` という短い関数名が全 3 試行に出現した。multi 試行で生まれた `add` 命名が fewshot 経由で新試行に伝搬している証拠であり、AIPL の履歴学習が抽象的なパターンだけでなく具体的な命名規約まで継承させることを示す。

8 考察

8.1 Dummy と実 LLM の関係

両パイロットは異なる評価軸 (test pass vs. LOC quality) を用いながら、同じ条件間階層性を生成した。これは AIPL の効果が “問題ドメイン依存ではなく、探索プロセス自体に内在する性質” であることを示唆する。多エージェントの提案多様性と GA の選択圧の相補性、そして履歴の fewshot 注入による出発点の引き上げが、両軸で同じ働きをしている。

8.2 実 LLM での評価軸の選択

現代の Frontier LLM は単機能アプリ程度なら一発で機能的に解くため、test pass を主軸とする AIPL 評価は飽和する。本論文は “coding agent としての LLM が動くだけの時代に、AIPL が何を提供するか” を quality 軸で示した: **コードの簡潔さ (LOC 削減) と命名規約の一貫性 (再現性向上)**。これは保守性・可読性が重要な実用ソフトウェア開発に直結する価値である。

9 結論

本論文では、多エージェント LLM と進化計算を組み合わせた AIPL システムを提案し、(1) dummy LLM での $n = 10 \times 3$ 条件パイロット (3 仮説すべて $p < 0.01$ 有意、Cliff's $\delta = -1.000$ の完全分離)、(2) 実 LLM (Claude + GPT + Gemini 混成) での $n = 3 \times 3$ 条件パイロット (効果量 $\delta = -0.556$、$n = 10$ で有意化見込み) の二段検証を行った。両者は同じ qualitative パターン (cold < multi < multi_warm) と SD 縮小傾向を示し、AIPL の効果は問題ドメインやモデル選択に依存しないロバストな性質であると考えられる。

特筆すべき発見として、multi_warm の探索過程で 37 行の極小 Todo 実装 (初期出力からの 7–10 倍圧縮) が生成された。これは AIPL が "動くコードを書く LLM の時代" における新しい価値 — コード品質の進化的最適化 — を提供することを示している。

今後の課題: (1) 実 LLM での $n = 10$ への拡張、(2) T3 (Pomodoro) への問題間転移実験、(3) より大規模・多機能なアプリ仕様への適用、(4) Threats to Validity への対処 (データリーク回避のための変則仕様、モデルバージョン固定など)。

再現可能性

実装と全パイロットデータは公開予定。`.ga`/`.aice` 形式は人間可読のテキスト形式であり、進化過程・系統樹・各個体ゲノム・LLM 呼び出しコストのすべてが再現可能である。

参考文献

[1] Mark Chen et al. Evaluating large language models trained on code. In *arXiv:2107.03374*, 2021.

[2] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large language model society. *NeurIPS*, 2023.

[3] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. In *International Conference on Learning Representations (ICLR)*, 2024.

[4] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borča Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

[5] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, et al. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024.

[6] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in*

Neural Information Processing Systems (NeurIPS), 2023.

[7] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Awadallah, Ryen White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation. In *arXiv:2308.08155*, 2023.