TimeMachine-bench: LLM は「あの日」のコードを最新環境に適応できるか？

藤井 諒 [1,2]　森下 睦 [2,1]　矢野 一樹 [1]　鈴木 潤 [1,3,4]
[1] 東北大学　[2] フューチャー株式会社　[3] 理化学研究所　[4] 国立情報学研究所 LLMC
is-failab-research@grp.tohoku.ac.jp

概要

既存のコードを最新の依存関係へ適応させる「マイグレーション」は，開発現場の日常的な重要課題であるにもかかわらず，その LLM による自動化は十分に検討されてこなかった．本研究では，実世界の Python プロジェクトにおける，リポジトリレベルのマイグレーション能力を評価するベンチマーク **TimeMachine-bench** を提案する．本ベンチマークは，任意のリポジトリから幅広いシナリオを自動で構築できる点に特徴がある．最先端の LLM を用いた評価の結果，過剰な修正や自身の過去の出力に起因するエラーパターンの反復といった，自律型エージェントの信頼性に関わる課題を明らかにした．

1　はじめに

大規模言語モデル (LLM) は，古典的な自然言語処理タスクのみならず，ソフトウェア工学の分野にも多大な影響を及ぼしている [1]．初めは局所的な関数補完に始まり，現在ではリポジトリを自律的に探索し，バグ修正や機能追加を行う高度なエージェントが登場するなど，開発プロセスに不可欠なパートナーとしてその地位を確立しつつある．この進化を支えるベンチマークも，SWE-bench [2] に代表されるリポジトリレベルの課題へと移行し，より高度な実用性が求められるようになっている．

しかしながら，既存ベンチマークの多くは，環境を経時的に変化しないものと仮定している点で実態に即していない．実際のソフトウェアは，絶えず更新されるライブラリやエコシステムの上に構築されており，その環境は常に流動的である．こうした環境変化への適応，すなわち「**マイグレーション**」は，ソフトウェアの健全性維持に欠かせない日常業務であるにもかかわらず，リポジトリレベルでの実用的な評価は十分に検討されてこなかった．

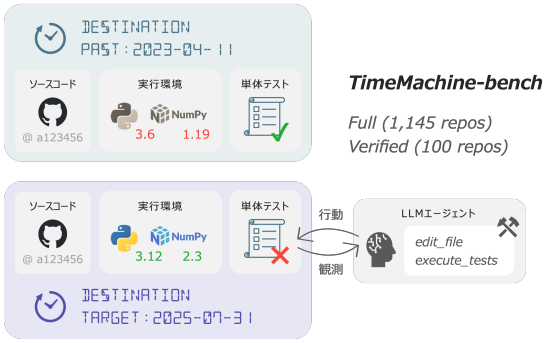

図 1　TimeMachine-bench の概要図．日付に基づく環境制御により，あるコードが正しく動作した過去の環境と，依存関係の更新によってエラーが生じる将来の環境を厳密に再現する．

そこで本研究では，実世界の Python プロジェクトにおけるマイグレーション能力を評価するベンチマーク **TimeMachine-bench** を提案する (図 1)．本ベンチマークの最大の特徴は，対象ライブラリの人手選定を完全に排し，GitHub 上の任意のリポジトリから実用的なマイグレーションシナリオを自動で構築できる点にある．また，データ構築パイプラインの自動化は，データの継続的な更新，すなわち「ライブ性」を提供する．これは，データ汚染 [3] が深刻な懸念となる LLM 時代において，評価の公平性を担保するために極めて重要な特性である．

実験では，解決可能性を保証した人手検証済みサブセット (**TimeMachine-bench-Verified**) を用いて，最先端の LLM を含む 11 のモデルを評価した．本研究の主な貢献は，(1) Python では初となるリポジトリレベル・マイグレーション評価ベンチマークの構築，(2) 自動構築による「ライブ性」の実現と人手検証済みサブセットの整備，(3) 11 モデルの包括的評価による現状の到達点と課題の明示，の 3 点に集約される．なお，本研究の再現性とコミュニティの継続的な発展のため，構築したベンチマークおよび実装を公開する．[1)]

1)　https://github.com/tohoku-nlp/timemachine-bench

2 関連研究

LLM によるコード生成能力の評価は，HumanEval [4] に代表される関数レベルのタスクから，実世界の開発ワークフローを反映したより複雑なタスク (例: SWE-bench [2]) へと移行している．しかしながら，既存ベンチマークの多くは環境が不変であることを前提としており，ライブラリやエコシステムの絶え間ない更新といった，ソフトウェアの動的な性質を考慮できていない．

このような課題に対して，近年では，ライブラリのバージョンを制約条件として，各時点の仕様に適応したコードを生成する能力を評価する LibEvolutionEval [5] や，バージョン間移行タスクを扱う VersiCode [6] が提案されている．しかし，これらの先行研究では，対象が単一行の補完や単一ファイル内の局所的な編集に限定されており，修正が複数ファイルに横断して存在するケースや，依存関係の連鎖的な更新に伴う段階的なエラー解決のプロセスを十分に反映していない．

リポジトリレベルのマイグレーションを扱う研究は極めて限定的であり，著者らの知る限り，Python を対象とした評価フレームワークは本研究が初である．Java プロジェクトを対象とした MigrationBench [7] は本研究と目的を共有するが，評価対象のライブラリやバージョンを人手で選定・固定している点に課題が残る．対照的に，本研究の手法は任意のリポジトリから実用的なマイグレーションシナリオを自動で構築可能であり，スケーラビリティやデータ汚染の懸念を原理的に解消している．

3 TimeMachine-bench

本節では，ベンチマーク構築の核となる環境再現と評価指標について，その概要を述べる．[2)]

3.1 任意の日付制約による環境の再現

マイグレーション能力の評価にあたっては，あるコードが正しく動作した過去の環境と，依存関係の更新によってエラーが生じる将来の環境という「2 つの断面」を正確に再現する必要がある．先行研究では `NumPy` や `pandas` 等の主要なライブラリを恣意的に選定し，特定の API の削除など，個別の修正事例をバージョン間で比較する「バージョンベース」の手法が主流であった．しかしながら，このような手法はエコシステム全体へのスケールが困難であるだけでなく，依存関係の連鎖に起因する，プロジェクト全体の整合性を捉えきれない．

このような課題に対して，本論文では個別のバージョン追跡に代わり，共通軸である「日付」に基づいて環境全体を制御するという，シンプルかつ明快なアプローチを提案する．具体的には，`pip` 等の依存関係解決器に対して，特定の日付以降にリリースされたパッケージを隠蔽したインデックス情報を提供する．これにより，解決器側のアルゴリズムを変更することなく，あたかも「過去のある日付」にいるかのように依存関係を解決させ，再現性の高い環境断面の構築を可能にする．この「依存関係解決器のタイムトラベル」とも呼べるアイデアをもって，本ベンチマークを TimeMachine-bench と呼称する．

3.2 ベンチマークの構築と検証

TimeMachine-bench は，The Stack v2 [8] より抽出した，実行可能な単体テストを含む Python プロジェクトに対して，2 つの断面におけるテスト実行結果を比較することで構築される．[3)] 実験の結果，過去環境ですべてのテストケースが成功したリポジトリのうち，36.8%のケースで依存関係の更新に伴うテストの失敗が確認された．ここから，スタックトレースの解析により，エラーの根本原因がユーザコードに起因しないものを除外し，最終的に 1,145 件からなる **TimeMachine-bench-Full** を構築した．

しかしながら，Full データにはサードパーティライブラリの実装に起因する計算誤差など，コード修正のみでは解決困難な課題も含まれる．安易なダウングレードという「ショートカット」を排し，システムの健全性維持というマイグレーションの本来の目的を厳密に評価すべく，所与の環境下での解決可能性を保証した人手検証済みサブセット **TimeMachine-bench-Verified** (100 件) を構築した．具体的には，実務を含む 8 年以上の Python 利用経験を持つ著者の 1 名が検証を担当し，問題の解決可能性の確認，およびテストを通過する最小の修正 (Gold Edit) をアノテーションした．アノテーションに際しては，1 件あたり最大 2 時間の制限を設け，解決までに要した時間に応じて Easy (10 分以内, 64 件), Medium (1 時間以内, 30 件)，Hard (2 時間以内, 6 件) の難易度ラベルを付与した．[4)]

2) 実装上の詳細については付録 B.1 に記載した．

3) 移行元を The Stack v2 上の各リポジトリのコミット時点，移行先を 2025 年 7 月 31 日とした．

4) タスクの実例は付録 A に記載．

表 1 TimeMachine-bench-Verified における評価結果．pass@1, prec@1 および難易度ラベル別の正解数 (括弧内は正解率) を示す．難易度別の問題数は Easy: 64, Medium: 30, Hard: 6 である．

カテゴリ	モデル	pass@1 (%)	prec@1 (%)	難易度ラベル別正解数 (正解率) Easy	Medium	Hard
プロプライエタリ	Claude Sonnet 4	**99.0**	**78.0**	**64** (100.0)	**30** (100.0)	**5** (83.3)
	Claude 3.5 Sonnet v2	91.0	66.8	61 (95.3)	25 (83.3)	**5** (83.3)
	GPT-5	91.0	54.2	62 (96.9)	27 (90.0)	2 (33.3)
	GPT-4o	76.0	61.4	57 (89.1)	19 (63.3)	0 (0.0)
オープン	Qwen3-Coder-480B [9]	90.0	70.1	62 (96.9)	26 (86.7)	2 (33.3)
	Qwen3-235B [9]	87.0	69.1	62 (96.9)	24 (80.0)	1 (16.7)
	Qwen3-32B [9]	53.0	44.1	40 (62.5)	13 (43.3)	0 (0.0)
	Llama-4-Maverick	76.0	63.2	56 (87.5)	20 (66.7)	0 (0.0)
	Llama-3.3 [10]	52.0	44.0	40 (62.5)	12 (40.0)	0 (0.0)
	DeepSeek-V3.1 [11]	75.0	61.4	52 (81.3)	21 (70.0)	2 (33.3)
	gpt-oss-120b (low) [12]	55.0	33.8	36 (56.3)	19 (63.3)	0 (0.0)

3.3 評価指標

マイグレーションの本質は新環境への適応であり，コード品質の改善 (リファクタリング) とは本質的に役割が異なる．両者を同時に行うことは変更の意図を不明瞭にし，レビューコストの増大や予期せぬバグの混入のリスクを増大させる．したがって，単にテストを成功させるだけでなく，「環境に適応するための最小限の変更」を正確に特定する能力を評価することが極めて重要となる．そこで本研究では，実行ベース評価で標準的に用いられる pass@k [4] を拡張した 2 つの評価指標を定義する．

- pass@1(n, m) : n 回以内の LLM 呼び出しおよび m 回以内のテスト実行での成功率 (効率性).
- prec@1(n, m) : 人手アノテーション (Gold Edit) に対するモデル修正行の適合率 (最小性).

なお，プログラムの等価性判定は一般に決定不能であり，テストコードの意味的な同一性を機械的に判定することは困難である．テストロジックの破壊 (例: `assert True` への書き換え) による不当なスコア上昇を防ぎ，評価の妥当性を担保するため，本研究ではモデルによるテストコードの編集は禁止した．

4 実験設定

最先端の LLM を含む 11 のモデルについて，TimeMachine-bench-Verified の 100 件のデータを用いて評価を行った．具体的には，プロプライエタリモデルとして Claude Sonnet 4, Claude 3.5 Sonnet v2, GPT-5, GPT-4o，オープンモデルとして Qwen3-Coder (480B), Qwen3 (32B, 235B) [9], Llama-3.3 [10], Llama-4-Maverick, DeepSeek-V3.1 [11], および gpt-oss-120b (low) [12] を選定した．なお，すべてのモデルについて最大出力長は 512 トークンとし，設定可能な場合においてはサンプリング温度を 0 とした．[5)]

リポジトリレベルのマイグレーションは未だ十分な検討がなされていない領域であり，標準的なアプローチも確立されていない．そこで本研究では，SWE-Agent [13] を拡張した，10 種類のツールによる ReAct エージェント [14] をベースラインとした．[6)] また，コンテキストの肥大化を避けるため，直近 5 ターンより前のツール出力は破棄し，推論過程のみを保持する履歴管理を実装した [13].

本実験では，LLM の最大呼び出し回数 $n = 100$, テストの最大実行回数 $m = 10$ とした．エージェントは，実行環境から得られるエラーログやファイルに関する観測結果を基に自律的にコードの修正を行い，すべてのテストケースが成功するか，いずれかの上限回数に到達するまで動作を継続する．

5 結果と考察

タスク成功率の全体傾向 表 1 に，TimeMachine-bench-Verified における各モデルのスコアを示す．まず特筆すべき点としては，Claude Sonnet 4 の高い成功率 (99.0%) が挙げられる．また，オープンモデルの飛躍も著しい．具体的には，Qwen3-Coder-480B (90.0%) や Qwen3-235B (87.0%) が，前世代の旗艦モデルである Claude 3.5 Sonnet v2 (91.0%) と同等の成功率を記録した．この結果は，高度な実用性が問われる実世界的なエンジニアリングタスクにおいて，オープンモデルとプロプライエタリモデルの性能差が急速に縮まっていることを示している．

5) その他のハイパーパラメータはデフォルト値とした．
6) 実装したツールの一覧は付録 B.2 に記載．

時系列的な推論の創発　このような高い成功率の一方で，その推論の質にもまた注目すべきである．例えば，Claude Sonnet 4 は特定の専門的なドメインで利用されるようなライブラリ (pysnmp) が関与する事例においても，*"The issue is that in pysnmp 7.x, the asyncore module has been replaced with asyncio"* のように，モジュールの変遷を明示的に言語化してみせた．マイグレーションに特化したリソースの欠如，とりわけバージョン間の代替関係を構造化した学習データに乏しい現状を考慮すれば，このような「時系列を汲んだ」推論の創発は極めて意外性が高い．ただし，当該事例における移行は実際には v6 系で行われており，モデルの説明と事実の間には差異がある．したがって，この驚くべき推論能力が，正確な事実理解に基づくのか，「後付けの合理化」に過ぎないのかは慎重な見極めが必要である．

タスクの難易度による影響　各モデルの正解率は，gpt-oss-120b (low) の例外を除き，タスク難易度の上昇に伴って低下する一貫した傾向を示した．特に，Hard カテゴリにおいて多くのモデルの成功率が 50%を下回った事実は，人間にとっての難問が，LLM にとっても依然として解決困難な課題であることを示している．さらに，本実験で対象とした人手検証済みサブセット (Verified) は一人の専門家が現実的な時間 (2 時間) で解決できる程度の難易度に限定されていることにも留意されたい．すなわち，本実験で得られた正解率は，実世界の課題における性能の楽観的な上限であり，人間が短時間で対処可能な課題に対する自動化の実現可能性が実証されたと解釈するのが妥当である．

修正の的確さと編集の質　prec@1 による編集の最小性に着目すると，最も高い成功率を記録した Claude Sonnet 4 においても，そのスコアは 78.0 に留まっており，これは全修正の 20%以上が「課題解決に寄与しない」編集であったことを示している．また，成功率で同等のスコアを記録したモデル群 (Claude 3.5, GPT-5, Qwen3) の間でも，prec@1 では Qwen3-Coder の 70.1 に対し GPT-5 は 54.2 に留まるなど，編集の質において明確な差が確認された．

失敗パターンの分析　GPT-5 の prec@1 が低迷した要因を分析すると，編集ツールにおける行境界の認識に課題が見られた．具体的には，編集範囲の包含・排他的な判定を誤り，既存のコードと重複する行を挿入してしまう事例が観測された．また，70B 以下の小・中規模モデルにおいては，ライブラリに関する知識量の差以上に，マルチターンのツール利用における「出力の一貫性」に課題が見られた．とりわけ，一度生じた形式エラーが，次ターン以降の自身の推論において文脈内事例 (in-context example) として作用し，同一の誤りを再帰的に繰り返す「エラーの自己模倣」は注目に値する (表 2)．したがって，本ベンチマークは，マイグレーション能力を評価する枠組みとしてはもちろん，長大なコンテキストの下で一貫した動作を継続する，自律型エージェントの「信頼性」を検証する評価基盤としても，高い有用性を備えていると考えられる．

表 2　Qwen3-32B の事例：#20 の「ツール呼び出しの欠落」から，#21 の「思考過程の欠落」を経て，最終的に#22 で #20 と同一の誤りパターンに回帰する．

model#19 (✓)	`[text]` (略) `[tool_use]` *edit_file(...)*
tool#19	Edit succeeded. (後略)
model#20 (×)	**`[text]` I have corrected line 40... I'll continue...** (`[tool_use]` が欠落)
error#20	*You must use one of the available tools. Please try again.*
model#21 (×)	`[tool_use]` *edit_file(...)* (`[text]` が欠落)
tool#21	Edit succeeded. (後略)
model#22 (×)	**`[text]` I have corrected line 41... I'll continue...** (再度 `[tool_use]` が欠落)
error#22	*You must use one of the available tools. Please try again.*

6　おわりに

本研究では，実世界の Python プロジェクトにおける，リポジトリレベルのマイグレーションに焦点を当てた新たなベンチマーク **TimeMachine-bench** を提案した．最先端の LLM を含む 11 のモデルに対する評価の結果，高いタスク成功率と驚くべき推論能力が示された一方で，必要以上の修正や「エラーの自己模倣」といった，自律型エージェントの信頼性に関わる課題も明らかとなった．なお，本ベンチマークの核である自動構築パイプラインは言語非依存であり，広範なエコシステムへの適用が可能である．今後は，他言語への拡張の検討に加えて，単体テスト自動生成 (UTG) や環境構築の自動化技術をアドオンとして組み込むことで，より堅牢な評価基盤の構築を目指したい．本研究が，今後のマイグレーション研究を人手選定から解放し，自動構築に基づく実世界的な評価へとシフトさせる道標となることを期待する．

謝辞

本研究の一部は，JST ムーンショット型研究開発事業 JPMJMS2011-35 (fundamental research)，および，文部科学省の補助事業「生成 AI モデルの透明性・信頼性の確保に向けた研究開発拠点形成」，JSPS 科研費 JP23H03508 の助成を受けたものです.

参考文献

[1] Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. A Survey on Large Language Models for Code Generation. **ACM Transactions on Software Engineering and Methodology**, 2025.

[2] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can Language Models Resolve Real-world Github Issues? In **Proceedings of the Twelfth International Conference on Learning Representations (ICLR)**, 2024.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, et al. Language Models are Few-Shot Learners. In **Proceedings of the Advances in Neural Information Processing Systems 34 (NeurIPS)**, pp. 1877–1901, 2020.

[4] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, et al. Evaluating Large Language Models Trained on Code. **arXiv:2107.03374**, 2021.

[5] Sachit Kuhar, Wasi Uddin Ahmad, Zijian Wang, Nihal Jain, Haifeng Qian, Baishakhi Ray, Murali Krishna Ramanathan, Xiaofei Ma, and Anoop Deoras. LibEvolutionEval: A Benchmark and Study for Version-Specific Code Generation. In **Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)**, pp. 6826–6840, 2025.

[6] Tongtong Wu, Weigang Wu, Xingyu Wang, Kang Xu, Suyu Ma, Bo Jiang, Ping Yang, Zhenchang Xing, Yuan-Fang Li, and Gholamreza Haffari. VersiCode: Towards Version-controllable Code Generation. **arXiv:2406.07411**, 2024.

[7] Linbo Liu, Xinle Liu, Qiang Zhou, Lin Chen, Yihan Liu, Hoan Nguyen, Behrooz Omidvar-Tehrani, Xi Shen, Jun Huan, Omer Tripp, and Anoop Deoras. MigrationBench: Repository-Level Code Migration Benchmark from Java 8. **arXiv:2505.09569**, 2025.

[8] Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, et al. StarCoder 2 and The Stack v2: The Next Generation. **arXiv:2402.19173**, 2024.

[9] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, et al. Qwen3 Technical Report. **arXiv:2505.09388**, 2025.

[10] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, et al. The Llama 3 Herd of Models. **arXiv:2407.21783**, 2024.

[11] DeepSeek-AI. DeepSeek-V3 Technical Report. **arXiv:2412.19437**, 2024.

[12] OpenAI. gpt-oss-120b & gpt-oss-20b Model Card. **arXiv:2508.10925**, 2025.

[13] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In **Proceedings of the the Thirteenth International Conference on Learning Representations (NeurIPS)**, 2025.

[14] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In **Proceedings of the Eleventh International Conference on Learning Representations (ICLR)**, 2023.

図 2　TimeMachine-bench-Verified のタスク例．この例では pandas のアップデートに関連して，複数のランタイムエラーに段階的に対応する必要がある (難易度ラベル = Easy)．

表 3　実装したツールの一覧．各ターンにおいて，エージェントはいずれかのツールを選択して実行する．

ツール名	説明
list_dir	指定ディレクトリ内のファイル・サブディレクトリを一覧表示する．
search_dir	指定ディレクトリ配下の全ファイルを対象に正規表現による検索を実行する．
search_file	指定ファイル内で正規表現による検索を実行する．
view_file	指定ファイルを開き，特定行の前後 50 行を表示する．
edit_file	指定した行範囲を特定のテキストで置換する．
replace_all	指定ファイル内の正規表現に一致する箇所を特定の文字列で一括置換する．
revert_last	直前の編集を取り消し，ファイルを元の状態に戻す．
execute_tests	テストを実行し，実行ログの末尾 100 行を表示する．
search_last_log	直近のテストログに対して正規表現検索を実行する．
view_last_log	直近のテストログを開き，特定行の前後 50 行を表示する．

A　TimeMachine-bench のタスク例

図 2 は，TimeMachine-bench-Verified に含まれるタスクの一例である．この例は，リポジトリ `EBISPOT/gwas-sumstats-validator` において，2023 年 4 月 11 日の依存関係を起点として，将来の環境 (2025 年 7 月 31 日) へ移行するシナリオであるが，ここで特筆すべきは，単一の修正のみでは完結しない「連鎖的な課題解決」が要求される点である．具体的には，`pandas.append` の廃止に伴う初動のエラーを解決することで，初めてその背後に隠蔽されていた別の不具合が顕在化するという多段の構造になっている．このような動的な問題解決プロセスは，一過性のバグ修正を主眼とする既存のベンチマークとは一線を画しており，実世界のマイグレーションが内包する本質的な複雑さを反映している．

B　実装の詳細

B.1　実行環境の再現

各タスクの実行環境は，Docker コンテナを用いた隔離環境上に構築した．日付制約に基づく環境再現は，依存関係解決器の参照先を `pypi-timemachine`[7] に向け，公式リポジトリである PyPI へのアクセスを仲介・フィルタリングすることで実現した．また，設定ファイルに記載されたバージョン固定 (==) や上限制約により，バージョンが更新されないことを防ぐため，設定ファイルを解析し，移行先環境の構築時にはこれらの制約を解除した．同様の理由から，リポジトリ内にロックファイルが存在する場合には該当のファイルを削除した．環境構築およびテスト実行には，それぞれ最大 10 分のタイムアウトを設定した．加えて，外部 API に依存するテストケースなど，実行結果が非決定的となる事例を除外するため，テスト実行時にはネットワークアクセスを遮断した．

B.2　ツール一覧

表 3 に，本実験で構築したエージェントが利用可能なツールの一覧を示す．各ターンにおいて，エージェントはいずれかのツールを選択して実行し，ツールの自作や，ツール以外の方法で環境と対話することは不可とした．なお，Verified サブセットでは，人手の検証によりすべてのタスクが「リポジトリ内の既存ファイル」の編集のみで解決できることが担保されているため，本実験では新規ファイルの作成を目的としたツールは提供していない．しかしながら，一部のモデルでは存在しないファイル名と，行番号 1 を指定して *edit_file* ツールを呼び出すことで，実験用の一時ファイルを作成しようとする挙動も確認された．本実験ではこれらの操作はエラーとして処理されるが，このような観測は，未知の環境を調査する際の人間の探索行動とも符合する興味深い事象と言える．

7)　https://github.com/astrofrog/pypi-timemachine