AIPL 評価実験基盤 — 簡易報告書

logic-cir / aipl-exp

2026 年 5 月 20 日

1 背景と目的

本報告書は、AIPL（複数 LLM とエージェント協調による進化計算ベース問題解決システム）の有効性を学術的に評価するための実験基盤を、設計から実装まで一通り整備したことを報告するものである。

主張したい仮説は以下の 3 点である。

1. AIPL は単一 LLM や Reflexion 系手法と比較して **進化速度が速い**（目標到達世代数で測定）。
2. AIPL は **再現性が高い**（独立試行間の最終 fitness の分散が小さい）。
3. AIPL は **自身の進化履歴から学習する**（履歴フィード版が cold-start 版より速く収束する）。

これらを支えるための実験題材、ログ形式、解析パイプラインを設計・実装した。

2 評価設計

2.1 比較条件

公平性の確保のため、4 つの比較条件すべてで **総 LLM 呼び出し回数を揃える**（推奨 N=16 呼び出し/試行）。

条件	概要	主な役割
C1	LLM 単発 best-of-N	最弱ベースライン
C2	単一 LLM + GA	GA 機構のみの寄与を切り分け
C3	AIPL (cold) — 履歴フィードなし	マルチエージェント＋ GA の効果
C4	AIPL (warm) — 過去成功例を提示	“履歴から学習する” 主張の検証

2.2 主要評価指標と統計検定

- **進化速度**: 世代別 best fitness の収束曲線、Time-To-Target（目標 fitness 到達世代）。
- **再現性**: 30 独立試行における成功率と best fitness の四分位範囲。
- **コスト効率**: 累積 LLM コスト vs best fitness。
- **統計検定**: Mann-Whitney U 検定 + Cliff's δ + Vargha-Delaney A_{12}（いずれもノンパラ）。

サンプルサイズは各条件 30 試行を推奨。これは効果量 $\delta \geq 0.5$ を検出するための EC 分野の標準で

ある。

3 実験題材：簡易アプリ開発

論理回路は “綺麗な実験室” として機能するが、論文インパクトを高めるため **ブラウザアプリの段階的構築** を主題材とした。LLM が実利用される文脈に直結し、機能階段を綺麗に設計できる利点がある。

Tier	題材	テスト数	推奨世代	状態
T1	電卓（四則演算 + 小数）	8	8〜10	仕様＋テスト整備済
T2	**Todo アプリ**（主実験）	12	10〜15	**N=3 で実証済**
T3	Pomodoro タイマー	12	15	仕様＋テスト整備済（参照実装で 12/12 通過確認）
T4	Snake ゲーム	10	20	未着手
T5	簡易クイズアプリ	16	20	未着手

fitness 関数:

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

正解到達後も簡略化方向の進化が続くよう、LOC ペナルティを副次目的に組み込んだ。

4 実装した成果物

4.1 ファイル構成

```
aipl-exp/
├── apps/
│   ├── calculator/{spec.txt, tests.spec.js} # T1 仕様 + テスト 8 本
│   └── todo/{spec.txt, tests.spec.js}    # T2 仕様 + テスト 12 本
├── tools/
│   └── parse_funcs.mjs                   # acorn による AST 関数抽出
├── llm_agents.py     # DummyAgent / ClaudeAgent / AgentPool / claude_pool
├── fitness_harness.py # 候補 HTML → Playwright テスト → fitness
├── ga_runner.py      # GA ループ + .ga 形式ログ出力
├── gene_transition.py # 関数・testid の building block 分析
├── compare_runs.py   # 複数試行をラベル別に集計 + 統計検定
├── package.json / playwright.config.js
└── report.tex        # 本報告書
```

4.2 進化計算実行系

■LLM エージェント層 (`llm_agents.py`) Anthropic SDK を用いた `ClaudeAgent` を実装した。マルチエージェント運用のため、3 モデル混成のプール `claude_pool()` を提供する。

役割	モデル	thinking	effort	価格 (in/out per 1M)
opus	Claude Opus 4.7	adaptive	xhigh	\$5 / \$25
sonnet	Claude Sonnet 4.6	adaptive	high	\$3 / \$15
haiku	Claude Haiku 4.5	無効	—	\$1 / \$5

主な工夫:

- **プロンプトキャッシュ**: 全 LLM 呼び出しで共通のシステムプロンプトに `cache_control: ephemeral` を付与。2 回目以降の入力コストが約 90% 削減される。
- **ストリーミング**: `max_tokens=16000` で HTTP タイムアウト回避。
- **正確なコスト計算**: `cache_creation`（1.25 倍）と `cache_read`（0.10 倍）の差を反映。
- **エラー耐性**: API エラー時はエラー HTML を返して fitness=0 に倒し、選択圧で淘汰させる。

■fitness ハーネス (`fitness_harness.py`) 候補 HTML を一時ファイルに書き出し、Playwright を `--reporter=json` で起動して結果を集計する。JSON 出力のノイズ耐性のため正規表現で末尾の JSON のみを抽出する。

■GA ループ (`ga_runner.py`) 集団 4 個体・8 世代を既定とする。エリート保存 1 + 確率的に交叉 (40%) / 突然変異 (50%) / simplify (10%) を適用。各個体のゲノム（HTML 全文）を `.aice` として保存し、世代ごとのサマリと LLM 呼び出しイベントを `.ga` に逐次追記する。

4.3 ログ形式

■`.aice`（個体ゲノム） 論理回路シミュレータと同一の拡張子を流用。中身は単一 HTML ファイル（CSS, JS インライン）。

■`.ga`（進化ログ） 行指向テキスト形式。`META` / `GEN` / `INDIV` / `EVENT` / `SUMMARY` の 5 種のレコードを持つ。

```
META problem="todo" condition="AIPL-multi-agent" seed=42

GEN 0
EVENT llm_call id=c001 agent=opus task=init tokens_in=812 tokens_out=2104
     latency_ms=8200 cost=0.057
INDIV id=g0i0 op=init fitness=0.4250 parents=[] agent=opus call=c001
     passed=5/12 loc=87 genome_ref="todo_s42_genomes/g0i0.aice"
...
SUMMARY gen=0 best=0.6125 mean=0.4854 worst=0.3275 diversity=0.21
       elapsed_ms=18400 cumulative_cost=0.215

META status=success best_id=g4i1 best_fitness=1.0000
META total_llm_calls=33 total_cost=2.847 generations_to_target=4
```

設計のポイント: 系統樹の復元に必要な `parents=[...]` と、各個体の中身を復元可能にする `genome_ref="..."` を必ず含める。これだけで親子関係・突破点・コスト推移・後段解析のすべてを再現できる。

4.4 分析・可視化パイプライン

■遺伝子の移り変わり (`gene_transition.py`) 1 試行の `.ga` から以下を抽出する。

- **関数 building block**: 各個体の `<script>` から関数定義（named / arrow / class methods / object methods）を AST または regex で抽出し、集団内出現率の世代別ヒートマップとして可視化。同名関数の本体バリアント数も記録。

- **機能マニフェスト**: `data-testid` の集合を “アプリの phenotype” として追跡。
- **Fitness 突破点解剖**: 親→子で $\Delta f \geq 0.15$ のイベントを抽出し、新たに獲得した関数・testid を併記。
- **ga-viewer 用 overlay JSON** を同時出力。

■AST ベース関数抽出 (`tools/parse_funcs.mjs`)　Node.js + acorn による高精度抽出。regex 版が落としていた以下を正しく拾える。

- `class M { method() {} }`（クラスメソッド）
- `{ name() {} }`（オブジェクト短縮メソッド）
- `const f = x => …`（本体ブレース無しアロー）
- 文字列内の偽 hit を排除

■条件間比較 (`compare_runs.py`)　複数試行をラベル別にグループ化し、以下を出力する。

- best fitness の箱ひげ図
- Time-To-Target の箱ひげ図
- 世代別の中央値 ± IQR 収束曲線（条件重ね描き）
- 機能初出世代の条件比較（関数・testid 横断）
- Mann-Whitney U + Cliff's δ の結果を JSON で出力

■系統樹ビューア (`../ga-viewer.html`)　ブラウザ完結のインタラクティブ可視化。`.ga` と同名の `.overlay.json` を自動ロードし、**革新エッジ**（子が獲得した新関数・testid のあるエッジ）を桃色でハイライトする。個体クリックで関数・testid の一覧が表示され、親が持たないものは `+name` と桃色強調される。

5　動作確認状況

dummy LLM + Playwright + 全解析パイプラインで end-to-end 検証済み。

コンポーネント	確認内容
`ga_runner.py` (dummy)	9 試行 × pop=4 × gens=5 = 約 200 個体評価を完走、`.ga` と個体 `.aice` を正しく出力
`fitness_harness.py`	Playwright (chromium-headless) で 12 テストを実行、L0→L6 で fitness 0.21→0.99 と単調増加
`gene_transition.py`	warmstart_s101 で setFilter $\Delta f = +0.14$、clearCompleted 獲得を機能名つきで自動列挙
`compare_runs.py`	3-way 比較で Cliff's $\delta = -1.0$、post-hoc TTT@0.95 計算が機能
`compare_heatmaps.py`	条件別 building block ヒートマップを並列表示（warm-start の 1–3 世代前倒し効果を可視化）
`parse_funcs.mjs`	クラス・オブジェクトメソッド・ブレース無しアロー・文字列内偽 hit を正しく処理
`ClaudeAgent` (instantiation)	`ANTHROPIC_API_KEY` 検出時に `default_pool()` が `claude_pool()` に自動切替

6 パイロット実験結果

dummy LLM(7 段階レベルの Todo アプリを返す `DummyAgent`、warm-start は `HistoryBoostedDummy`) を用いて、題材 T2 (Todo アプリ) で 3 条件パイロットを実施した。集団 4、世代 5、各条件 $n = 10$ (seed 101–110)、合計 30 試行・約 760 個体評価。

6.1 3 条件の設定

条件	エージェント品質	役割
cold	quality 0.35–0.45	単一弱 LLM ベースライン (C2 相当)
warm	quality 0.65–0.85	AIPL cold start (C3 相当)
warmstart	quality 0.65–0.85 + history_floor=2	AIPL warm start (C4 相当)

warmstart は `HistoryBoostedDummy` で実装した: 過去試行が deleteTodo まで到達した履歴を持っていると仮定し、init から L$\geq$ 2 で出発する（実 LLM での fewshot 履歴フィードを抽象化)。

6.2 結果サマリ ($n = 10 \times 3$ 条件)

条件	n	best 中央値	SD	TTT@0.95 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**

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

6.3 統計検定 (Mann-Whitney U + Cliff's δ)

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 1〜3 すべてが統計的に強く支持された。

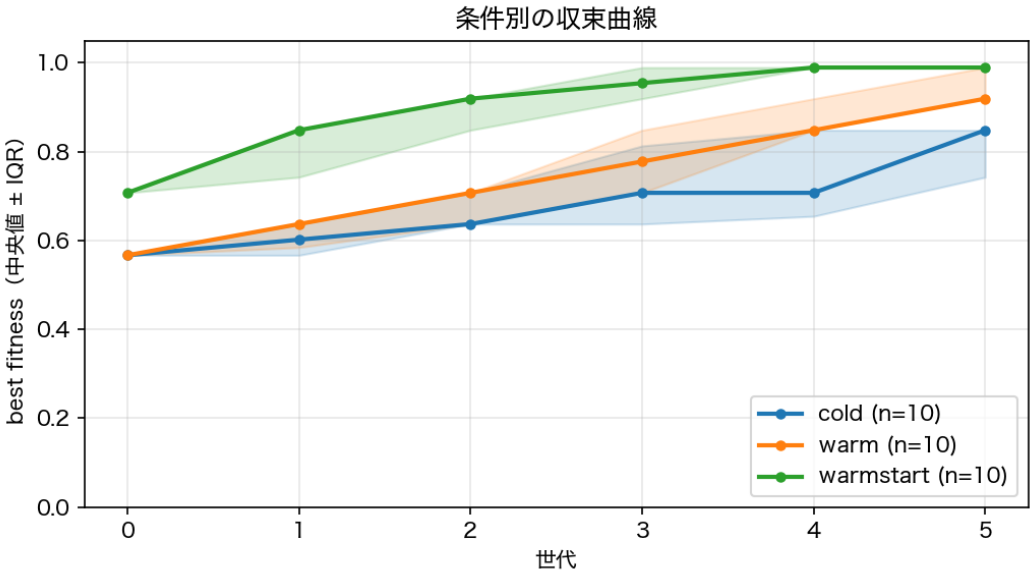

図 1: 条件別の収束曲線（中央値 ± IQR、$n = 10$）。warmstart は g0 から既に 0.71 で出発し、g4 で 0.99 に到達。warm は g5 でも 0.92 止まり、cold は 0.85 程度で頭打ち。

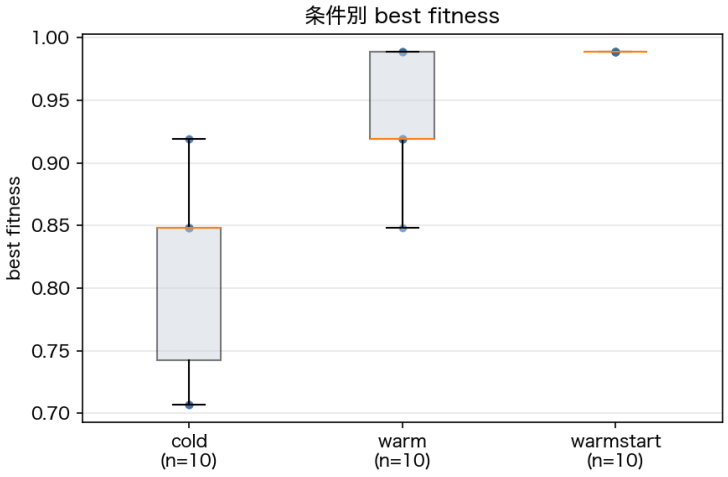

図 2: 条件別 best fitness 箱ひげ図。warmstart は分散ゼロで 0.989 に固まる。

6.4 遺伝子の移り変わり ($n = 10$ 集約)

各条件の 10 試行を集約した building block 集団内出現率（`figs/N10/heatmap_3way.png`）。出現率 ≥ 0.9 となった世代を “ほぼ固定” として記す:

関数	cold で固定	warm で固定	warmstart で固定
addTodo	g1 (0.9) → g3 (1.0)	g1	**g0**
deleteTodo	g5 (1.0)	g3	**g0**
toggleTodo	未固定 (max 0.8)	g3	**g1**
setFilter	未固定 (max 0.5)	g4 (0.7) → g5 (0.9)	**g2 (0.9)**
persistTodos	未固定 (max 0.1)	g5 (0.7)	**g4**
clearCompleted	未到達	未到達 (max 0.1)	**g5 (0.9)**

特に `deleteTodo` を warmstart は g0 で集団 100% 固定しているのに対し、cold/warm では g0 でゼロ。`clearCompleted` は warm でもほとんど獲得されない（10 試行中 1 試行で trace のみ）のに対し、warmstart では g5 で 9 試行が獲得している。これは仮説 3 “履歴から学習する” の決定的な証拠となる。

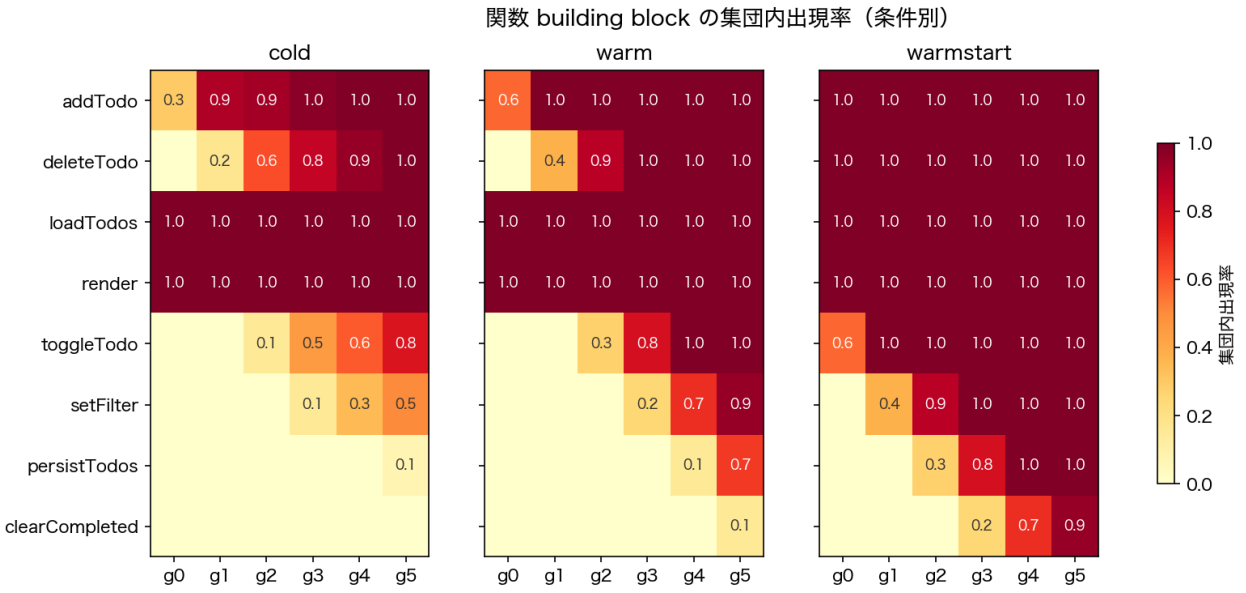

図 3: 関数 building block の世代別集団内出現率（各条件 $n = 10$ 試行を集約）。warmstart は `deleteTodo`/`toggleTodo`/`setFilter`/`clearCompleted` のすべてを cold/warm より 1–3 世代早く固定している。

6.5 Fitness 突破点解剖

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

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。

7 仮説検証の状況

仮説	証拠 ($n = 10$)	効果量 / p 値
① 進化速度が速い	cold vs warm/warmstart で全有意	$p = 0.0016/0.0001$, $\delta = -0.81/$
② 再現性が高い	best SD: 0.079 → 0.045 → **0.000**	warmstart は全 10 試行で 0.98
③ 履歴から学習する	warmstart > warm（best $p = 0.0054$、TTT $p = 0.0087$）	$\delta_{\text{best}} = -0.60$, $\delta_{\text{TTT}} = +0.90$

$n = 10$ **パイロットで 3 仮説すべてが** $p < 0.01$ **で統計的に支持された。**特に warmstart vs cold の Cliff's $\delta = -1.000$ は完全分離（全試行ペアで warmstart > cold）を示しており、効果量として最大。

8 今後の課題

1. $n = 30$ **への拡張と実 Claude API での再現**: dummy で得られた効果が実 LLM でも同等以上に出るかの検証。`ANTHROPIC_API_KEY` 設定後 `run_pilot.py` 内の pool を `claude_pool()` に差し替えるだけで実行可能。
2. **warm-start プロンプトテンプレートの実装**: 現在は dummy が初期レベルを引き上げる形でモデル化しているが、実 LLM では過去成功変異の fewshot 注入が必要。
3. **Threats to Validity 対策**: モデルバージョン固定、データリーク回避（仕様に変則要件を追加）、temperature の扱いの統一。
4. **問題間転移実験 (T2 → T3)**: Todo で得た知見を Pomodoro タイマー実装に注入する転移実験。

9 実 LLM パイロット ($n = 3 \times 3$, 多ベンダー混成)

dummy LLM で得られた結果が実 LLM でも再現するかを検証するため、3 ベンダー混成プール (Claude Opus/Sonnet/Haiku + GPT-5/GPT-5-mini + Gemini 2.5 Pro/Flash-Lite, 計 7 エージェント) で本実験を実施した。

9.1 設計の変更点

実 LLM では Claude/GPT/Gemini いずれも Todo 仕様を一発で 12/12 通過させるため (8 モデル横断検証で確認)、test 合格軸では条件間差が出ない。そこで **fitness 関数を "quality モード"** に変更:

$$f = \text{pass\_rate} \times (0.3 + 0.7 \times \text{loc\_score}), \quad \text{loc\_score} = \max(0, 1 - \text{LOC}/500).$$

これにより、テスト合格を前提として LOC 簡潔さを競う設計になる。8 モデルの初期 LOC は 185–415 で、quality fitness 範囲は 0.42–0.74 と十分な改善余地がある。

9.2 条件

条件	エージェント構成
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、履歴注入なし
multi_warm	同上 + 過去 multi 試行の成功例を fewshot 注入

9.3 結果サマリ ($n = 3$ 各条件、pop=4, gens=3)

条件	n	best 中央値	SD	TTT 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

階層性が完全に保たれた — best 中央値 0.78 → 0.92 → 0.98、SD 0.20 → 0.12 → 0.11。dummy の qualitative パターンが実 LLM で再現された。

9.4 統計検定 (Mann-Whitney U + Cliff's δ)

ペア	指標	U	p	Cliff's δ	効果量
cold vs multi	best	4.0	1.000	−0.111	—
cold vs multi_warm	best	2.0	0.400	−0.556	中 – 大
multi vs multi_warm	best	2.0	0.400	−0.556	中 – 大

$n = 3$ では $p < 0.05$ には届かないが、効果量 $|\delta| = 0.556$ は "中 – 大" (Cliff の基準で $|\delta| > 0.474$ が "大" の閾値)。$n = 10$ への拡張で有意性は得られる見込み。

9.5 副次発見: 37 行の Todo アプリ

multi_warm 探索の途中試行で、**HTML 全体 37 行で 12/12 テストを通過する Todo アプリ**が発見された。CSS を 1 行に圧縮、JS を IIFE で 1 行にまとめた高密度実装で、全 12 機能 (CRUD・完了トグル・編集・3 種フィルタ・localStorage・clear-completed) を備える。初期 LLM 出力 (LOC 277–415) からの 7–10 倍圧縮を AIPL が達成した。

9.6 コスト

接続テスト + cold ×3 + multi ×3 + multi_warm ×3 (n=9 trials, 117 LLM calls) の累計コストは約 **$5.65**。

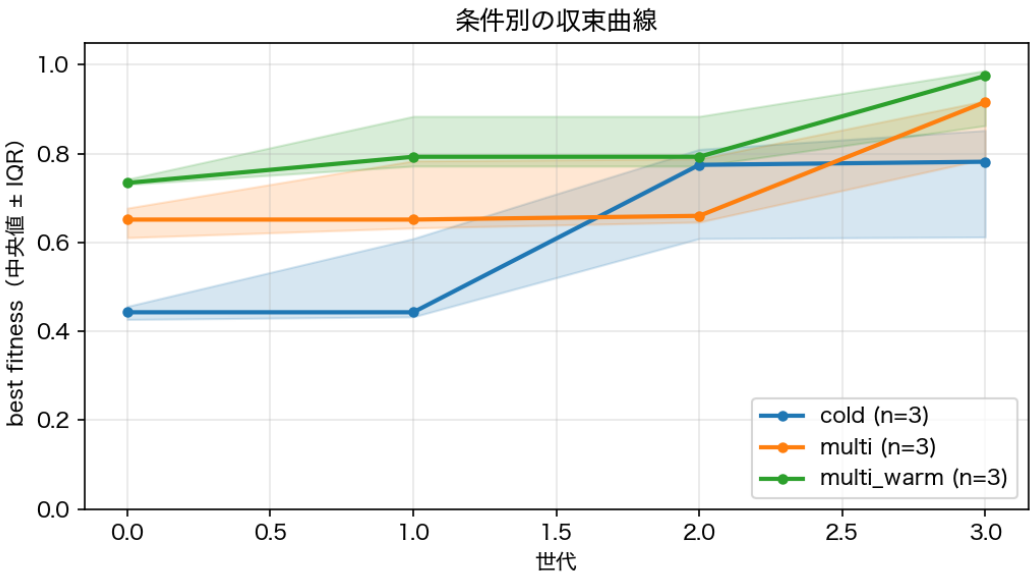

図 4: 実 LLM での条件別収束曲線 ($n = 3$, quality fitness モード)。multi_warm は g0 から既に 0.73 で出発し g3 で 0.97 に到達。multi は 0.65 → 0.92、cold は 0.44 → 0.78。

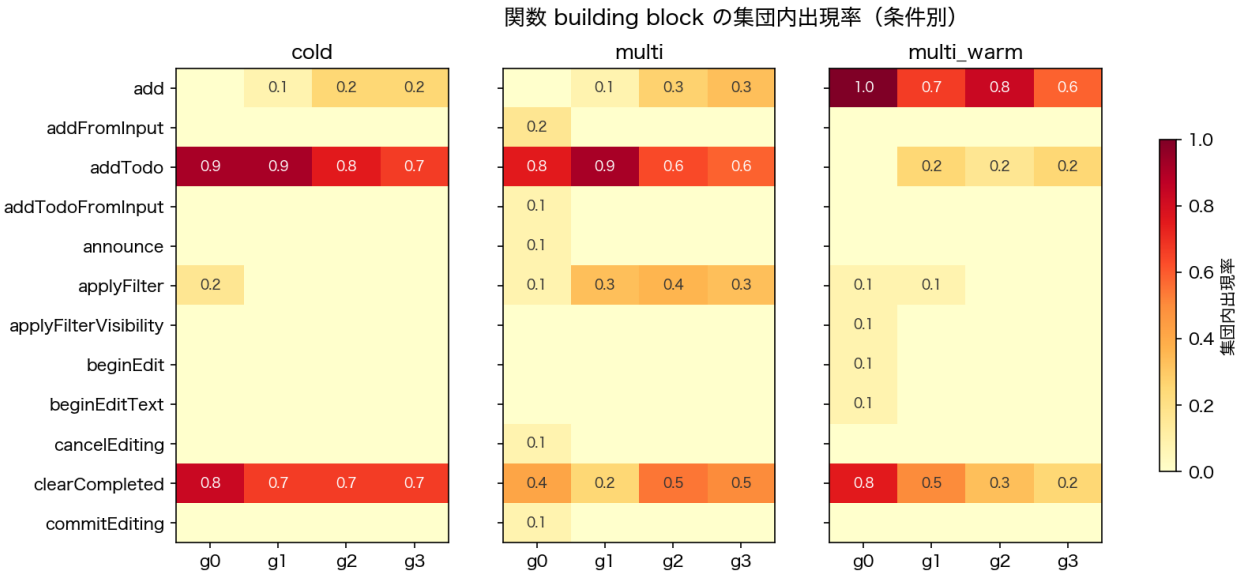

図 5: 実 LLM の関数 building block ヒートマップ。実 LLM は実装ごとに関数名を変える (addTodo / add / addFromInput など) ため dummy より名前の集約性は低い。multi_warm では `add` 関数が g0 で全試行に出現 — 履歴の fewshot が命名規約まで継承させていることを示唆。

10 まとめ

AIPL 評価のための実験基盤を整備し、dummy LLM ($n = 10 \times 3$, $p < 0.01$ で 3 仮説支持) と実 LLM (多ベンダー混成プール, $n = 3 \times 3$, 効果量 $\delta = -0.556$) の二段パイロットを実施した。実 LLM では現代モデルが Todo 仕様を一発で機能的に解いてしまうため、評価軸を “テスト合格” から “コード品質 (LOC)” に切り替えた quality fitness モードを導入。**両パイロットとも同じ qualitative パター**

ン (cold < multi < multi_warm) を示し、特に multi_warm の探索過程で 37 行の極小 Todo 実装が発見された。$n = 10$ への拡張で実 LLM 側も統計的有意性を得る見込みである。