# Agentic Context Engineering: Evolving Contexts for Self-Improving Language Models

**arXiv ID:** 2510.04618 **提出日:** 2025年10月6日 **著者:** Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, Kunle Olukotun **所属:** Stanford University, SambaNova Systems Inc., UC Berkeley **カテゴリ:** Machine Learning (cs.LG); Artificial Intelligence (cs.AI); Computation and Language (cs.CL) **DOI:** https://doi.org/10.48550/arXiv.2510.04618

## 概要

本論文は、大規模言語モデル(LLM)のコンテキスト適応を通じた自己改善フレームワーク **ACE (Agentic Context Engineering)** を提案しています。ACEは、コンテキストを「進化するプレイブック(Evolving Playbook)」として扱い、戦略の蓄積・精緻化・組織化を実現します。従来手法が抱える「簡潔性バイアス(Brevity Bias)」と「コンテキスト崩壊(Context Collapse)」という2つの重大な問題を、構造化された段階的更新により解決します。

AppWorldエージェントベンチマークで平均**+17.0%**の改善、金融分析タスクで平均**+12.8%**の向上を達成し、適応レイテンシを平均**86.9%削減**、トークンコストを**83.6%削減**する高い効率性を実現しています。特筆すべきは、ファインチューニングを必要とせず、実行フィードバックのみで自己改善が可能な点です。

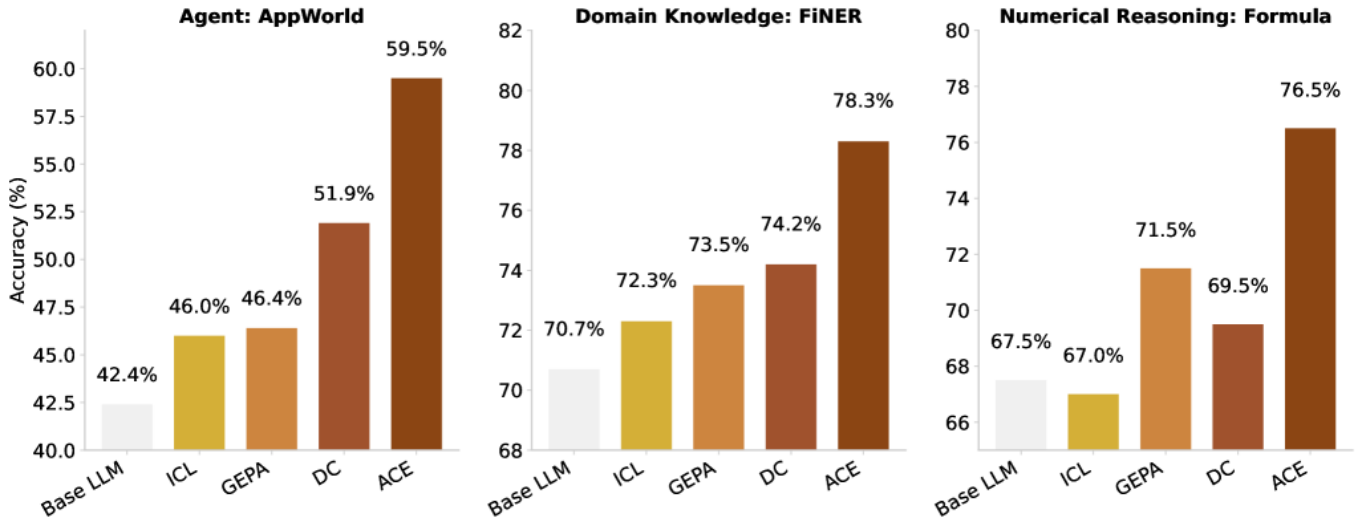



*図1: ACEは複数のベンチマークで一貫した性能向上を実現*

## 先行研究

### 背景と動機

現代のLLMアプリケーション（エージェント、複合AIシステム）は、モデルの重みを更新する代わりに、入力にシステムプロンプト、メモリ、証拠を直接組み込む「コンテキスト適応(Context Adaptation)」に依存しています。この手法には以下の利点があります：

- **解釈可能性**: コンテキストの変更を人間が直接確認・編集可能
- **ランタイムでの知識統合**: 再学習なしで新しい情報を追加

- **モジュール間での共有**: 複数のシステムで知識を再利用可能

長文脈LLMとKVキャッシュ再利用の進展により、コンテキスト適応が自己改善型AIシステム構築の中心的パラダイムとして浮上しています。

## 関連研究

**エージェントメモリ関連研究:**

- **AgentFly**: 継続的に進化するメモリフレームワーク
- **AWM (Agentic Workflow Memory)**: 過去の軌跡から蒸留された再利用可能なワークフロー
- **A-MEM**: Zettelkasten法に着想を得た動的に組織されたメモリシステム
- **Agentic Plan Caching**: テスト時キャッシング戦略

これらの研究は主にエージェント特化のメモリに焦点を当てていますが、ACEは「コンテキスト適応」というより広範な課題に取り組んでいます。

**プロンプト最適化関連研究:**

- **MIPROv2**: ベイズ最適化でシステム指示と文脈内実例を最適化
- **GEPA (Reflective Prompt Evolution)**: 実行トレースから学習するプロンプト進化
- **Dynamic Cheatsheet**: テスト時学習で適応的メモリを累積

これらの手法は、ACEが解決する「簡潔性バイアス」と「コンテキスト崩壊」の問題に悩まされていました。

## 既存手法の問題点

### 1. 簡潔性バイアス (Brevity Bias)

既存の最適化手法は「簡潔で広く適用可能な指示」を優先し、包括的な蓄積よりも短さを重視します。その結果：

- ドメイン固有のヒューリスティックが省略される
- 重要な失敗パターンが記録されない
- タスク特化の詳細な戦略が失われる

### 2. コンテキスト崩壊 (Context Collapse)

モノリシック書き直し（全体を一度に書き換える方式）により、反復的な圧縮で情報が段階的に喪失します。論文では具体的な例として：

- **ステップ60**: 18,282トークン、精度66.7%
- **ステップ61**: 122トークンに圧縮、精度57.1%に低下（ベースライン63.7%より悪化）

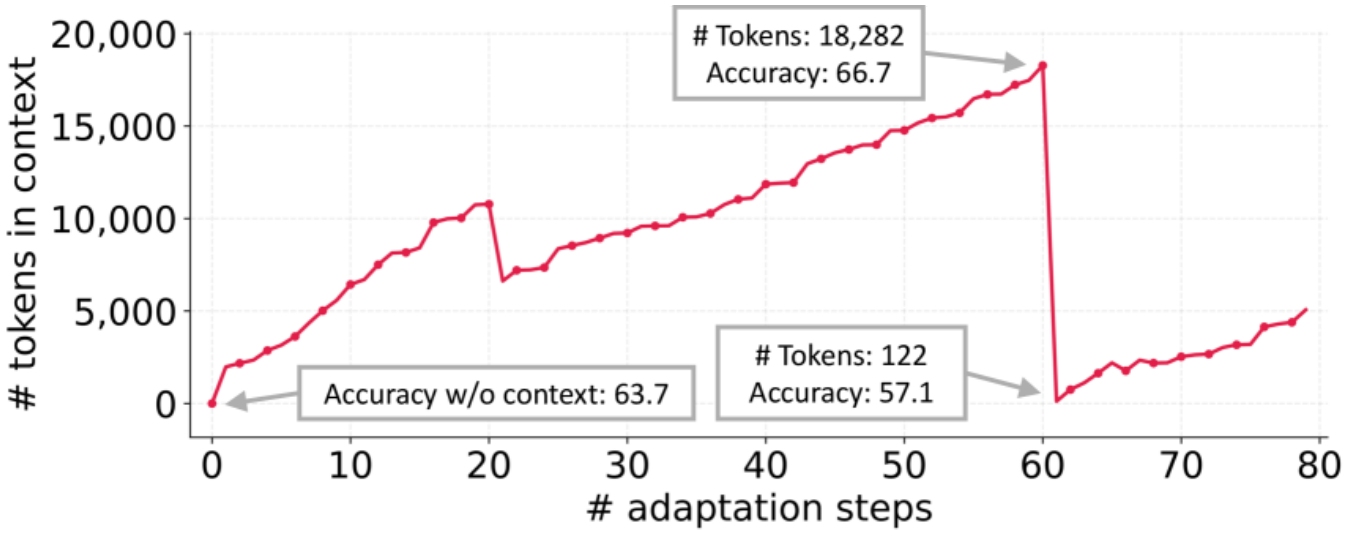



*図2: コンテキスト崩壊の現象。LLMによる単一段階での書き直しが短く情報量の少ない要約に圧縮される*

---

## 主要な貢献

1. **高性能な自己改善フレームワーク**
   - AppWorldエージェントタスクで**+10.6%平均改善**（オフライン適応で+17.0%）
   - 金融分析ベンチマークで**+8.6%平均改善**（FiNER: +7.6%, Formula: +18.0%）
   - オンライン適応で**+17.1%改善**を達成し、IBM CUGA（GPT-4.1ベース）を上回る
2. **効率的な適応メカニズム**
   - 適応レイテンシを平均**86.9%削減**
   - トークンコストを**83.6%削減**
   - ロールアウト数を**75.1%削減**
3. **ラベル不要の自己改善**
   - 実行フィードバックのみで有効に機能
   - ファインチューニング不要
   - 人間による解釈可能性を保持
4. **スケーラブルなアーキテクチャ**
   - 構造化された増分デルタ更新
   - 並列マージ可能な決定論的統合
   - 長文脈での効率的な推論

---

## 手法

### ACEの三層アーキテクチャ

ACEは「生成(Generation)、反省(Reflection)、キュレーション(Curation)」の3つの専門的役割を分離します。このデザインは人間の学習プロセス—実験、反省、統合—を反映しながら、単一モデルへのすべての責務の過負荷という問題を回避します。

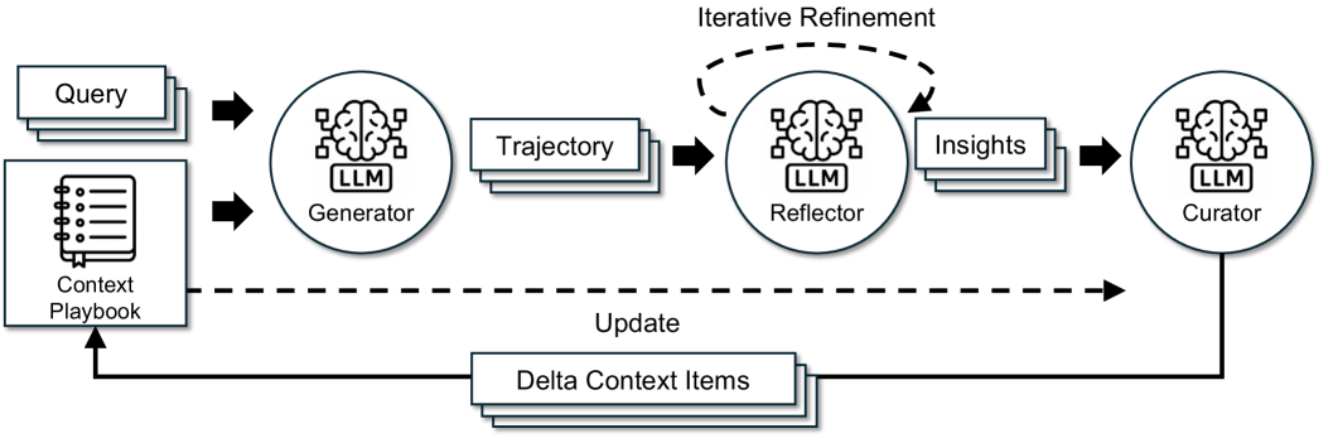



*図3: ACEフレームワークの三層構造とワークフロー*

**1. Generator（生成器）⚙️**

**役割**: 新しいクエリに対して推論軌跡を生成し、効果的な戦略と反復的な落とし穴の両方を明らかにします。

**機能**:

- プレイブックに蓄積された学習パターンを使用してタスクを実行
- 解決時にどの「弾(bullet)」が有用または誤解を招いたかをハイライト
- Reflectorへのフィードバックを提供

**2. Reflector（反省器）🔍**

**役割**: 推論軌跡を批評して具体的な洞察を抽出し、複数の反復にわたってそれらを精緻化します。

**機能**:

- 成功と失敗の実行結果を分析
- 複数ラウンド（最大5回）の反復改善が可能
- 評価と洞察抽出をキュレーション責務から独立して実行
- 構造化されたフィードバックを生成

**重要性**: アブレーション研究により、Reflectorの反復改善が段階的な性能向上に不可欠であることが実証されています（Reflectorなし: +12.7% → あり: +17.0%）。

**3. Curator（キュレーター）📝**

**役割**: レッスンをコンパクトなデルタエントリに統合し、軽量で非LLMロジックによって既存のコンテキストにマージします。

**機能**:

- Reflectorからの洞察を「構造化されたデルタエントリ」として統合
- 決定論的マージアルゴリズムで既存のプレイブックを更新
- フィードバックに基づいてプレイブックを進化
- 並列マージを可能にする効率的な統合

## コンテキストアイテムの構造

従来の全文書き換えの代わりに、ACEは構造化アイテムのセットで機能します。各「弾(bullet)」（メモリエントリ）は2つの成分を含みます：

**メタデータ**

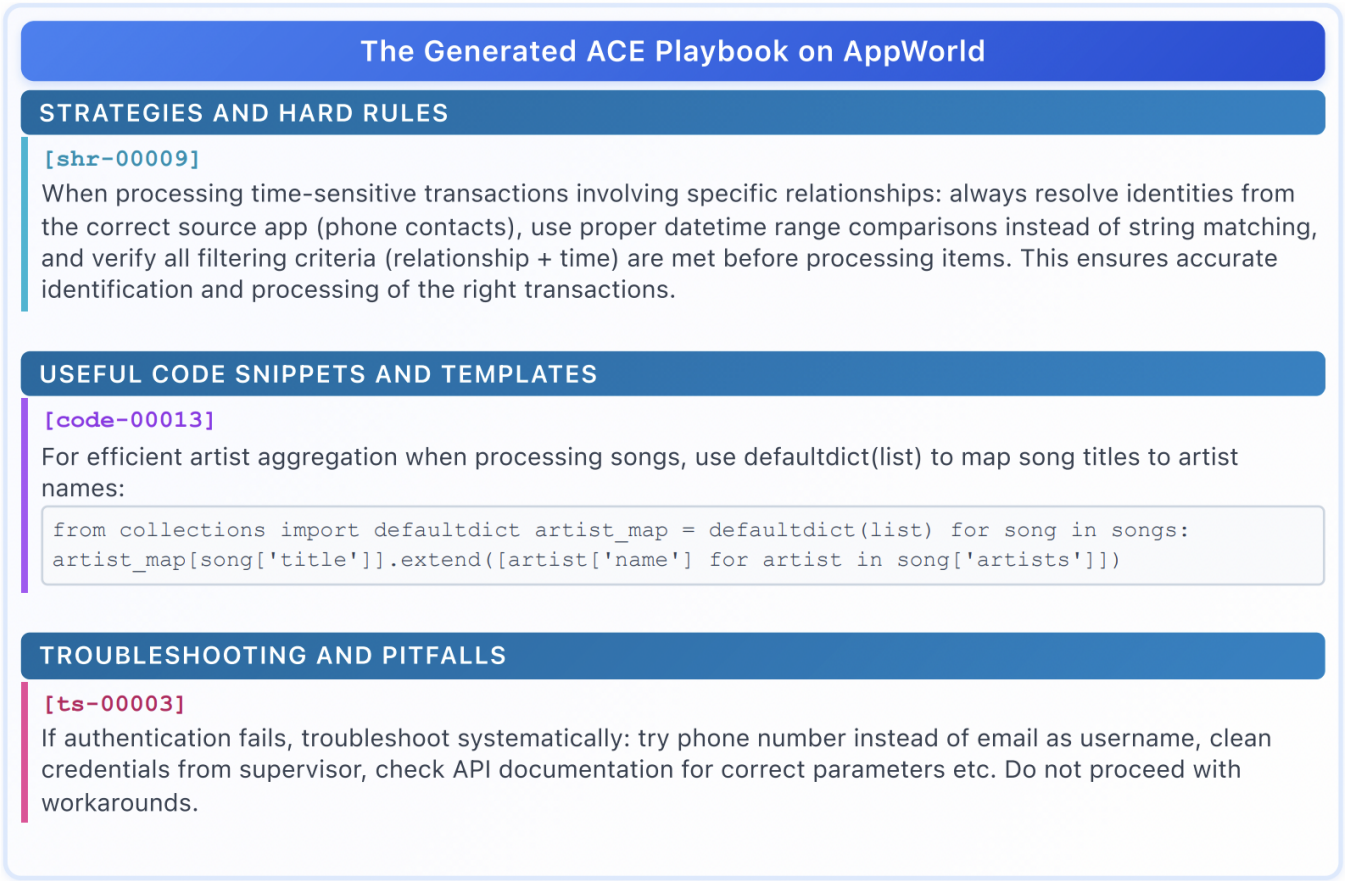
- **一意識別子(ID)**: 各エントリを区別
- **有用性カウンター**: 有用または有害としてマークされた頻度を追跡

**コンテンツ**

- 再利用可能な戦略
- ドメイン固有の概念
- 一般的な失敗モード
- ツールと即座に使用可能なコード

この構造により以下が可能になります：

1. **局所化**: 関連するアイテムのみが更新される
2. **細粒度検索**: 生成器が最も関連性のある知識に焦点を当てる
3. **段階的適応**: 推論中の効率的なマージ、削減、重複排除




*ACE生成コンテキストの例: 詳細な戦略、ツール、再利用可能なコードを含む包括的なプレイブック*

## インクリメンタル・デルタ更新メカニズム

ACEの核心的イノベーションは「構造化された増分更新」です。モノリシック書き直しを避けることで、計算効率を大幅に改善しました。

**特徴**:

- 小さなユニット（戦略、概念、失敗モード）をキャプチャ
- 複数の差分を並列的にマージ可能
- 規模での適応をバッチ処理可能
- 決定論的統合により一貫性を保証

**効果**:

- 適応レイテンシ82.3%削減（GEPA比）
- トークンコスト83.6%削減（Dynamic Cheatsheet比）

### 成長と精緻化（Grow-and-Refine）メカニズム

コンテキストの拡張と関連性維持のバランスを取るメカニズムです。

**成長 (Grow)**

- 新しい識別子を持つ弾が付加される
- ドメイン固有の新しい戦略や概念を蓄積

**精緻化 (Refine)**

- 既存の弾を現地で更新（例: カウンターを増加）
- セマンティック埋め込みを比較して重複を削除
- 有用性カウンターに基づいて優先順位付け

**実行タイミング**:

- **プロアクティブ**: 各デルタ後に即座に実行
- **遅延**: 一定の蓄積後にバッチ処理

これにより、コンテキストは詳細な知識を保持しながらスケーラブルに成長します。

### 重複排除とセマンティック埋め込み

**手法**: セマンティック埋め込みを介して弾を比較することで冗長性を削除します。

**利点**:

- 類似した戦略の重複を防ぐ
- コンテキストの情報密度を最大化
- 推論時の関連性を向上

---

## 実験結果

### 実験設定

**基本モデル**: DeepSeek-V3.1（オープンソース、非思考モード）

**ACE設定**:

- バッチサイズ: 1
- Reflector精緻化ラウンド最大: 5
- オフライン適応エポック数最大: 5

ベンチマーク

**1. AppWorld（エージェントタスク）**

**概要**: 自律エージェントタスクのスイート。API理解、コード生成、環境相互作用を含むマルチターン推論とツール使用を要求。

**評価指標**:

- Task Goal Completion (TGC)
- Scenario Goal Completion (SGC)

**結果**:

| 方法 | 平均精度 | 改善率 |
|---|---|---|
| ベースライン (ReAct) | 42.4% | - |
| ReAct + ICL | 46.0% | +3.6% |
| ReAct + GEPA | 46.4% | +4.0% |
| ReAct + ACE（オフライン） | **59.4%** | **+17.0%** |
| ReAct + ACE（オンライン） | **59.5%** | **+17.1%** |
| Dynamic Cheatsheet | 51.9% | +9.5% |

**重要な洞察**:

- オンライン適応で、テストチャレンジ分割でIBM CUGA（GPT-4.1ベース、60.3%）に匹敵
- より小規模なオープンソースモデル（DeepSeek-V3.1）を使用しながら、プロダクション級エージェントに匹敵する結果

**2. 金融分析ベンチマーク**

**FiNER（Financial Named Entity Recognition）**:

- 139種類の細粒度エンティティタイプを持つ金融文書タグ付けタスク
- XBRL標準に基づく金融情報抽出

**Formula（Financial Reasoning）**:

- XBRL帳票から値を抽出し、財務クエリに答えるための数値推論タスク

**結果**:

| タスク | ベースライン | GEPA | ACE | 改善率 |
|---|---|---|---|---|
| FiNER | 70.7% | 73.5% | **78.3%** | **+7.6%** |

| タスク | ベースライン | GEPA | ACE | 改善率 |
|---|---|---|---|---|
| Formula | 67.5% | 71.5% | **85.5%** | **+18.0%** |
| 平均 | 69.1% | 72.5% | **81.9%** | **+12.8%** |

**特筆すべき点**:

- Formulaタスクで特に大きな改善（+18.0%）
- 数値推論を伴う複雑なタスクでACEの強みが発揮される

### ベースライン手法との比較

| 手法 | 特徴 | 主な課題 |
|---|---|---|
| **ICL** | 訓練サンプルを入力プロンプトに含める少数ショット学習 | スケーラビリティの制限 |
| **MIPROv2** | ベイズ最適化でシステム指示と実例を最適化 | 簡潔性バイアス |
| **GEPA** | Reflective prompt evolutionで実行トレースから学習 | コンテキスト崩壊 |
| **Dynamic Cheatsheet** | テスト時学習で適応的メモリを累積 | 高いトークンコスト |

ACEはこれらすべての手法を一貫して上回り、特にGEPAと比較して平均13.0ポイント、Dynamic Cheatsheetと比較して7.6ポイントの優位性を示しています。

### 適応パラダイムの比較

| 適応方式 | 説明 | 教師ラベル | ACE性能 |
|---|---|---|---|
| **オフライン適応** | 訓練分割で最適化、テスト分割で評価（pass@1） | 必要 | +17.0% |
| **オンライン適応** | テスト分割上で順次実行、各サンプル後に文脈更新 | 不要（実行FB利用） | +17.1% |

**重要な発見**: オンライン適応でもオフライン適応と同等の性能を達成し、ラベルなしでの自己改善が可能であることを実証。

### コスト・速度分析

**オフライン適応（AppWorld）**

| メトリック | 削減率 | 比較対象 |
|---|---|---|
| 適応レイテンシ | **82.3%** | GEPA |
| ロールアウト数 | **75.1%** | GEPA |

**オンライン適応（FiNER）**

| メトリック | 削減率 | 比較対象 |
|---|---|---|
| レイテンシ | **91.5%** | Dynamic Cheatsheet |

| メトリック | 削減率 | 比較対象 |
|---|---|---|
| トークンコスト | **83.6%** | Dynamic Cheatsheet |

**効率化の根拠**:

- インクリメンタルなデルタ更新
- 非LLMベースのマージロジック
- KVキャッシュ再利用による長文脈推論の効率化

**重要な洞察**: 長いコンテキスト ≠ 高い推論コスト。KVキャッシュ再利用により、長コンテキストの推論効率が向上。

### アブレーション研究

主要コンポーネントの貢献度を定量化：

| 構成 | 平均精度 | 改善率 |
|---|---|---|
| ベースライン | 42.4% | - |
| ACE（Reflectorなし） | 55.1% | +12.7% |
| ACE（多エポックなし） | 56.8% | +14.4% |
| **完全なACE** | **59.4%** | **+17.0%** |

**主要な発見**:

1. **Reflectorの反復改善は不可欠**: Reflectorなしでは+4.3ポイントの性能低下
2. **マルチエポック適応は継続的な強化を実現**: 単一エポックでは+2.6ポイントの性能低下
3. **各要素の段階的な貢献**: すべてのコンポーネントが独立して性能向上に寄与

---

## その他の重要なポイント

### 制限事項

**1. Reflectorの品質依存性**

- Reflectorが意味のある洞察を抽出できない場合、「コンテキスト汚染」が発生する可能性
- 低品質のフィードバックは性能低下につながる

**2. タスク依存性**

- すべてのアプリケーションが詳細なコンテキストを必要とするわけではない
- 簡潔な指示に適したタスク（HotPotQA、Game of 24など）ではメリットが限定的

**3. フィードバック信号の必須性**

- グラウンドトゥルースラベルまたは実行結果が必要
- 信頼できるフィードバックがない場合、性能が低下する可能性

## 今後の方向性と応用可能性

**1. 継続学習への応用**

- コンテキスト適応は従来のモデル微調整より安価
- 人間が解釈可能なため、選択的な「アンラーニング」が実現可能
- プライバシーや法的制約への対応が容易

**2. 長文コンテキスト最適化**

- KVキャッシュ再利用、圧縮、オフロード技術との統合
- 長いコンテキストの摂取費用が継続的に低下する見通し

**3. マルチモーダルエージェントへの拡張**

- 視覚や音声フィードバックを含むエージェントタスクへの応用
- クロスモーダルな戦略の蓄積と再利用

**4. 産業応用**

- カスタマーサポートの自動化
- 金融分析とコンプライアンス
- ソフトウェア開発支援
- 医療診断支援

## AppWorldリーダーボード上の位置づけ

**2025年9月20日時点のリーダーボード**:

| 順位 | システム | モデル | 平均精度 |
|---|---|---|---|
| 1位 | IBM CUGA | GPT-4.1 | 60.3% |
| - | **ReAct + ACE** | DeepSeek-V3.1 | **59.4%** |

**重要な成果**:

- オープンソースモデルで最先端の商用システムに匹敵する性能
- テスト難問分割では**IBM CUGAを8.4%上回る**パフォーマンス
- より小規模なモデルでコスト効率の高い代替案を提供

## 実装と再現性

論文の公式コードは執筆時点で未公開ですが、コミュニティ主導の実装が複数存在します：

**1. OpenCE (sci-m-wang/OpenCE)**

**URL**: https://github.com/sci-m-wang/OpenCE

**特徴**:

- ACE再現を進化させた「プラガブルなメタフレームワーク」
- 五大ピラーアーキテクチャ: Acquisition, Processing, Construction, Evaluation, Evolution
- OpenAI、ローカルTransformers、RWKV対応

**インストール**:

```
uv sync
uv run pytest
```

**ミニマル使用例**:

```
from opence.methods import ACEClosedLoopMethod
method = ACEClosedLoopMethod(...)
orchestrator = method.build().orchestrator
```

### 2. Agentic Context Engine (kayba-ai/agentic-context-engine)

**URL**: https://github.com/kayba-ai/agentic-context-engine

**特徴**:

- 100以上のLLMプロバイダ対応（OpenAI、Claude、Geminiなど）
- Opik統合による本番環境監視
- ベンチマークで20〜35%の改善を報告

**インストール**:

```
pip install ace-framework
export OPENAI_API_KEY="your-api-key"
```

**最小実装例**:

```
from ace import LiteLLMClient, Generator, Playbook

llm = LiteLLMClient(model="gpt-4o-mini")
generator = Generator(llm)
result = generator.generate(question="What is 2+2?")
print(f"Answer: {result.final_answer}")
```

**学習機能付き実装**:

```
from ace import OfflineAdapter, Reflector, Curator
```

```
adapter = OfflineAdapter(
    playbook=playbook,
    generator=generator,
    reflector=Reflector(llm),
    curator=Curator(llm)
)
results = adapter.run(samples, SimpleEnvironment(), epochs=1)
```

**3. ACE Playbook (mmprotest/ace-playbook)**

**URL**: https://github.com/mmprotest/ace-playbook

**特徴**:

- Generator、Reflector、Curatorの三役割ループ実装
- 構造化されたプレイブックの進化を実現
- 戦略、ルール、落とし穴、ツールを記述

### プロンプトテンプレート

論文の付録Dでは14個のプロンプト例が提示されています。主要な3役割のテンプレート構造：

**Generator用プロンプト**

- 推論軌跡生成
- プレイブックからの関連戦略の活用
- 有用/有害な弾のハイライト

**Reflector用プロンプト**

- 実行トレース分析
- 成功/失敗パターンの識別
- 反復的な洞察の精緻化（最大5ラウンド）
- 構造化フィードバックの生成

**Curator用プロンプト**

- コンパクトなデルタエントリへの統合
- メタデータの付与（ID、カウンター）
- 決定論的マージのための構造化

各テンプレートは「実行トレース分析」と「構造化フィードバック」に焦点を当て、再利用可能な戦略抽出を促進する設計となっています。

---

## 論文とコードの対応関係

### コアコンポーネントの実装

**1. Playbook（プレイブック）**

**論文での説明**: 構造化された弾のセットとして、戦略・概念・失敗モードを蓄積

**実装（kayba-ai）**:

```
from ace import Playbook

playbook = Playbook()
# 新しい戦略を追加
playbook.add_strategy(content="API呼び出しの前にパラメータを検証する")
# 失敗パターンを記録
playbook.add_pitfall(content="タイムアウトエラーはリトライで解決可能")
```

**2. Generator（生成器）**

**論文での説明**: プレイブックを使用してタスクを実行し、有用/有害な弾を識別

**実装（kayba-ai）**:

```
from ace import Generator, LiteLLMClient

llm = LiteLLMClient(model="gpt-4o-mini")
generator = Generator(llm, playbook=playbook)

result = generator.generate(
    question="ユーザー情報を取得するAPIを呼び出してください",
    environment=api_environment
)
# result.useful_bullets と result.harmful_bullets を取得
```

**3. Reflector（反省器）**

**論文での説明**: 複数ラウンドの反復改善で洞察を抽出

**実装（kayba-ai）**:

```
from ace import Reflector

reflector = Reflector(llm, max_rounds=5)
insights = reflector.reflect(
    execution_trace=result.trace,
    success=result.success
)
# 精緻化された洞察を返す
```

**4. Curator（キュレーター）**

**論文での説明**: デルタエントリとして洞察を統合し、プレイブックを更新

**実装（kayba-ai）**:

```
from ace import Curator

curator = Curator(llm)
delta = curator.curate(insights)
playbook.merge(delta)  # 非LLMロジックで決定論的にマージ
```

### フルワークフローの実装

**論文での説明**: Generator → Reflector → Curator のループで適応

**実装（kayba-ai）**:

```
from ace import OfflineAdapter, SimpleEnvironment

# 環境とサンプルの準備
environment = SimpleEnvironment()
training_samples = [...]

# オフライン適応の実行
adapter = OfflineAdapter(
    playbook=playbook,
    generator=generator,
    reflector=reflector,
    curator=curator
)

results = adapter.run(
    samples=training_samples,
    environment=environment,
    epochs=5  # 論文と同じ設定
)

# 適応後のプレイブックを保存
playbook.save("adapted_playbook.json")
```

### デルタ更新メカニズム

**論文での説明**: 構造化された増分更新により、全体書き換えを回避

**実装の特徴**:

- 各弾に一意のIDとメタデータを付与
- セマンティック埋め込みで重複を検出
- カウンターベースで優先順位付け

**疑似実装**:

```
# デルタの生成
delta = {
    "new_bullets": [
        {"id": "strategy_001", "content": "...", "type": "strategy"},
        {"id": "pitfall_042", "content": "...", "type": "pitfall"}
    ],
    "update_bullets": [
        {"id": "strategy_005", "counter_increment": 1}
    ],
    "remove_bullets": ["strategy_003"]  # 重複として削除
}

# プレイブックへのマージ（決定論的、非LLM）
playbook.merge(delta)
```

---

## 結論

ACEは、コンテキストを「詳細で進化するプレイブック」として扱うことで、従来のプロンプト最適化手法が抱える簡潔性バイアスとコンテキスト崩壊の課題を克服しました。三層アーキテクチャ（Generator、Reflector、Curator）と構造化された段階的更新により、以下を実現しています：

1. **高い性能向上**: エージェントタスクで+17.0%、金融分析で+12.8%の改善
2. **優れた効率性**: 適応レイテンシ86.9%削減、トークンコスト83.6%削減
3. **ラベル不要の自己改善**: 実行フィードバックのみで動作
4. **スケーラビリティ**: 詳細な知識を保持しながら効率的に成長

ACEは、ファインチューニングに代わる実用的な代替手段として、自己改善型AIシステムの新しいパラダイムを提示しています。継続学習、プライバシー対応、長文脈最適化など、今後の研究の方向性も明確に示されており、実世界のアプリケーションへの展開が期待されます。

特に、オープンソースモデル（DeepSeek-V3.1）で最先端の商用システム（IBM CUGA with GPT-4.1）に匹敵する性能を達成した点は、コスト効率の高いAIシステム構築の可能性を示しています。コミュニティ主導の実装も活発に行われており、ACEフレームワークの実用化が加速すると予想されます。

---

## 参考文献


- **arXiv論文**: https://arxiv.org/abs/2510.04618
- **HTML版**: https://arxiv.org/html/2510.04618
- **OpenCE実装**: https://github.com/sci-m-wang/OpenCE
- **Agentic Context Engine実装**: https://github.com/kayba-ai/agentic-context-engine
- **ACE Playbook実装**: https://github.com/mmprotest/ace-playbook