Claude Codeで挑むOSSコントリビュート

LiteLLM Token処理バグ修正の実践例

LLMが得意な0からのコード生成ではなく、既存の大規模コードベースの理解と改修にClaude Codeを活用した事例紹介

kmuto

プロジェクトの出発点

💼 実務でのFriction Log

- **きっかけ**: 仕事でのトークン数不整合に遭遇
- **発見**: GPT-4oでencode/decodeとtoken_counterが異なる結果
- **疑問**: なぜ同じライブラリで結果が違うのか？

実際に遭遇した問題

```
from litellm import encode
from litellm.token_counter import token_counter

sample_text = "こんにちは、これが私の入力文字列です！私の名前はイシャーンCTOです"
encode_count = len(encode(model='gpt-4o', text=sample_text))     # 25 tokens ❌
counter_count = token_counter(model='gpt-4o', text=sample_text)  # 21 tokens ✅
```

🔍 Claude Codeでの調査開始

Claude Codeとの協調調査

> 「以下のコードを実行した時に、どのような関数が呼ばれるかを順番に教えてください」

```
sample_text = "Hellö World, this is my input string!"
openai_tokens = encode(model="gpt-3.5-turbo", text=sample_text)
```

📊 Claude Codeが明らかにした呼び出しフロー

1. `encode(model="gpt-3.5-turbo", text="Hellö World...")`
2. `_select_tokenizer(model="gpt-3.5-turbo")`
3. `_return_openai_tokenizer()`
4. `tiktoken cl100k_base encoding.encode()`

🔍 Claude Codeが発見した問題

> このフローの中で、何かおかしい部分はありますか？

AI の洞察

> **"設計上の問題: encode/decode は適切なモデル固有処理をしていない"**

具体的な問題点

- `_return_openai_tokenizer()` は常に `cl100k_base` を返す
- モデル固有処理（GPT-4o → `o200k_base`）が未実装
- `token_counter` でのみ正しい処理が実装済み

🎯 解決への思考プロセス

1. 解決方針の策定

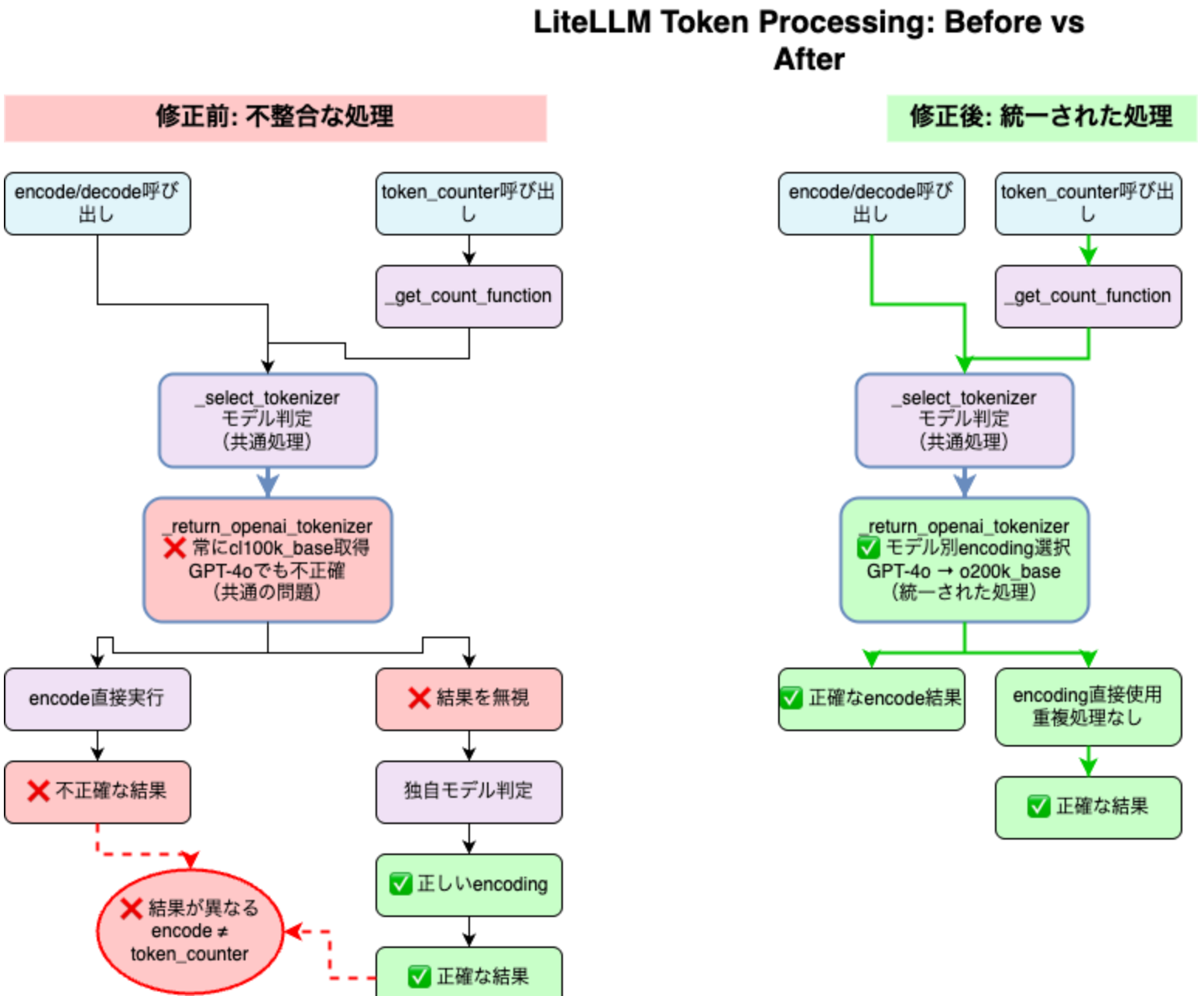
> **人間**: "/plan encode/decode と token_counterの実装の差異をなくすことを検討しています"
> **Claude Code**: 詳細な実装計画を提案し、実行

2. 依存関係の修正

> **Claude Code**: 依存関係が逆になるコードを追加
> **人間**: 依存関係の整理を指示

3. リファクタリングの実施

> **Claude Code**: 新しいファイルにロジックを追加
> **人間**: 既存ファイルとの関連性を指摘し、コードの統合を指示


LiteLLM Token Processing: Before vs After
修正前: 不整合な処理
encode/decode呼び出し
token_counter呼び出し
_get_count_function
_select_tokenizer
モデル判定
(共通処理)
_return_openai_tokenizer
❌ 常にcl100k_base取得
GPT-4oでも不正確
(共通の問題)
encode直接実行
❌ 結果を無視
❌ 不正確な結果
独自モデル判定
❌ 結果が異なる
encode ≠
token_counter
✅ 正しいencoding
✅ 正確な結果
修正後: 統一された処理
encode/decode呼び出し
token_counter呼び出し
_get_count_function
_select_tokenizer
モデル判定
(共通処理)
_return_openai_tokenizer
✅ モデル別encoding選択
GPT-4o → o200k_base
(統一された処理)
✅ 正確なencode結果
encoding直接使用
重複処理なし
✅ 正確な結果

⚠️ Claude Code活用の課題と対策

1. Pythonコード実行の危険性

簡単な検証をClaude Codeが行う際に、危険なコマンドを実行する可能性がある

```
# 危険！システムエラーを引き起こす可能性
Bash(python3 -c "import os; os.remove('important_file')")
```

対策: コード実行前に内容を人間が確認、Dev Containerで安全に実行

2. 依存関係の複雑さ

- Claude Codeが逆方向のimportを提案することがある
- **対策**: 既存コードベースの構造を理解してから指示

3. 完全自動化の限界

- Claude Codeは既存のコードを再利用することが苦手

Claude CodeでOSS貢献のベストプラクティス

📋 効果的なアプローチ

1. **明確な質問設計**: 「答えが一意に定まる質問」を心がける
    - 例: 「このコードを実行した時に、どのような関数が呼ばれるか？」
2. **人間とAIの役割分担**: AIに任せる部分と人間が確認すべき部分を明確化
    - 例: AIにコードの追加を任せ、人間がリファクタリングを指示（改善の余地はある）
3. **安全な実行環境**: Dev Containerなどでコードを安全に実行

✅ 本プロジェクトの成果

- **実際のPR**: https://github.com/BerriAI/litellm/pull/13907
- **修正内容**: GPT-4o Token処理の統一化完了
- **費用**: $11.87で問題発見

💡 リアルな現実

ちなみに、PRのCIは落ちています 😅

Flakyなテストが存在 → **新たな課題発見！**

→ 新たな課題へ

ありがとうございました

※ちなみに、このスライドもClaude Code&Marpで作成しました
※図はdrawio形式で出力させたものを人手で微調整しています