

Knowledge Graph Based Repository-Level Code Generation

Mihir Athale, Vishal Vaddina

ICSE 2025

背景：LLMによるコード生成の進化

- LLMの登場により、自然言語から機能的なコードを生成する能力が飛躍的に向上
- 大量のコードとドキュメントで学習されたこれらのモデルは、HumanEvalなどのベンチマークで前例のない性能を達成
- ソフトウェア開発サイクルの加速や、開発者の認知的負荷の軽減が期待

課題：開発現場におけるLLMの限界

LLMは単一のコードを生成するには優秀だが、進化し続ける実際のコードベースにおいては、文脈の正確性を維持するのに苦労

プロジェクト固有のアーキテクチャパターンや内部の依存関係を理解せずに、コードを生成しがち

既存の機能を重複させる冗長なコードを生成したり、プロジェクトのコーディングスタイルと矛盾するコードを生成することがある

既存アプローチの限界

従来のRAG

従来の検索システムは「キーワードの一致（文字列検索）」や「意味の近さ（単純なベクトル検索）」に依存していて、コードが複数のファイルにまたがって複雑に呼び出し合っている場合、本当に必要なファイルを正しく集めてくることができない

エージェント型ワークフロー

LLMエージェントによる反復的な修正アプローチも提案されているが、余分な計算コストの発生や非生産的なループに陥る課題がある

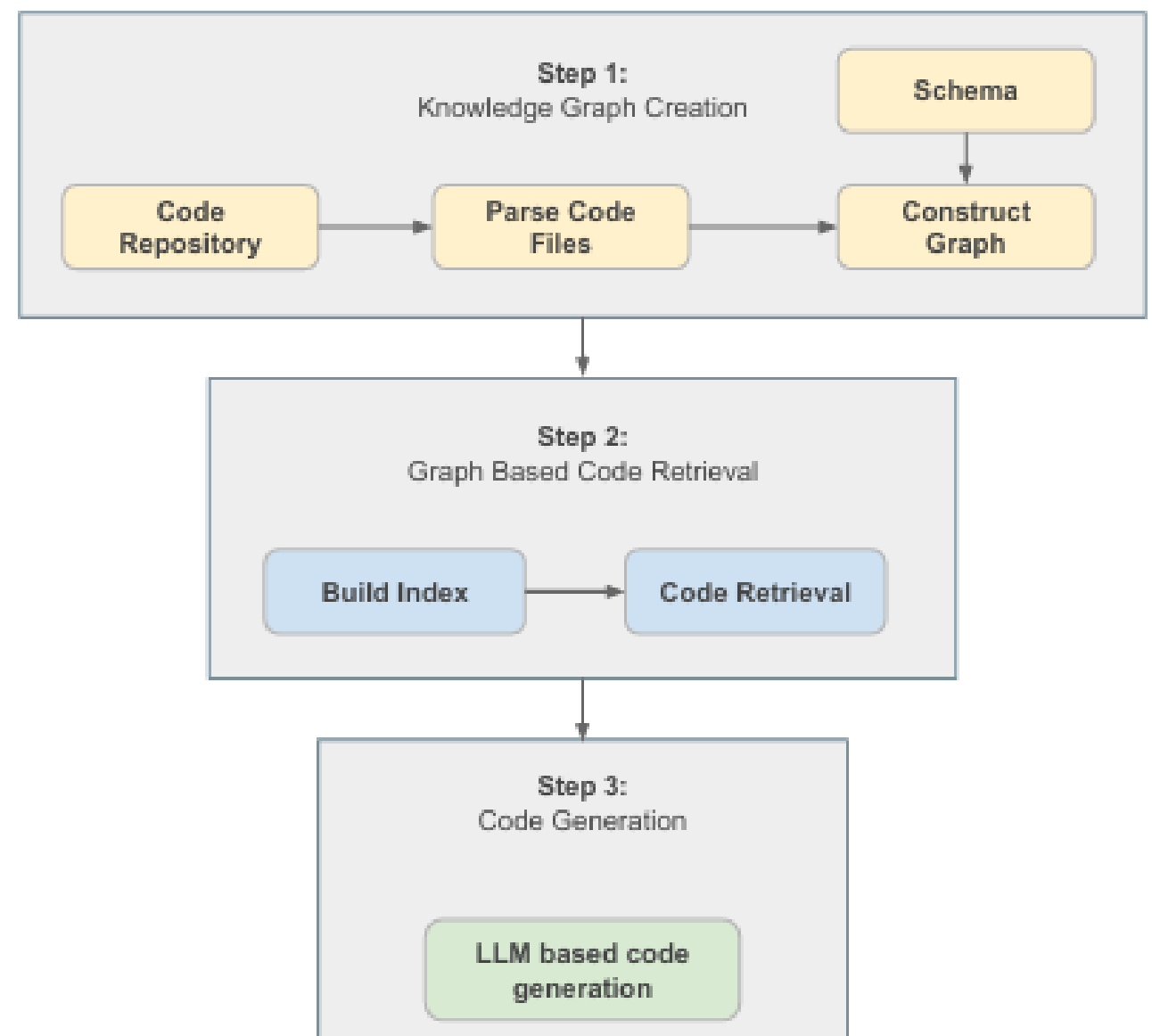
提案手法

ナレッジグラフベースのコード生成

コードリポジトリを「グラフ」として表現し、構造的および関係的な情報を捉えることで、コンテキストを認識したコード生成を強化

構成ステップ

- ナレッジグラフの構築
- グラフベースのコード検索
- LLMを用いたコード生成



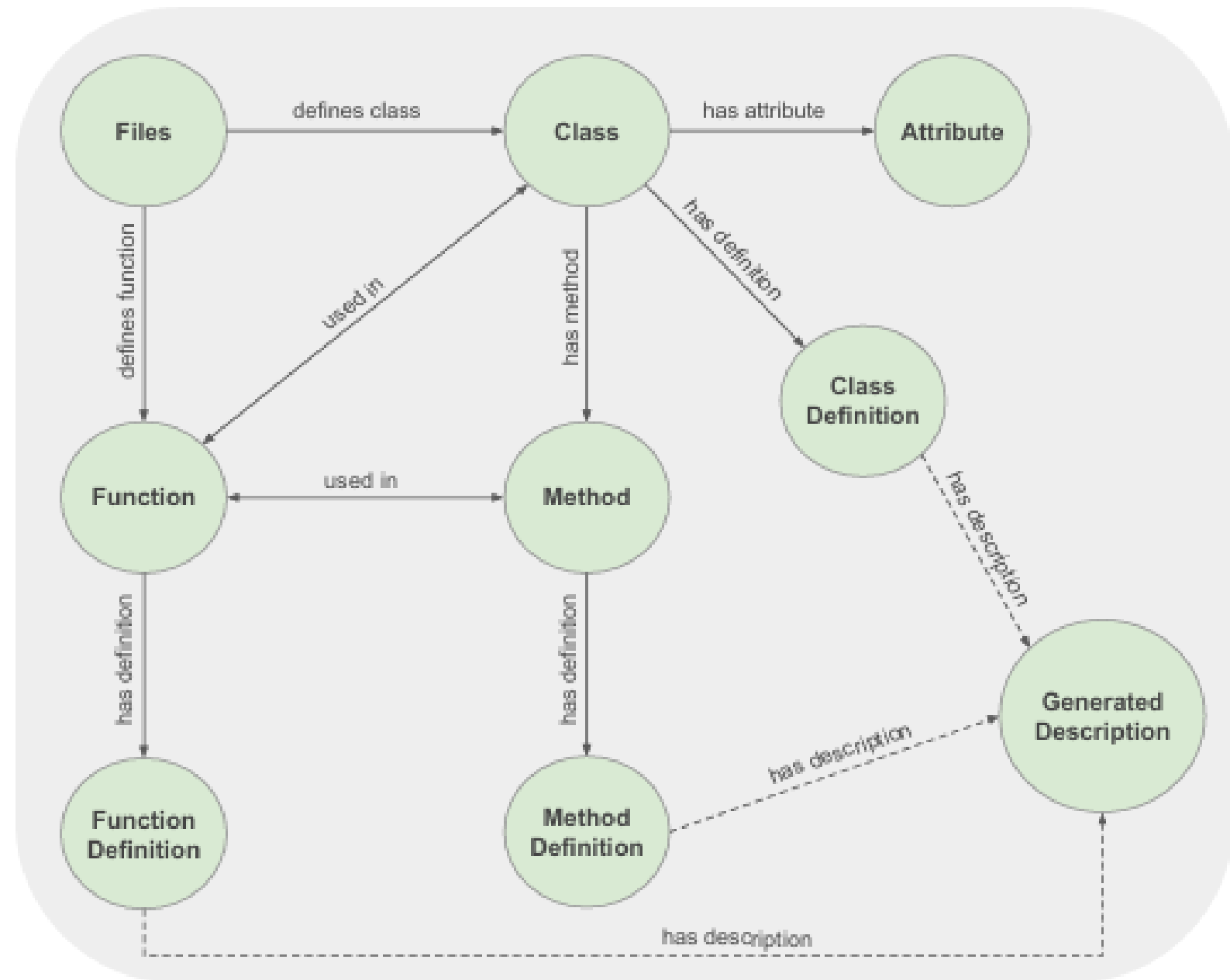
STEP1:ナレッジグラフの構築

- | Pythonのコードファイルを抽象構文木を使用して解析
- | コードベースから中核となる要素（クラス、メソッド、関数、属性など）を抽出
- | コードのドキュメントやコメントも抽出
- | リポジトリ内のコードの機能的な意味を捉えるため、LLMを使用してコードスニペットの説明文を生成し追加

STEP1:ナレッジグラフスキーマ

ナレッジグラフ

ファイル、クラス、メソッドといったノードを定義し、それぞれが「定義する」「使用される」といった関係性で結ばれるモジュール形式のスキーマ



STEP1:データベースへの統合

抽出したデータとスキーマをグラフデータベースに取り込む

検索を最適化するため、2種類のインデックスを作成

- **フルテキストインデックス**: 関数名、クラス名、メソッド名などの文字列検索用
- **ベクトルインデックス**: LLMが生成した説明文などをエンコーダーでベクトル化した意味検索用

STEP2:ハイブリッドコード検索

- | ユーザーが自然言語でクエリを入力すると、まずLLMがスキーマ要素と一致する特定の「エンティティ」を抽出
- | 同時に、エンコーダーモデルを使ってユーザーのクエリをベクトル化
- | フルテキスト検索とベクトル検索を組み合わせで実行し、最初の関連ノードを取得

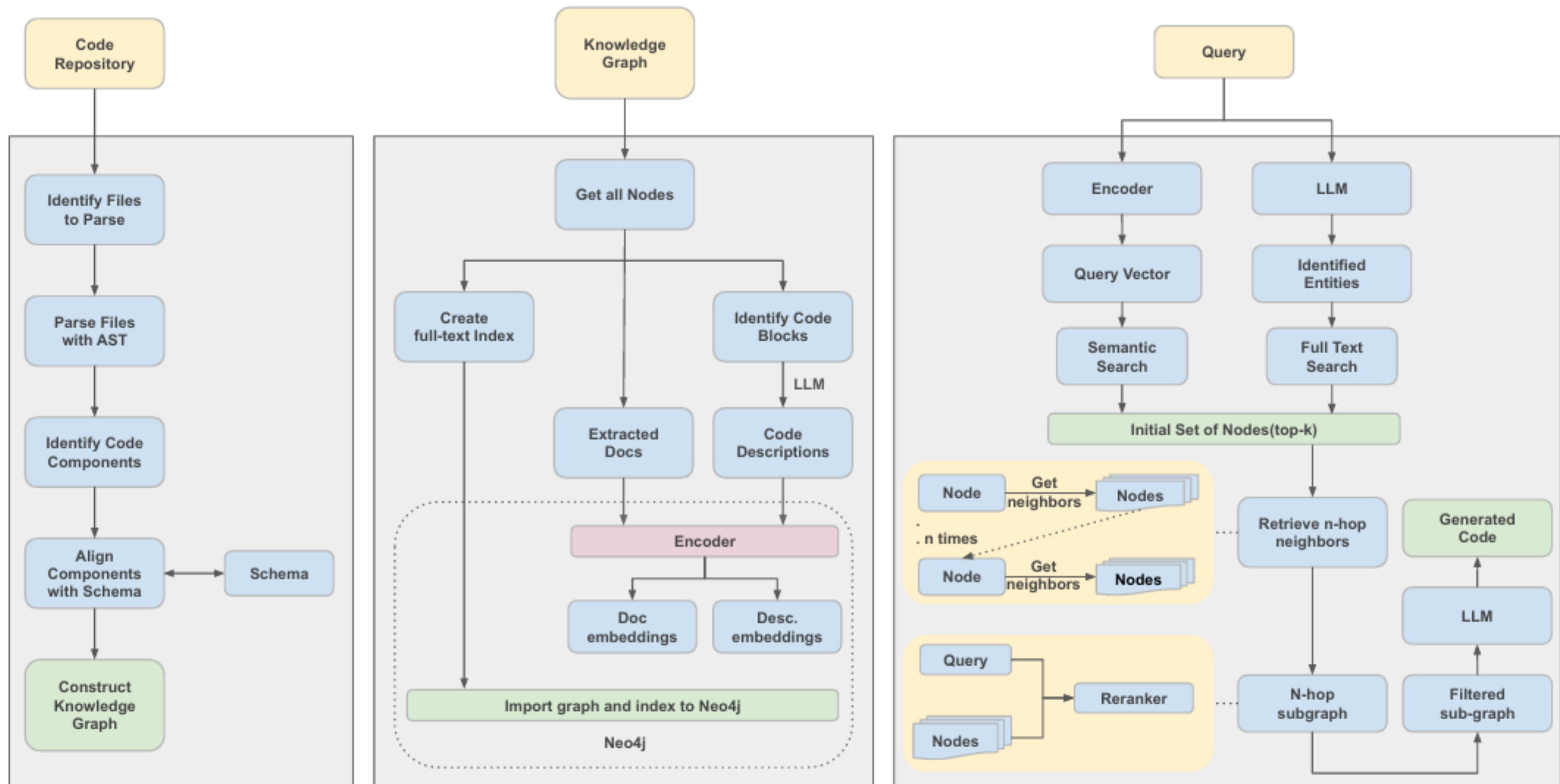
STEP2:検索結果の拡張・剪定

- 初期検索で見つけたノードから、グラフをn階層辿ることで、関連する関数やクラスなどの周辺コンテキストを網羅した「n-hopサブグラフ」を取得
- 最後にベクトル類似度を使って関連性の低いノードを削り、LLMに渡すノイズを減らす

STEP3:LLMによるコード生成

- | 洗練されたサブグラフとユーザーの初期クエリを、LLMにコンテキストとして渡す
- | サブグラフには単なるコードの断片だけでなく、アーキテクチャや依存関係の情報も関係性としてカプセル化されている
- | 得られた情報を基盤とすることで、LLMはリポジトリのフレームワークに一致したコードを生成できる

アプローチの全体像



評価設定

リポジトリレベルのコード生成機能を評価するために設計されたベンチマーク「EvoCodeBench」を使用

オープンソースの25リポジトリから抽出された275個のPythonコード生成タスクが対象

生成されたコードがテストケースを初回で通過するかを測定する `pass@1` メトリクスを利用

サブグラフは2-hopで抽出

評価結果

提案手法は、Claude 3.5 Sonnetを使用した場合に **36.36%** の最高 pass@1 スコアを達成

Category	Method	LLM Model	pass@1 Score
Proposed Approach	Graph-based Retrieval	GPT-4	32.00%
		GPT-4o	33.45%
		Claude 3.5 Sonnet	36.36%
Baseline (EvoCodeBench)	Local File (Infilling)	GPT-4	20.73%
		GPT-3.5	17.82%
	Local File (Completion)	GPT-4	17.45%
		GPT-3.5	15.64%
	Without Context	GPT-4	7.27%
		GPT-3.5	6.55%
CodeXGraph	Graph-based (212/275 Samples)	Qwen2	19.34%
		DS-Coder	25.47%
		GPT-4o	36.02%

結論

- グラフベースのアプローチは、コードの依存関係や使用状況のコンテキストを提供し、LLMがより正確なコードを生成するのを助ける
- フルテキスト検索とベクトル検索を組み合わせたハイブリッドシステムが、コードベースから関連スニペットを抽出する上で効果的である
- 生成後の手動修正の必要性を減らし、高品質なコード生成のワークフローを合理化

所感

発展としてLLMエージェントが検索の深さ（n階層）を動的に調整する自律的ワークフローの導入などが考えられる

リポジトリ全体を正確な関係性グラフとして保持するアプローチはAIが生成したコードが設計要件等に違反していないか、AI自身が比較・検証を行う自己反省システムに応用できる可能性を感じた

グラフ自体を人間が操作可能なキャンバスとして表出させ、人間の操作を中間表現にしてコードの更新トリガーとする双方向型開発環境の可能性