ChatGPT

- **Sec.1（導入部）** “fill-in-the-middle performs poorly, programs may not fit into the context window, generated types may not type check” [1] – LLM 単体では、FIM モデルが短い型注釈を生成するのに適さず、プログラム全体がコンテキストウィンドウに収まらない（大規模コンテキスト問題）、また生成した型が型チェックに通らない等の課題があると指摘している。これらにより LLM 単体だけでの型予測は困難である。 – (失敗モード (1),(3),(4))
- **Sec.1（導入部）** “we recursively decompose a program into smaller contexts, then run inference on the respective subprograms” [2] – OpenTau は大規模プログラムを小さなコードブロックに**分解**して LLM で型推論を行う。この分解により、コンテキスト長制限（失敗モード1）を回避しつつ型推論を可能にする。 – (失敗モード (1))
- **Sec.9（結論）** “OpenTau, a search-based approach for type prediction that leverages large language models” [3] – OpenTau は LLM を用いながら型予測の**探索**を行うアプローチであり、分解した各部分の候補解を組み合わせて評価する。このように探索を導入することで、型注釈の組合せ爆発（失敗モード2）に対応している。 – (失敗モード (2))
- **Sec.4（Ranking）** “OpenTau runs the TypeScript compiler’s type checker on each candidate and extracts the number of type errors. If there are no type errors, then the solution type checks.” [4] – 各候補プログラムに対し TypeScript コンパイラの型チェッカを実行し、型エラー数を取得している。エラーがなければその候補は型チェック済みとみなし、少ないエラーの解を優先する。型チェッカ検証を入れることで、生成型が実際に検証可能かどうかを判断し、誤った型付けを防いでいる。 – (失敗モード (3))
- **Sec.3（Program Typedness）** “we propose a typedness metric that captures type information, but is also amenable to type checking and does not require ground truth data.” [5] – オープンTauでは、新たに **typedness（型情報度）** メトリクスを導入している。これはプログラムに含まれる型情報の量・精度を表し、型チェッカで検証可能でありつつ具体性の低い型（any など）をペナルティとする。この指標により、生成プログラムの「どれだけ十分に型付けされているか」を評価する。 – (失敗モード (4))
- **Sec.6.2（実験評価）** “percent of files that type check… We emphasize that our methodology counts files that type check, which is more rigorous than prior work…” [6] – OpenTau の評価指標として「型チェックに通過したファイルの割合」を重視している。従来の部分的な型正解数と異なり、ファイル全体として型チェック可能かを評価することで、より厳密なファイル単位評価を行う設計になっている。 – (失敗モード (4))
- **Sec.1（導入部）** “entire programs are often very large and may not fit within a context window” [7] – 大規模プログラムは依然としてモデルのコンテキスト制限に収まらない場合が多いと述べている。コンテキスト長の制約（失敗モード1）は根本的な問題であり、分解手法でこれを回避しているが、完全な解決ではないことが示唆される。 – (失敗モード (1))

TypeWeaver との比較

1. **目的関数:** OpenTau は最終的に型エラーを最小化し、情報量の多い型注釈を最大化するよう（型情報度を考慮し）探索する [8] のに対し、TypeWeaver は「精度」ではなく型チェック通過率に着目し [9]、完全な型チェック可能コードを出力することを評価目標としている。
2. **評価:** TypeWeaver はパッケージ単位とファイル単位の型チェック成功率（例：21%のパッケージ、69%のファイルが通過）を報告する [10] のに対し、OpenTau は主にファイル単位の成功率に注目し（47.4%のファイルが通過 [11]）、加えて型情報度（typedness）スコアで質を評価する点が異なる。
3. **介入点:** TypeWeaver は外部モデルの出力を受け取り、依存関係の型取得やJS→TS変換、予測型の挿入、非型出力の除外といった**移植パイプライン**を自動化する（例:「予測型注釈の挿入」「非型予測の拒否」[12]）。一方 OpenTau は**生成プロセス**に介入し、プログラムを木構造で分解して個々のブロックで探索的に型を推論し、型チェッカによる検証結果で解を絞り込む手法を取っている。

---

1 2 3 4 5 6 7 8 11 [2305.17145] Type Prediction With Program Decomposition and Fill-in-the-Type Training
https://ar5iv.labs.arxiv.org/html/2305.17145

9 10 12 Do Machine Learning Models Produce TypeScript Types That Type Check? (Artifact)
https://drops.dagstuhl.de/storage/05darts/darts-vol009/darts-vol009-issue002_ecoop2023/DARTS.9.2.5/DARTS.9.2.5.pdf