ChatGPT

ライブラリ型推論手法の分類

JavaScript/TypeScript のライブラリ型推論研究は、大きく「静的解析」「動的解析」「自然言語利用」「機械学習（ML/LLM）」といったアプローチに分類できる [1] [2]。例えば、**静的解析**では tsinfer/tsevolve [1] のように、ライブラリをブラウザ実行して得られるヒープスナップショットを静的解析し、`.d.ts` 定義を自動生成する手法がある。このほかに、純粋な静的解析型推論（型伝播や制約解法）もあるが、JavaScript の動的性質から完全型推論は困難である。**動的解析**では、実行時トレースやテスト実行を通じて型情報を収集する手法が考えられる（例: tsinfer の初期化フェーズではブラウザでライブラリを実行し型情報を取得 [1]）。**自然言語利用**型推論は、ソースコード中の識別子名やコメントなどから型を推定するもので、代表例に NL2Type [2] [3] がある。NL2Type は関数名・コメント等の自然言語情報を特徴量とし、JavaScript関数のシグネチャ（引数・戻り値型）を推定するモデルで、Top-1 提案の精度は84.1%（F1=81.2%）を達成している [3]。
機械学習（ML/LLM）型推論では、DeepTyper [4] [5]（RNN）や LambdaNet [6]（GNN）、TypeBert/DiverseTyper [7]（Transformer）、さらには近年の大規模言語モデル（LLM）を応用した OpenTau [8] などが知られる。これらは大量の型付きコードを学習し、コードコンテキストから型注釈を予測する。下表に各カテゴリと代表研究例をまとめる。

カテゴリ	特徴と代表研究例
静的解析	ブラウザ実行＋解析による `.d.ts` 生成（tsinfer/tsevolve [1]）など。完備型推論が難しいJSでは、静的分析の精度に限界あり。
動的解析	実行時トレースやインストルメンテーションで型情報を収集（例: tsinfer のブラウザ初期化） [1]。動的性質の把握に有効。
自然言語利用	関数名・コメントから型を予測（NL2Type [2]）。Top-1 で84.1%精度、トップ5提案なら95.5%まで精度向上 [3]。
機械学習(伝統的)	DeepTyper [4]（RNN, 文脈重視）や LambdaNet [6]（GNN, 型依存グラフ）など。コンテキスト理解により従来手法を上回る精度を示す。
機械学習(事前学習)	TypeBert [7]（BERTモデル, 事前学習）や DiverseTyper（TypeBert拡張）、CodeT5系など。大規模コーパス事前学習で高精度（TypeBertはTop-100型で89.5%） [9] を達成。ただし固定語彙や未知型への対応が課題。
機械学習(LLM)	OpenTau [8] など、LLM（GPT系・Fill-in-middle）＋分解探索で型予測。47.4%のファイル型チェック成功率を報告 [8]。ツリー分解や“typedness”評価により精度を改良。

型注釈生成の評価指標

従来の研究では、個々の型予測（各識別子に対する正解型の予測）の**Top-1精度**や**Top-K精度**が主な評価指標だった。しかし、この指標は大局的な型整合性を必ずしも反映しない [10]。例えば TypeWeaver [10] [11] の研究では、「型予測精度が高くても、生成コードがTS型チェッカーを通るとは限らない」点を指摘し、実際に最良モデルでもライブラリ全体で型検査に合格するのは21%に過ぎなかった（ファイル単位では69%） [11]。このように近年は、個別精度ではなく**型検査通過率**（tsc通過率）を重視する流れが強まっている。また、OpenTau [8] は新たに「typedness」という指標を導入し、生成プログラムの型情報の充実度を定量化している [8] [12]。実際、OpenTauは生成したTSファイルの47.4%を型チェックさせて成功させ、ファイル当たり平均3.3個の型エラーに抑えたと報告する [8]。TypeWriter（Python向け）や TypeGen など、型チェッ

カーを組み込んで注釈を検証・修正する手法も同様の発想である。今後の移行支援では、「tscで通るか」がより実用的な指標とみなされている点が重要である [10] [11]。

代表的手法の比較

下表に代表的手法の要約を示す。DeepTyper [4] はRNNベースで文脈情報を活用し、トレーニングデータ数千個の識別子に対して型予測を行った。実験ではTop-1精度は約60% [5] に達し、コンパイラ連携により95%以上精度で4,000個超の型注釈を自動追加できたという [4] [13]。LambdaNet [6] は型依存グラフとGNNを用い、既存手法をライブラリ型に関して14%絶対向上させた [6]。TypeBert [7] [9] はBERT形式の事前学習モデルで、トップ100の標準型では89.5%という高い精度を達成した [9]。ただし語彙を100種に制限し、固有のユーザー定義型予測は行えないため、新たな型表現への適用には弱点がある [7] [9]。DiverseTyper などで固有型対応を拡張している。NL2Type [2] [3] は自然言語情報に基づく手法としてトップレベルの精度を示すが、あくまで関数シグネチャに特化しており、全体の移行ツールとは位置づけが異なる。

TypeWeaver [10] [11] は「MLモデルとTypeScriptコンパイラ導入」のパイプラインツールで、DeepTyper・LambdaNet・InCoderなどを統合し実験した。上述の通り、最良でもパッケージレベルの成功率は低く、「モデルの高精度≠移行成功」であることを示した [11]。OpenTau [8] はGPT系LLMにコード分割＋探索を組み合わせ、従来モデルと比べて生成コードの型チェックスルー率を14.5%上回る47.4%に改善した [8]。FlexType [14] はエディタプラグインとして、様々な型予測モデルを組み合わせて対話的に注釈を追加する仕組みで、小規模モデルでも動作可能である。各手法の**特徴と課題**をまとめると、MLモデルは個別精度で高い結果を得ている一方で、型の網羅性や未学習型への対応、全体整合性の保証が課題である。一方、TypeWeaver/OpenTauのように「型チェッカーで通す」方向は有望だが、まだ成功率は限定的である [11] [8]。

モデル	手法/特徴	評価指標・性能	型チェック成功率
DeepTyper (2018) [4] [5]	RNNベース、文脈依存型	Top-1 精度 約60% [5]	—
LambdaNet (2020) [6]	GNN＋型依存グラフ	Top-1 精度 66.9%（標準ライブラリ型） [6]	—
TypeBert (2021) [7] [9]	事前学習型トランスフォーマ	Top-1 精度 89.5%（上位100型） [9]	—
NL2Type (2019) [2] [3]	RNN＋自然言語特徴	Top-1 精度 84.1%（型予測精度） [3]	—
TypeWeaver (2023) [11]	MLモデル＋移行ツールチェーン	—	パッケージ成功率21%、ファイル成功率69% [11]
OpenTau (2023) [8]	LLM＋ツリー探索（FIT学習）	—	ファイル成功率47.4% [8]

提案手法の位置づけ

本研究で提案する **TypeBERT + コンパイラ誘導修正** は、上記の潮流を踏まえたアプローチである。まず高精度を示すTypeBERTのような分類モデルで各識別子型を予測し（局所精度の高さ [7] [9]）、その後TypeScriptコンパイラを用いて型エラーを検出・修正する。具体的には、生成した型注釈に対し `tsc` チェックを実行し、型エラー箇所を探索的に修正する。これにより、「精度の高い予測」と「型整合性の保証」を組み合わせる点が本手法の特徴である。TypeWeaverやOpenTauは検証ループを持つが、TypeWeaverは既存

モデル出力を流用する一般手法、OpenTauは主にLLMから生成して型検証する方式である。本手法は**TypeBERTの高精度予測**と**コンパイラによるグローバル検証**を組み合わせる点で差別化される。期待される貢献としては、TypeBERT単独では達成困難な未出現型の推定支援や、ファイル全体の型チェック成功率向上が挙げられる。また、多様なプロジェクトに適用可能な手法となるよう、実装にはFlexTypeのようなプラグインフレームワークとの統合も検討する。以上のように、本提案は従来研究のモデル精度重視から**型整合性と実用性重視**へのシフトに呼応した手法であり、TypeWeaver/OpenTauの流れを受け継ぎつつ、BERTベースモデルと型検証の長所を両立するものである。

参考文献: TypeWeaver [11]、OpenTau [8]、TypeBERT/DiverseTyper [7]、DeepTyper [4] [5]、LambdaNet [6]、NL2Type [2] [3]、tsinfer/tsevolve [1] など（必要に応じて本文引用）。各手法の特徴・課題は上記分類表および比較表にまとめた。

---

[1] users-cs.au.dk
https://users-cs.au.dk/amoeller/papers/tstools/paper.pdf

[2] [3] software-lab.org
https://www.software-lab.org/publications/icse2019_NL2Type.pdf

[4] [5] [13] Deep Learning Type Inference
https://vhellendoorn.github.io/fse2018-j2t.pdf

[6] [2005.02161] LambdaNet: Probabilistic Type Inference using Graph Neural Networks
https://arxiv.org/abs/2005.02161

[7] [14] Predicting TypeScript Type Annotations and Definitions with Machine Learning
https://mhyee.com/publications/2024-phd-thesis.pdf

[8] [12] [2305.17145] Type Prediction With Program Decomposition and Fill-in-the-Type Training
https://ar5iv.org/abs/2305.17145

[9] Learning Type Annotation: Is Big Data Enough?
https://www.cs.ucdavis.edu/~devanbu/typebert_esec_fse_.pdf

[10] [11] [2302.12163] Do Machine Learning Models Produce TypeScript Types That Type Check?
https://arxiv.org/abs/2302.12163