先行研究調査レポート：“精度”ではなく“移行成功（type-check成功）”を評価する型注釈研究

作成日：2025-12-30（Asia/Tokyo）

目的：TypeScript（JS→TS）型付け・型推論研究において、個々の型の正解率（accuracy）よりも、プロジェクトが TypeScript 型検査（tsc）に通るか（type-check success）を評価軸とする先行研究を整理し、卒論の技術背景・先行研究章で引用できる形にまとめる。

要旨

近年、JavaScript コードへの型付与を機械学習で自動化する研究が増えているが、従来の評価は「型注釈を1箇所ずつ当てる」局所精度（Top-1/Top-k exact match）に偏りがちである。ところが TypeScript の型付けは、ファイル横断の依存関係・型の流れ・外部定義（.d.ts）等により“全体整合性”が要求されるため、局所精度が高くても最終成果（ビルド可能／移行可能）に直結しないことがある。 本レポートでは、評価を「tsc が通るか」に置き換える（または併用する）研究潮流を整理し、(1) TypeWeaver（ECOOP 2023）の「accuracy は誤解を生む」主張と package/file-level type-check 成功率、(2) OpenTau（arXiv 2023）の program decomposition + 検証ループによる file-level type-check 改善、(3) 型チェッカを検証器（oracle）として組み込む TypeWriter 等の近接研究、(4) ベンチマーク側で repository-level の型整合性を測る TypyBench（2025）を中心に、評価指標の設計指針をまとめる。

目次（概要）

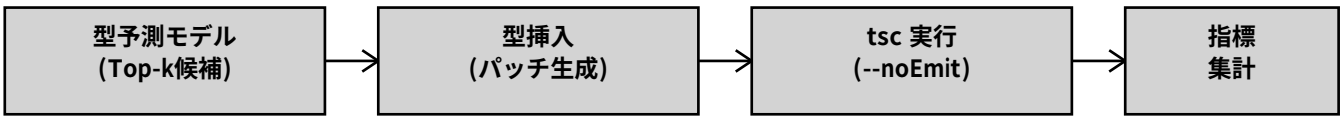
1. 背景：なぜ“ 精度 ” だけでは不十分か

TypeScript の型付け・型推論研究では、長らく「個々の型注釈が正解と一致したか」を主指標として報告する例が多かった。例えば DeepTyper や LambdaNet
など、学習モデルが出した候補と正解ラベルを照合し、Top-1/Top-k の一致率で性能を比較する。こうした指標は計測が容易で、モデル改善を追いやすい一方で、実際の移行作業における“ 成功 ”
を直接表しにくい。

JS→TS
移行の現場では、最終的に重要なのは「ユーザがビルドを壊さずに導入できるか」であり、これは TypeScript コンパイラ（tsc）による型検査（あるいはビルド）通過として観測できる。局所的に“ それっぽい型 ” が多く当たっていても、次のような要因で tsc は失敗し得る。 全体整合性：
同一識別子が複数箇所で異なる使われ方をし、適切な union / generics が必要になる。 依存関係：
外部ライブラリの型定義（.d.ts）が不足すると、上流が正しくても下流で崩れる。
構文・モジュール変換： JS→TS 変換（import/export
等）の副作用で、型以前にコンパイルが落ちる。 設定依存：
strictness（例：noImplicitAny）により「通る/落ちる」が変わる。

この問題意識を最も明確に打ち出したのが TypeWeaver である。同論文は、既存研究が報告する高い accuracy が「型移行が実用的に成功する」ことを意味しない、と主張し、問いを「生成した型が type check するか」に置き換えるべきだと論じる。さらに、package-level（パッケージ全体）と file-level（ファイル単位）で成功率が大きく異なることを実証し、移行成功の定義が多層であることを示した。

図1：type-check成功を“ 最終判定 ” にした評価パイプラインの概念図



※“ 精度 ” は局所的指標。ここでは tsc/type-check を“ 最終判定（oracle） ” として用いる。

以降では、(i) 指標の分類（何を“ 移行成功 ” とみなすか）、(ii)
その指標を採用・提案した主要論文、(iii)
卒論での引用・実装に落とし込む際の注意点、を整理する。

2. 評価指標の整理：accuracy vs type-check success

評価指標は大きく (A) 局所一致（accuracy 系）と (B) 型チェッカ通過（type-check 系）に分けられる。重要なのは、(B) が (A) を置き換えるというより、研究目的に応じて (A) の限界を補う“ 最終到達点 ”として (B) を採用する点である。TypeWeaver はこの立場を明確化し、OpenTau は探索・検証ループにより (B) を直接最適化する。

表1：代表的な評価指標の整理

カテゴリ	代表指標	測定単位	長所	限界（移行の観点）
局所一致（accuracy）	Top-1 / Top-k exact match（正解型と一致）	注釈1箇所	実装が簡単。モデル比較に向く。	全体整合性・依存関係を反映しない。
型チェッカ通過（type-check）	tsc success rate（エラー0）	file / package / repo	実用価値に直結。整合性も含む。	設定依存。エラー0以外の情報が落ちる。
エラー量ベース	tsc エラー件数 / 種別の変化	file / package	部分改善や回帰も評価可能。	エラー重み付けが必要。ノイズも増える。
タイピング強度（typedness）	any 比率	file / package	“ 通るが any だらけ ”を補正。	品質と相関が単純でない。

論点： 卒論で「tsc success / type-check率」を採用する場合、(1) 評価単位（file/package/repo）を明確にし、(2)“ エラー0 ” だけでなくエラー数・エラー種別（回帰）も併記し、(3) any などの逃げが増える副作用を補助指標で監視する、という設計が先行研究と整合的である。特に TypeWeaver は package-level と file-level のギャップを報告しており、単位の選び方が結論に影響する。

3. 主要先行研究（TypeScript中心）

3.1 TypeWeaver（ECOOP 2023）：accuracy の限界と type-check 評価の提案

TypeWeaver は「Do Machine Learning Models Produce TypeScript Types That Type Check?」として発表された。中心主張は、既存モデルが報告する高い annotation-level accuracy は“ 移行成功 ”を過大評価し得るため、研究の問いを“ type check するか ”に据えるべき、という点である。実験では DeepTyper / LambdaNet / InCoder を同一の移行ツール（TypeWeaver）に接続し、file-level および package-level の type-check success を測定する。結果として、最良モデルでも package-level は 21% と低い一方で、file-level は 69% が type check する、と報告する（成功率の多層性）。

また同論文は、移行ツールとして最低限必要な周辺工程（依存型の導入、JS→TS構文変換、型挿入、予測の拒否/フィルタ）を明示し、単なるモデル出力では“ tsc に通る TypeScript ”を得にくいことを示した。卒論で評価軸を“ type-check success ”に置く場合、TypeWeaver は「なぜその軸が妥当か」を先行研究として裏付ける最重要文献になる。

3.2 OpenTau（arXiv 2023）：分解＋探索＋検証で type-check を直接改善

OpenTau（論文：Type Prediction With Program Decomposition and Fill-in-the-Type Training）は、プログラムが長くてモデル文脈に収まりにくい／関連情報が散在する、という現実的課題に対して、(i) 依存関係に沿ってプログラムを分解し、(ii) 各部分に対して型を生成し、(iii) TypeScript コンパイラで type check しながら探索する、という枠組みを提案する。評価では、従来の生成ベース手法に比べて「type check するファイルの割合」を増やし、最大で 47.4% のファイルが type check し、14.5%（absolute）の改善を報告している。

ここで重要なのは、OpenTau が type checker を“ 後処理の検査 ”に留めず、探索のフィードバックとして統合している点である。つまり、評価指標としての type-check success を、そのまま最適化対象に近づけている。卒論で「type-check成功」を最終指標にする場合、OpenTau は“ type-check を上げるための設計（分解・探索） ”の事例として引用できる。

3.3 DeepTyper / LambdaNet：accuracy 主体の評価と、type-check 観点への接続

DeepTyper（Deep Learning Type Inference）は JS/TS の型推論で広く参照される研究で、コード文脈や関係を学習して型候補を提示する。論文自体は主に annotation-level accuracy で評価するが、「提案された型は type checker によって検証できることが多い」旨を述べ、型チェッカとの関係を早期に意識していた。

LambdaNet（Probabilistic Type Inference using Graph Neural Networks）は TypeScript を対象に、型依存グラフを用いて標準型だけでなくユーザ定義型も推定する。評価は主に精度改善として報告される。一方で、TypeWeaver は DeepTyper/LambdaNet を“ 同じ移行パイプライン ”に接続したときの type-check 成功率を測り、accuracy 主体の評価が実用成功を保証しないことを示す。つまり、TypeWeaver は先行モデル群を否定するのではなく、評価のレイヤを引き上げた位置づけになる。

3.4 型定義の整合性問題（DefinitelyTyped など）と実務ツールの視点

Scotty（Highly illogical, Kirk: spotting type mismatches in the large ...）は、DefinitelyTyped の型定義と実装の不一致（mismatch）を検出する研究であり、静的型が付いていても

“ 正しくない型 ” が現実に存在することを示す。移行成功（type-check success）を最終指標に置く際、外部 .d.ts の品質がボトルネックになり得るという根拠として引用できる。

実務では、Airbnb の ts-migrate のように「まずビルドを通す（compiling TypeScript を得る）」ことを第一段階として設計し、any や @ts-expect-error を残して段階的に改善する戦略が採られる。これは研究側の package/file-level type-check success と相性が良く、“ 移行成功＝最終的に厳密に型安全 ” ではなく、“ まず type-check を成立させ、その後 typedness を上げる ” という二段階評価の妥当性を裏付ける。

4. 近接分野（Python等）に見る“型チェッカ検証”の一般性

TypeScript 以外でも、動的言語に型注釈を追加する研究は多い。そこで共通して現れるのが「予測が正しいか」を最終的に型チェッカ（mypy / Pyre / TypeScript など）で確認する、という設計である。これは“局所一致”と“整合性”のギャップが、言語を超えた構造的問題であることを示唆する。

4.1 TypeWriter（FSE 2020）：予測＋探索＋型チェッカで“確実に通る型”へ

TypeWriter（Neural Type Prediction with Search-Based Validation）は Python を対象に、ニューラル予測の候補を探索し、型チェッカを呼び出して不整合を排除しながら最終解を選ぶ枠組みを提案する。論文は「予測だけでは正しさを保証できない」ことを前提に、型チェッカを validation（検証）として組み込む。結果として“type correctness を保証しつつ”ファイルを自動注釈できる割合を報告する。

4.2 HiTyper（ICSE 2022）：静的推論＋ニューラル予測＋拒否規則

HiTyper（Static Inference Meets Deep Learning）は Python の型推論で、静的推論が得意な部分と DL の補完能力を統合し、さらに“type rejection rules”によって不適切な予測を排除する。ここでも重要なのは、最終的にプログラム全体の整合性を満たす方向に設計が向いている点である（単純な accuracy の最大化ではない）。

4.3 TypeT5（2022）：infilling + 静的解析コンテキスト

TypeT5（TypeT5: Seq2seq Type Inference using Static Analysis）は、欠落型注釈を code infilling として扱い、静的解析で得た文脈をモデル入力に組み込む。TypeScript/JavaScript と Python の両方を視野に入れ、希少型・複雑型の改善を狙う。TypeWeaver/OpenTau が“type-check success”を前面に出すのに対し、TypeT5 は主として予測精度の改善だが、静的解析を入れる方向性は“整合性”の重要性を示す補助線になる。

4.4 TypyBench（2025）：リポジトリ全体の型整合性（TYPECHECK）を測るベンチマーク

TypyBench は、LLM の型推論を“リポジトリ全体”で評価するベンチマークとして提案され、指標として TYPESIM（型の意味的近さ）と TYPECHECK（コードベース全体の型整合性）を導入する。重要なのは、ベンチマーク側が“局所一致だけでは足りない”ことを明示的に認め、整合性指標を設計に組み込んでいる点である。TypeScript における tsc success の採用は、この広い潮流に整合する。

5. 研究設計への示唆：卒論での評価軸としての妥当性

ここまでの先行研究から、卒論で“ 移行成功（type-check成功） ”
を主評価軸に置く妥当性は次のように整理できる。 先行研究が明示的に提案： TypeWeaver は accuracy の代替として type-check success を提案し、実験でその差を示した。
最適化対象としても利用： OpenTau は tsc を探索ループに入れ、type-check 成功率の改善を結果として報告した。 言語横断の一般性： TypeWriter/HiTyper/TypyBench など、型チェッカ検証は“ 整合性問題 ”への共通解として現れる。 実務的整合： ts-migrate は「まずビルドを通す」ことを第一フェーズに置き、研究評価（type-check success）と同型の成功概念を採る。

一方で、type-check success
を採用するときの落とし穴も先行研究から読み取れる。卒論の実験設計では次の対策が推奨される。
単位の選択： file-level / package-level（あるいは repo-level）を分けて報告する（TypeWeaver が示すように結論が変わる）。“ 通るが弱い型 ”の監視： any / unknown / Function などの割合、または typedness を補助指標にする。 エラー種別の内訳：
エラーコード別の増減（回帰）をログとして残す（TS7006 等）。 tsconfig の固定： strictness を統一し、skipLibCheck 等の条件を明記する。

あなたの研究計画（research_overview_v2.md）で述べられている“ Gate A/B ”や Phase ごとのログ設計は、この落とし穴を潰すための前処理に相当する。特に「構文エラーで tsc が先に落ちると偽陽性が出る」という注意は、type-check success を扱う研究が必ず直面するノイズ源であり、先行研究でも移行パイプライン（変換→挿入→検査）の重要性が強調されている。

5.2 卒論でのレポーティング用チェックリスト（提案）

type-check success を主指標にすると、実験条件の差が結果に直結する。先行研究の書き方を踏まえ、卒論の“ 実験 ”節に最低限載せると再現性が上がる項目をチェックリスト化する。

項目	記載例 / 具体化のポイント
対象（ライブラリ/consumer）	対象 package 数、選定条件、コミット固定（lockfile）
tsconfig 条件	strict, noImplicitAny, skipLibCheck, moduleResolution など主要フラグ
成功の定義	file-level success（該当ファイルがエラー0）／package-level success（全ファイルエラー0）
補助指標	エラー総数、エラーコード別内訳（例：TS7006 等）、any/unknown 比率
比較ベースライン	①無変更 ②any挿入のみ ③モデル予測挿入（提案）
回帰（regression）	ベースラインで通っていたのに落ちたケース数、代表例の原因分析
失敗モード分析	依存型不足、JS→TS変換起因、局所型の衝突、設定依存 などを分類

このチェックリストは、TypeWeaver のような“ 移行ツール＋モデル比較 ”系の書き方を卒論に落とし込む際の骨格になる。特に「成功単位の明記」「回帰の扱い」「tsconfig の固定」は、読み手が結果を解釈するために必須である。

6. DeepResearch（深掘り検索）のための推奨クエリ集

卒論の先行研究探索では、(A)“ type check ”を明示的に評価する研究、(B) accuracy を批判・補完する議論、(C) 実務移行での“ build success ” 指標、の3系列を並行で探すと効率がよい。以下に、DeepResearch に投入しやすい形でクエリ例をまとめる（英語中心、必要に応じて日本語併用）。

狙い	推奨クエリ例（そのまま貼れる形）
TypeScript / tsc success 指標	TypeScript" type migration "type check" success rate
accuracy の限界議論	"accuracy can be misleading" type prediction type check
package-level vs file-level	"package" "file" type check success TypeScript migration
探索＋型チェッカ	"search" "type checker" validation type inference
repo-level 指標	"repository" type inference benchmark TYPECHECK metric
外部定義の品質	"DefinitelyTyped" type mismatch detection tool
実務移行（コンパイル通過）	"ts-migrate" compiling TypeScript project out

読み方のコツ： 検索では“ type check / type-check / typechecker / compilation success / build success ”など表記揺れがある。TypeScript の場合は“ tsc --noEmit ” で論文内検索すると評価節に到達しやすい。arXiv PDF なら“ files that type check ” という表現が頻出する。

参考文献（主要）

[1] Ming-Ho Yee, Arjun Guha. Do Machine Learning Models Produce TypeScript Types That Type Check? ECOOP 2023 (LIPIcs).

[2] Francesco Cassano et al. Type Prediction With Program Decomposition and Fill-in-the-Type Training (OpenTau). arXiv:2305.17145, 2023.

[3] Vincent J. Hellendoorn et al. Deep Learning Type Inference. ESEC/FSE 2018.

[4] Jiayi Wei et al. LambdaNet: Probabilistic Type Inference using Graph Neural Networks. arXiv:2005.02161, 2020.

[5] Michael Pradel, Georgios Gousios. TypeWriter: Neural Type Prediction with Search-Based Validation. FSE 2020.

[6] Jiayi Wei et al. TypeT5: Seq2seq Type Inference using Static Analysis. (公開版PDF), 2022.

[7] Julian Hoeflich et al. Highly illogical, Kirk: spotting type mismatches in the large ... ACM (2022).

[8] Airbnb Engineering. ts-migrate: A Tool for Migrating to TypeScript at Scale (記事) / airbnb/ts-migrate (repo).

[9] Ming-Ho Yee. Predicting TypeScript Type Annotations and Definitions with Machine Learning (PhD thesis), 2024.

[10] H. Dong et al. TypyBench: Evaluating LLM Type Inference for Untyped Python Repositories (pdf), 2025.

参照URL（抜粋）

- TypeWeaver (ECOOP 2023 LIPIcs PDF): drops.dagstuhl.de/storage/00lipics/lipics-vol263-ecoop2023/LIPIcs.ECOOP.2023.37/LIPIcs.ECOOP.2023.37.pdf
- TypeWeaver (arXiv): arxiv.org/pdf/2302.12163
- OpenTau (arXiv): arxiv.org/pdf/2305.17145
- DeepTyper (FSE 2018 PDF): vhellendoorn.github.io/fse2018-j2t.pdf
- LambdaNet PDF: maruthg.com/pubs/lambdanet.pdf
- TypeWriter PDF: software-lab.org/publications/fse2020_TypeWriter.pdf
- TypeT5 PDF: www.cs.utexas.edu/~isil/TypeT5.pdf
- Scotty (ACM DOI): dl.acm.org/doi/10.1145/3563305
- ts-migrate blog: medium.com/airbnb-engineering/ts-migrate-a-tool-for-migrating-to-typescript-at-scale-cd23bfeb5cc
- TypyBench (PDF): www.cs.toronto.edu/~fanl/papers/typebench-icml25.pdf

7. 追加候補文献（拾い読み用）

“ type-check success ” を直接扱う／または評価設計に示唆が大きい文献の候補を、検索の起点として列挙する。本文の主要文献（TypeWeaver / OpenTau）を核に、必要に応じて深掘りする。

文献	ポイント（type-check評価との関係）
Yee & Guha (ECOOP 2023) TypeWeaver	package/file-level の type-check success を中心指標として提案。
Cassano et al. (arXiv 2023) OpenTau	tsc を探索ループに組み込み、type-check するファイル割合を改善。
Pradel & Gousios (FSE 2020) TypeWriter	型チェッカで validation しながら探索（Python）。
Wei et al. (2022) TypeT5	静的解析文脈を統合（整合性を意識）。
Dong et al. (2025) TypyBench	repo-level TYPECHECK 指標を導入（Python）。
Hoeflich et al. (2022) Scotty	外部 .d.ts の mismatch 検出 → 移行のボトルネック。
Airbnb (2019) ts-migrate	“ まずコンパイルを通す ” を目的化（実務的 success 定義）。
Hellendoorn et al. (FSE 2018) DeepTyper	accuracy 主体。TypeWeaver で type-check 観点に再評価される。
Wei et al. (2020) LambdaNet	accuracy 主体。TypeWeaver の比較対象。

注：本レポートは TypeScript 中心だが、Python 側の “ 型チェッカ検証 ” 研究やベンチマークは、評価設計の一般性を説明する根拠として有用。卒論の先行研究章では、言語差を明示した上で引用すると通りが良い。