

提出先：（未記入）  
学部・学科：（未記入）  
氏名：（未記入）  
提出日：2026-01-16

---

## ## 概要

TypeScript移行の障壁として、依存ライブラリが型定義(`.d.ts`)を提供しない、または不十分である問題。利用形(usage evidence)から、依存ライブラリ境界で必要となる`.d.ts`を生成し注入することで、下流に関わるエラーを減らす支援手法を提案する。

評価は局所的な型推定精度ではなく、下流プロジェクトを対象とした`tsc`の結果（成功率・エラー分布）を`install→baseline` → `tsc` → 注入 → 再`tsc` → 集計を構築し、生成`.d.ts`の壊れ（構文エラー）を検知して回帰(TS2339/TS2554等)の失敗要因を分類して軽量な安全策（例：`external`判定の改善）を導入した。

---

## ## 1. 背景

TypeScript導入は広く進む一方、依存ライブラリがJavaScriptのみで型定義を提供しない場合、下流プロジェクトの型定義生成は有望だが、生成物の局所的な正しさだけでは下流のコンパイルが通るとは限らない。TypeWeaverの重要性を指摘している。

本研究はこの観点を、依存境界における`.d.ts`注入という介入問題に適用し、下流の`tsc`結果を直接改善する。

---

## ## 2. 研究目的

- JSライブラリの型欠如が原因で失敗する下流プロジェクトに対し、`.d.ts`を注入して`tsc`失敗を減らす。
- 効果測定は下流プロジェクトの`tsc`結果（成功率・エラーコード分布）で行う。
- 回帰（特にTS2339/TS2554）を生む失敗要因を体系化し、軽量な安全策で実験を安定化する。

---

## ## 3. 研究課題 (RQ)

- \*\*RQ1（実用効果）\*\*：`.d.ts`注入により下流プロジェクトの`tsc`成功率・エラー分布はどれだけ改善されるか。
- \*\*RQ2（トリビアル型濫用の懸念）\*\*：改善は単なる`any`での握りつぶしではないか。
- \*\*RQ3（失敗要因）\*\*：回帰はどのエラー（特にTS2339/TS2554）に集中し、原因はどのパターンに分類されるか。

---

## ## 4. 手法概要

### ### 4.1 Phase分解

本研究では、下流`tsc`失敗を意的的に分解し、段階ごとに介入する。

- \*\*Phase1（型解決）\*\*：TS2307/TS7016（型定義が見つからない）
- \*\*Phase2（境界整合）\*\*：TS2305/TS2613/TS2614（`export/import`形の不一致）
- \*\*Phase3（API整合）\*\*：TS2339/TS2554/TS2345/TS2322/TS2769/TS2353/TS2741/TS7053（APIレベルの整合性）

本ドラフトでは、主にPhase3（API整合）の実験基盤と失敗要因分析に焦点を当てる。

### ### 4.2 Usage evidence抽出 (Phase3)

Phase3では、Phase3 coreコードが出たファイル群から、外部importを抽出して「注入すべきモジュール」で、`ns.foo`や`Foo.bar`のような参照を解析して、必要なメンバ名も収集する（回帰抑制のため）。

### ### 4.3 `d.ts`生成と注入

生成方法は2系統を持つ。

- \*\*DTS\_STUB\*\*: importされた名前を`any`として宣言する下限ベースライン。
- \*\*DTS\_MODEL\*\*: 小型コード生成モデル（例：Qwen2.5-Coder-1.5B-Instruct）で`declare module`を生成し、危険な構文はサニタイズする。

注入は repo 内に `typeRoots` を作り、派生tsconfig (`tsconfig.\_\_phase3\_\_.json`) を介して確

---

## ## 5. 実装

### ### 5.1 Real runner

- `evaluation/real/phase3-run.mjs`
- clone → install → baseline tsc
- Phase3診断ファイルからimport抽出
- `d.ts`生成(stub/model)
- `d.ts`注入後に再tsc
- 結果を`results.jsonl`と`summary.tsv`に保存

### ### 5.2 Model adapter

- `evaluation/model/typebert\_infer.py`
- stdin JSON (modules) → stdout JSON (dts文字列)
- `declare module`ブロック抽出(brace balancing)
- サニタイズ、危険ならstubにフォールバック
- 結果追跡のため`cache\_key`と`meta`を返す

### ### 5.3 実験の信頼性 (invalid検知)

LLM生成`d.ts`は壊れる可能性があり、壊れた場合はTS1005/TS1127/TS1434等で`tsc`が先に落ち、Phase3が見えてくる。本研究では`invalid`を検知し、集計で除外する。

### ### 5.4 回帰抑制の安全策 (external判定)

大規模repoでは、`core/\*`のような内部aliasを外部扱いして注入してしまうとTS2339が爆増する。これを防ぐために含まれるパッケージのみを外部とみなす \*\*depsフィルタ\*\* (`--external-filter deps`) を導入した

---

## ## 6. 実験

### ### 6.1 実験設定

対象: `evaluation/real/inputs/phase3\_ts1000\_ranked100.txt` の先頭30件  
条件: `--external-filter deps` を有効化し、`--max 30` で評価

出力ディレクトリ:

- `evaluation/real/out/phase3-ts1000-ranked30-qwen1\_5b-v17-depsfilter/`

### ### 6.2 指標

- valid injection数 (invalid/timeout除外)
- Phase3Reduced / Phase3Eliminated
- Phase3 core合計差分
- TS2339/TS2554などコード別差分

### ### 6.3 結果 (ranked30 / depsfilter)

集計 (validのみ) :

- repos\_total: 30
- repos\_valid\_injection: 11
- repos\_invalid\_dts: 1
- repos\_model\_timeout: 6
- phase3Reduced\_valid: 3
- phase3Eliminated\_valid: 3

Phase3 core合計 (validのみ) :

- baseline: 194
- injected: 210
- delta: +16

コード別差分 (validのみ) :

- TS2339: 71 → 84 (+13)
- TS2554: 10 → 17 (+7)
- TS2345: 33 → 25 (-8)
- TS2769: 20 → 13 (-7)

改善例 (valid内の上位) :

- type-challenges: 21 → 0 (eliminated)
- vee-validate: 11 → 0 (eliminated)
- shoelace: 7 → 0 (eliminated)

---

## ## 7. 考察

### ### 7.1 改善が得られるケース

一部repoではPhase3 coreが0化し、`d.ts`注入が下流`tsc`に実用的効果を持ちうることが確認できた。

### ### 7.2 失敗要因 (TS2339/TS2554回帰)

本研究の観測では回帰はTS2339/TS2554に集中する傾向があり、原因は少なくとも以下に分類できる。

- **外部判定ミス**: 内部alias (例: `core/\*`) を外部扱いし注入することで、既存の型関係を壊してTS → depsフィルタで抑制可能。
- **export欠落**: モデルが必要なexport名を出し忘れることでTS2339が増える。  
→ requested exportsの`any`補完で抑制可能。
- **静的メンバ参照**: `Foo.bar` のような参照が、型宣言側で表現できずTS2339を生む。  
→ namespace merge (`export namespace Foo { const bar:any }`) で一部吸収可能。
- **invalid生成**: `d.ts`が壊れると偽陽性が起きる。  
→ invalid検知・除外が必須。

### ### 7.3 卒論としての位置づけ

平均的改善の主張よりも、「下流`tsc`で評価する設計」「失敗要因の体系化」「軽量安全策で実験を安定化す」にしても論文として主張が成立する。

---

## ## 8. 新規性

- **評価軸**: 局所精度ではなく、下流プロジェクトの`tsc`結果（成功率・エラー分布）を一次指標として

- **\*\*Phase分解\*\***: TSエラーコードを段階にマッピングし、介入の意味を整理（解決→境界→API整合）。
- **\*\*運用可能な評価基盤\*\***: `invalid`検知・フォールバック・追跡 (`cache_key/meta`) を含む実験基盤を構築。
- **\*\*失敗要因の知見\*\***: TS2339/TS2554回帰を中心に原因を分類し、`deps`フィルタ等の軽量安全策で抑制する。

---

## ## 9. 妥当性への脅威

- 外的妥当性: 対象repoの偏り（言語設定、monorepo構造、依存の種類）。
- 内的妥当性: `tsconfig`/`strictness`差によるノイズ、`timeout`の扱い。
- 構成概念妥当性: `tsc`成功が実行時安全性を保証しない点。

---

## ## 10. 結論と今後

本研究は、下流使用例から`.d.ts`を生成・注入し、下流`tsc`で評価するエンドツーエンド基盤を構築した。9/TS2554回帰が主要な課題である。`deps`フィルタ等の安全策により回帰を抑制し、より解釈可能な評価を可能にする要因に基づく追加の安全策、必要に応じたモデル比較を行う。

---

## ## 参考文献（暫定）

1. TypeWeaver: Learning Type Annotations from Examples (ECOOP 2023). <https://drops.dagstuhl.de/storage/00lipics/lipics-vol263-ecoop2023/LIPIcs.ECOOP.2023.37/LIPIcs.ECOOP.2023.37.pdf>