

PROJECT REVIEW 2

AuxFlow: Static Analysis for Multi-Pass Document Compilation

1. Project Details

Project Title	AuxFlow: Static Analysis for Multi-Pass Document Compilation
Student Name(s)	Upayan Mazumder
Register Number(s)	24BDS0367
Programme / Branch	B.Tech CSE (Data Science)
Project Guide	Banupriya R
Review Date	28 October 2026

2. Project Overview

Introduction

AuxFlow is a static analysis tool for LaTeX documents that need more than one compilation pass. It has its own compiler front end (lexer, parser, AST, symbol tables) and an intermediate representation called the Auxiliary Reference Graph (ARG). Graph analysis over the ARG classifies the cross-reference dependency structure of a document before the LaTeX engine runs.

Problem

Cross-references, tables of contents, and bibliographies make LaTeX write auxiliary files that the next pass reads. A document is stable only when these files stop changing. Build tools find the pass count by trial: they run the engine until the files stop changing, or they run a fixed schedule such as three passes. A fixed schedule wastes runs on simple documents and stops too early on documents that need more. A small number of documents never stabilise because a page number controls how much content is generated. No tool we surveyed (l^atexmk, l^atexrun, Tectonic) analyses this dependency structure before it runs the engine.

Objectives

- Build a compiler front end (lexer, parser, AST, symbol tables, semantic checks) for a declared subset of LaTeX.
- Lower the resolved AST into the Auxiliary Reference Graph, an instance-level intermediate representation with typed, lagged dependency edges.

- Classify each document into one of four convergence states (CERTIFIED, BENIGN_CYCLIC, AMPLIFYING, UNKNOWN) with strongly connected components and taint analysis.
- Derive a safe build plan with a predicted engine run count, and check that prediction at runtime.
- Measure the approach on a benchmark corpus against seven baselines and against latexmk, latexrun, Tectonic, and Typst.

Approach

AuxFlow treats pass-count selection as a compiler problem. It parses the source once, builds a graph of every construct that reads or writes auxiliary state, and decides from that graph whether the auxiliary state settles and after how many runs. For a certified document the tool predicts the run count in under 10 ms with no engine execution. For constructs outside the declared subset it reports UNKNOWN and uses a conservative schedule.

3. Implementation Progress

Progress: 16 of 16 implementation phases complete (100% of the implementation plan). This document covers the front end and the intermediate representation (Phases 1 to 7).

Phases	Scope	Status
1	Project setup and test infrastructure	Complete
2 to 7	Lexer, parser, AST, symbol tables, semantic checks, IR	Complete
8 to 11	SCC analysis, taint analysis, stable-pass fixpoint, classifier	Complete
12 to 13	Build planner, runtime verification	Complete
14 to 16	Benchmark corpus, harness, evaluation report	Complete

System architecture

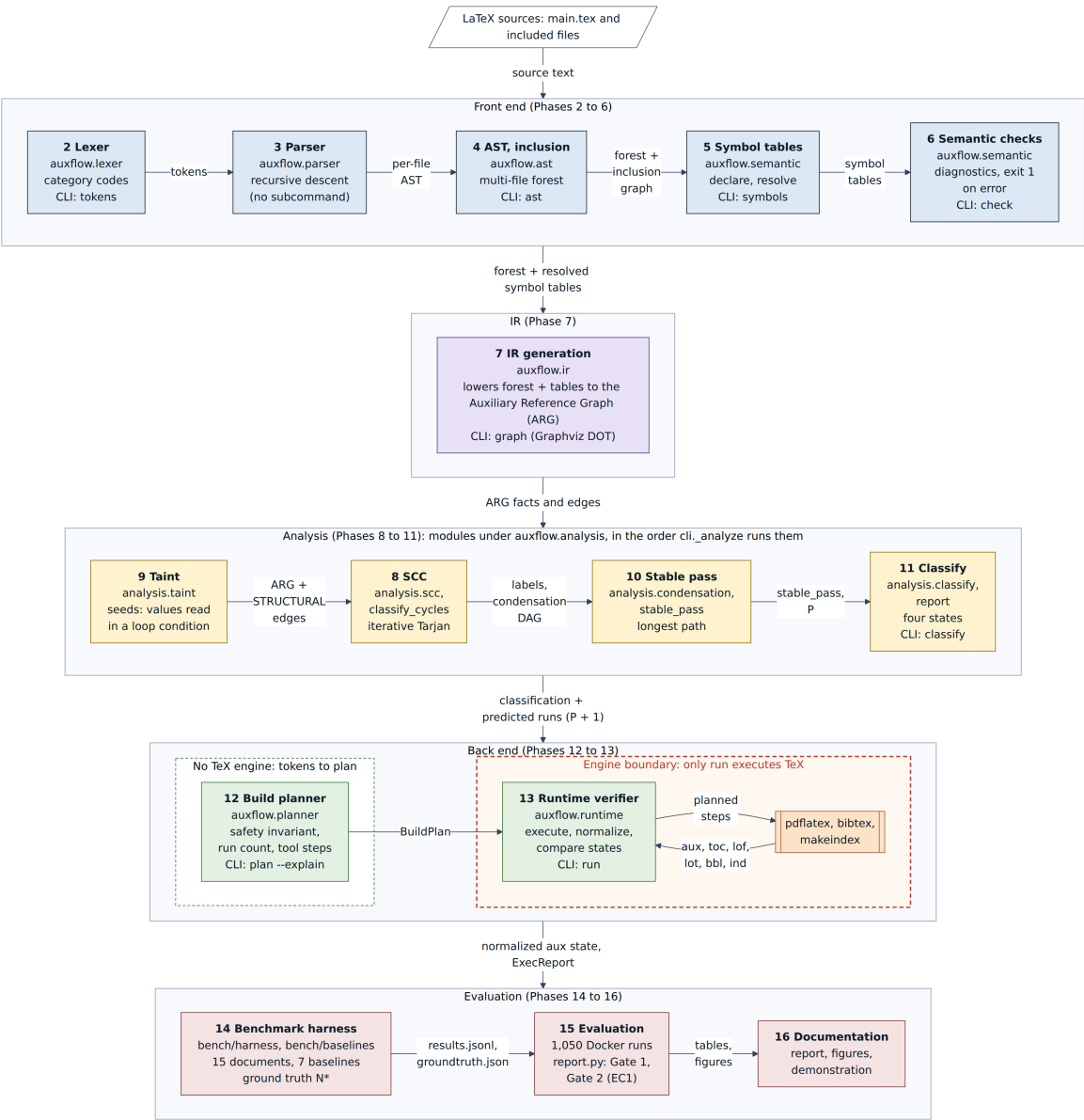


Figure 1. AuxFlow compiler pipeline. Phases 1 to 7 are the subject of this document.

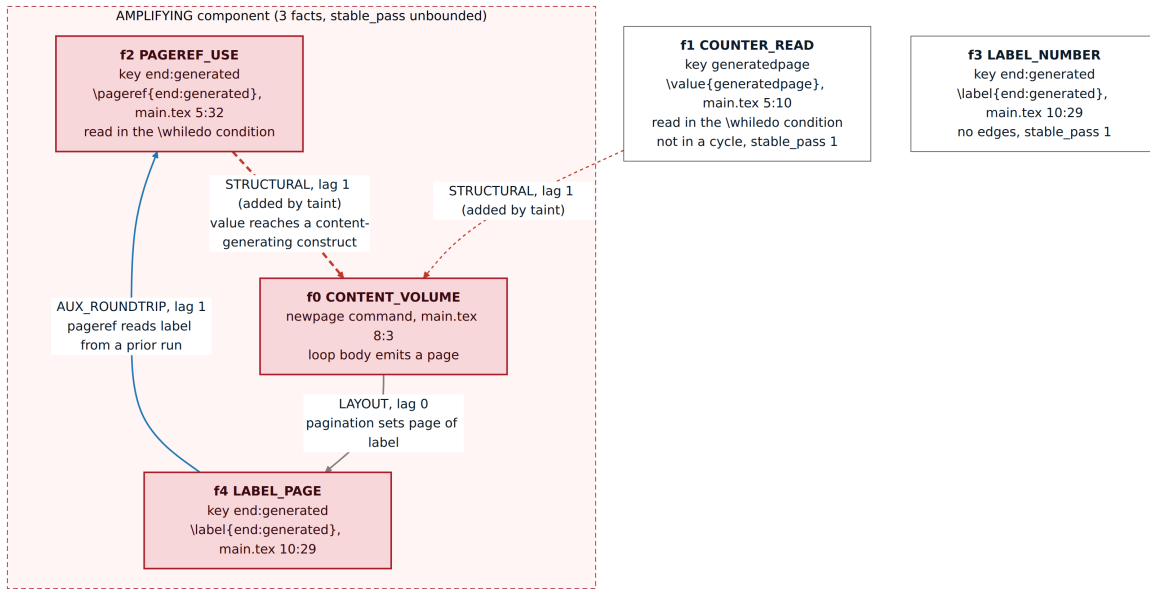


Figure 2. Auxiliary Reference Graph of `b08_cycle` (auxflow graph). Taint analysis adds the dashed `STRUCTURAL` edges. The thick one closes the `AMPLIFYING` cycle.

Implemented modules (Phases 1 to 7)

- **Lexer (auxflow.lexer):** Single-pass, table-driven scanner for the sixteen TeX category codes. It recognises comments, control words and symbols, and `\verb` verbatim regions. A property test checks that the tokens reproduce the exact source bytes.
- **Parser (auxflow.parser):** Hand-written recursive-descent parser with one token of lookahead. A macro-signature table drives mandatory and optional argument parsing. The parser produces one AST per file over sixteen node kinds, and every node has a source span. Malformed input, such as an unterminated group, becomes a diagnostic and does not raise an exception.
- **AST and inclusion resolution (auxflow.ast):** Joins the per-file ASTs into a multi-file forest and builds the inclusion graph from `\input` and `\include`. Depth-first search detects circular inclusion and files that the entry file cannot reach.
- **Symbol tables (auxflow.semantic):** Six tables (LabelTable, RefTable, CiteTable, PackageTable, FileTable, CounterTable). A declare pass records every definition, then a resolve pass binds the uses, so forward and cross-file references resolve.
- **Semantic analysis (auxflow.semantic):** Checks that need no engine run: undefined references and citations, duplicate labels, missing included files, missing graphics files, inclusion cycles, and unmodelled packages.
- **Intermediate representation (auxflow.ir):** The ARG has one fact per modelled construct (label, reference, contents entry, citation, counter read) with a source span. Typed edges (`AUX_ROUNDTRIP`, `TOOL`, `LAYOUT`, `STRUCTURAL`) carry a pass lag. Each list entry is split

into an exists fact and a value fact, so a document with only a table of contents stays acyclic. The graph serialises to JSON and Graphviz DOT.

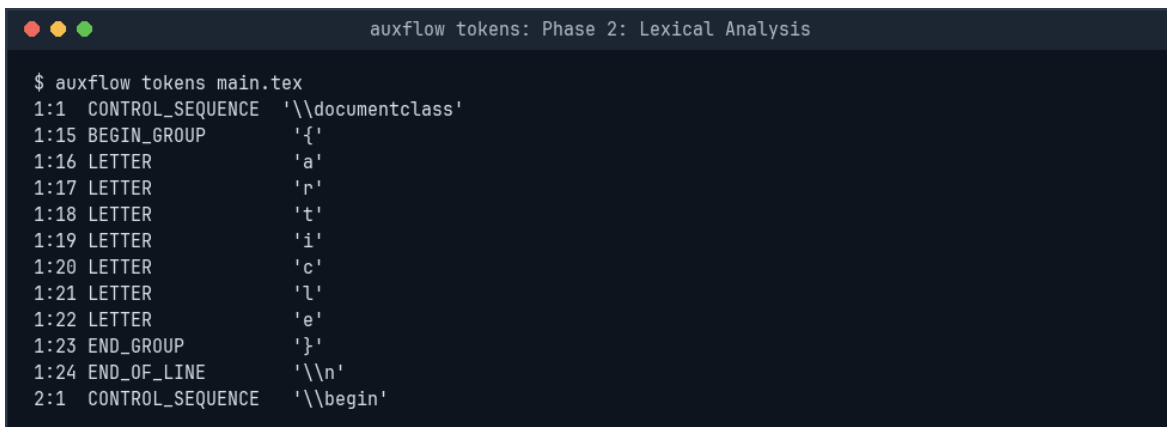
- **Command-line interface (auxflow.cli):** auxflow tokens, ast, symbols, check, and graph expose Phases 2 to 7. The same CLI also has classify, plan, and run for the Review 3 phases.

Technologies

- Python 3.12 or later. CI tests 3.12 and 3.13, and the suite also passes on 3.14.
- No third-party runtime dependencies. The lexer, parser, symbol tables, IR, and the Tarjan SCC algorithm are written by hand.
- pytest and Hypothesis for unit, property, golden, and end-to-end tests.
- mypy in strict mode for type checking, and ruff for linting and formatting.
- Docker for the benchmark: a pinned image with TeX Live, latexmk, latexrun, Tectonic, and Typst.
- Input language: a documented subset of LaTeX2e.

Screenshots

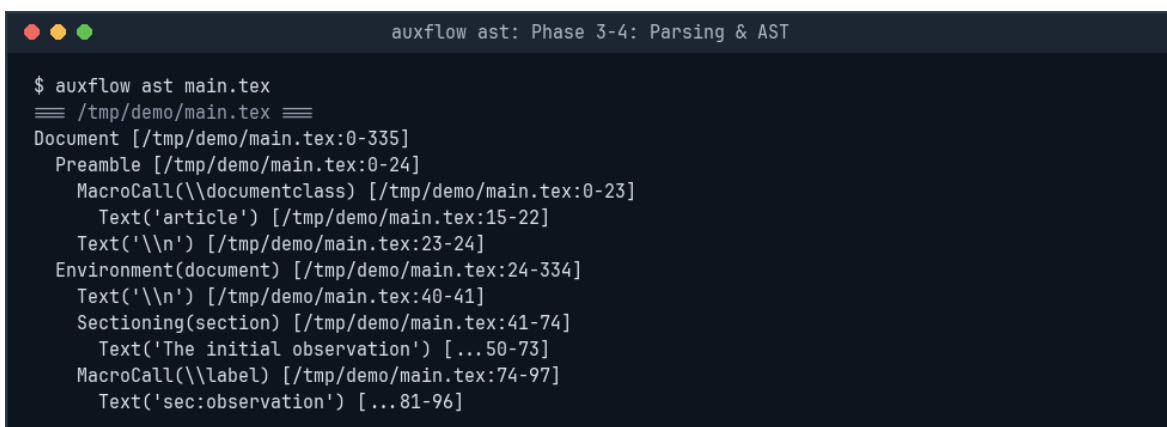
Each image shows a real auxflow command and its output on small test documents.



```
auxflow tokens: Phase 2: Lexical Analysis

$ auxflow tokens main.tex
1:1  CONTROL_SEQUENCE  '\\documentclass'
1:15 BEGIN_GROUP       '{'
1:16 LETTER            'a'
1:17 LETTER            'r'
1:18 LETTER            't'
1:19 LETTER            'i'
1:20 LETTER            'c'
1:21 LETTER            'l'
1:22 LETTER            'e'
1:23 END_GROUP         '}'
1:24 END_OF_LINE       '\\n'
2:1  CONTROL_SEQUENCE  '\\begin'
```

Phase 2, lexical analysis: auxflow tokens



```
auxflow ast: Phase 3-4: Parsing & AST

$ auxflow ast main.tex
≡≡ /tmp/demo/main.tex ≡≡
Document [/tmp/demo/main.tex:0-335]
  Preamble [/tmp/demo/main.tex:0-24]
    MacroCall(\\documentclass) [/tmp/demo/main.tex:0-23]
      Text('article') [/tmp/demo/main.tex:15-22]
      Text('\\n') [/tmp/demo/main.tex:23-24]
  Environment(document) [/tmp/demo/main.tex:24-334]
    Text('\\n') [/tmp/demo/main.tex:40-41]
    Sectioning(section) [/tmp/demo/main.tex:41-74]
      Text('The initial observation') [...50-73]
    MacroCall(\\label) [/tmp/demo/main.tex:74-97]
      Text('sec:observation') [...81-96]
```

Phases 3 and 4, parsing and AST: auxflow ast

```
auxflow symbols: Phase 5: Symbol Tables

$ auxflow symbols main.tex
LabelTable:
  sec:observation: /tmp/demo/main.tex:74-97
  counter=section duplicate=False
RefTable:
  sec:observation: ref /tmp/demo/main.tex:236-257
  → /tmp/demo/main.tex:74-97
CiteTable:
PackageTable:
FileTable:
  /tmp/demo/main.tex: path=/tmp/demo/main.tex
  includer=- reachable=True opaque=False
CounterTable:
```

Phase 5, symbol tables: auxflow symbols

```
auxflow check: Phase 6: Semantic Analysis

$ auxflow check main.tex
(no diagnostics)
$ echo $?
0

$ auxflow check broken.tex
broken.tex:45-62: error: undefined reference: 'fig:missing'
broken.tex:100-123: error: duplicate label: 'sec:observation'
$ echo $?
1
```

Phase 6, semantic analysis: auxflow check on a clean and a broken file

```
auxflow graph: Phase 7: Intermediate Representation

$ auxflow graph main.tex
digraph arg {
  rankdir=LR;
  f0 [label="LABEL_NUMBER\\nsec:observation\\n[value]"];
  f1 [label="LABEL_PAGE\\nsec:observation\\n[value]"];
  f2 [label="REF_USE\\nsec:observation\\n[value]"];
  f0 → f2 [label="AUX_ROUNDTRIP lag 1"];
}
```

Phase 7, intermediate representation: auxflow graph (Graphviz DOT)

```
auxflow classify: CERTIFIED (acyclic)

$ auxflow classify main.tex
auxflow: CERTIFIED -- 2 runs

longest dependency chain (stable_pass 1):
  LABEL_NUMBER(sec:observation)  main.tex:3:34  → 1
  LABEL_PAGE(sec:observation)     main.tex:3:34  → 1
  REF_USE(sec:observation)        main.tex:7:39  run 2

The facts above require all 2 runs to stabilise.
```

Phase 11 classifier on the same IR: a CERTIFIED document

```
auxflow classify: AMPLIFYING (unmodelled risk, refused)

$ auxflow classify cycle.tex
auxflow: AMPLIFYING CYCLE -- this document may not converge.

cycle (3 facts):
  PAGeref_USE(end:generated) cycle.tex:5:32
    → CONTENT_VOLUME cycle.tex:8:3 STRUCTURAL
    → LABEL_PAGE(end:generated) cycle.tex:10:29
    → PAGeref_USE(end:generated)

No build plan emitted. Use --force to execute under a cap of 6 runs.
$ echo $?
1
```

Phase 11 classifier: an AMPLIFYING document

```
Quality gates: pytest, mypy, ruff

$ python -m pytest
245 passed, 1 skipped, 8 warnings in 2.16s

$ mypy
Success: no issues found in 41 source files

$ ruff check src tests
All checks passed!
```

Test suite, strict type check, and lint

Results

- All 16 phases are implemented: about 3,930 lines of implementation code in 41 modules, and about 4,080 lines of test code in 46 files. The Review 2 modules (lexer, parser, AST, semantic analysis, IR) are about 2,290 of those lines.
- 245 tests pass and 1 is skipped. mypy --strict reports no issues in 41 source files, and ruff reports no lint issues in src, tests, and bench.
- The parser recovers from unterminated groups and missing arguments without an exception. Dedicated recovery tests cover these cases.
- The exists/value split keeps the table-of-contents benchmark (b05_toc) acyclic: it classifies CERTIFIED.
- On the 15-document corpus, every document classified as predicted before implementation, and both evaluation gates passed.

Remaining work

- Write the research paper. The paper/ directory has the section structure, and the evaluation results it needs are available.

4. Novelty

Contribution

AuxFlow applies standard compiler construction (lexing, parsing, symbol tables, an intermediate representation, and graph dataflow analysis) to multi-pass LaTeX compilation. Existing tools handle that problem only by running the engine again. The central design choice is an instance-level dependency graph in which each list entry is split into an exists fact and a value fact, so ordinary documents stay acyclic.

Comparison with existing tools

latexmk, latexrun, and Tectonic decide after each engine run whether another run is needed. They read rerun warnings or compare auxiliary-file checksums. AuxFlow makes this decision from the source before the first run. It classifies the document, predicts the run count when the document is certified, and names the facts and source locations that form a risky cycle.

Specific techniques

- A declared LaTeX subset with a default-deny rule: an unsupported construct makes the document UNKNOWN, and the tool does not assume it is safe.
- The exists/value split in the IR, which stops a table of contents from appearing as a false cycle.
- Four classification states. Value-only cycles (BENIGN_CYCLIC) are kept separate from cycles in which a value controls how much content is generated (AMPLIFYING).
- Static prediction paired with runtime verification. The verifier records a prediction miss when the real run count exceeds the prediction.