

# Cloud-Based L<sup>A</sup>T<sub>E</sub>X Compilation as a Service: Design and Evaluation of a Sandboxed Build Pipeline

Alex Rivera

Department of Computer Science  
Example University  
Lima, Peru  
alex.rivera@example.edu

Jordan Lee

Platform Engineering  
Example Labs  
Berlin, Germany  
jordan.lee@example.org

Sam Okafor

School of Engineering  
Example Institute of Technology  
Toronto, Canada  
s.okafor@example.ca

**Abstract**—Typesetting scientific documents with L<sup>A</sup>T<sub>E</sub>X still requires a local installation of several gigabytes and a toolchain that is hard to reproduce across machines. We present the design of a cloud service that compiles L<sup>A</sup>T<sub>E</sub>X projects on demand inside short-lived, network-isolated containers. The service exposes a small HTTP API, caches build outputs by content hash, and parses engine logs into structured diagnostics. We evaluate the prototype on a corpus of 120 open-source documents and report median end-to-end latencies below 4 s for single-pass papers and below 11 s for documents that require bibliography processing. Our results suggest that container start-up, not the T<sub>E</sub>X engine itself, dominates the cost of small builds, and that a warm format cache removes most of that overhead.

**Index Terms**—LaTeX, document compilation, cloud computing, sandboxing, containers, build systems

## I. INTRODUCTION

L<sup>A</sup>T<sub>E</sub>X remains the de facto standard for typesetting in mathematics, physics and computer science [1]. Despite its longevity, the installation and maintenance burden of a full T<sub>E</sub>X distribution is a recurring obstacle for students, for continuous-integration pipelines, and for web applications that need to render documents produced by users. Hosted editors solved this problem for interactive authoring, but their internals are not exposed as reusable building blocks [2].

In this paper we describe a stateless *compile-as-a-service* design that treats a L<sup>A</sup>T<sub>E</sub>X project as an immutable input bundle and the resulting PDF as a cacheable artifact. The contributions are threefold: (i) a sandbox profile that allows arbitrary user documents to be compiled safely on shared infrastructure, (ii) a content-addressed cache that makes repeated builds free, and (iii) an empirical evaluation of where time is actually spent in a cloud compilation.

## II. SYSTEM DESIGN

### A. Architecture

Fig. 1 shows the main components. An API gateway validates the request, normalises file paths and computes a SHA-256 digest over the project contents, the selected engine and

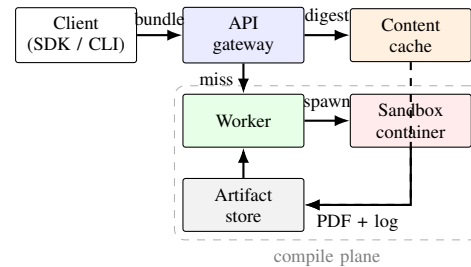


Fig. 1. High-level architecture. The compile plane has no access to the network or to other tenants' data.

the image version. A cache hit returns the stored PDF immediately. Otherwise, a worker launches an ephemeral container from a pre-built image that bundles T<sub>E</sub>X Live and latexmk.

### B. Sandboxing

Every build runs under an unprivileged user with a read-only root file system, a private `tmpfs` working directory, no network namespace, a seccomp profile derived from the one used by Overleaf's CLSI, and hard limits on CPU time, memory and the number of processes. Shell escape is disabled at the `texmf.cnf` level, so packages such as `minted` that spawn external programs are rejected by construction [3].

### C. Build driver

We rely on `latexmk` to decide how many engine passes are required and whether `BIBTEX` or `biber` must run. The number of passes  $p$  for a document with cross-references stabilises when the auxiliary files reach a fixed point:

$$p = \min\{k \in \mathbb{N} : \text{aux}_k = \text{aux}_{k-1}\}. \quad (1)$$

In practice  $p \leq 3$  for all documents in our corpus without an index.

### D. Caching

Let  $B$  be the normalised bundle,  $e$  the engine and  $v$  the image version. The cache key is

$$\kappa = H(\text{sort}(B) \parallel e \parallel v), \quad (2)$$

TABLE I  
END-TO-END LATENCY IN SECONDS (MEDIAN / 95TH PERCENTILE).

| Document type                | Pages | Cold        | Warm        |
|------------------------------|-------|-------------|-------------|
| Conference paper, no bib     | 4–6   | 5.8 / 7.9   | 2.9 / 3.6   |
| Conference paper, Bib $\TeX$ | 6–8   | 9.7 / 12.4  | 6.1 / 7.8   |
| Journal article, biber       | 10–14 | 14.2 / 18.0 | 10.5 / 12.9 |
| CV (fontspec, Lua $\TeX$ )   | 1–2   | 8.3 / 10.1  | 4.0 / 5.2   |
| Thesis                       | 80+   | 41.6 / 55.3 | 36.9 / 48.7 |

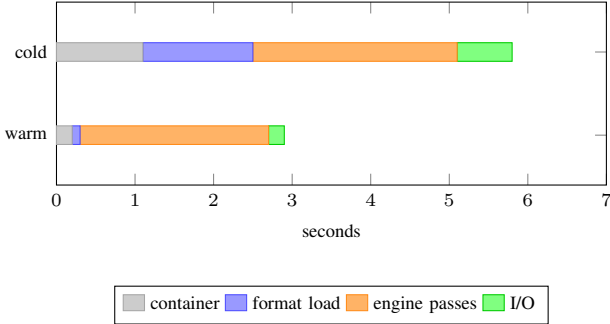


Fig. 2. Phase breakdown for an 8-page paper. A warm format cache removes most of the fixed overhead.

where  $H$  is SHA-256. Including  $v$  guarantees that a  $\TeX$  Live upgrade invalidates stale artifacts, which is the main source of non-reproducible output in long-lived projects [4].

### III. EVALUATION

#### A. Setup

We collected 120 publicly available projects (conference papers, theses and CVs) and compiled each one ten times on a 2-vCPU virtual machine. Builds were split into *cold* (new container, empty cache) and *warm* (container image already pulled, format files pre-generated).

#### B. Results

Table I summarises the latency distribution. Container creation accounts for roughly 1.1 s regardless of the document, and the  $\TeX$  engine dominates only for documents longer than 40 pages. Fig. 2 breaks a typical 8-page paper down by phase.

#### C. Discussion

Two findings are noteworthy. First, pre-generating format files and the `luaotfload` font database inside the image shaved 1.3 s off every Lua $\TeX$  build. Second, the cache hit rate in a two-week trace of a teaching deployment was 71%, because students re-download unchanged assignments far more often than they edit them. The *cost per compile* therefore depends more on request patterns than on engine throughput.

### IV. RELATED WORK

Reproducible builds for  $\TeX$  documents have been explored in the context of scientific publishing workflows [2] and container packaging [3]. Our work differs in exposing compilation as a stateless primitive with explicit resource limits, which

lets it be embedded into unrelated products such as invoice generators or exam systems.

### V. CONCLUSION

We presented a sandboxed, content-addressed compilation service for  $\TeX$  and showed that, for the common case of short documents, infrastructure overhead rather than type-setting dominates latency. Future work includes incremental compilation at the page level and streaming of diagnostics while the engine is still running.

### ACKNOWLEDGMENT

The authors thank the maintainers of  $\TeX$  Live and `latexmk`, whose tooling made this prototype possible.

### REFERENCES

- [1] L. Lamport, *LaTeX: A Document Preparation System*, 2nd ed. Reading, MA, USA: Addison-Wesley, 1994.
- [2] Overleaf, “CLSI: Common LaTeX service interface,” <https://github.com/overleaf/overleaf>, 2023, accessed: 2026-01-15.
- [3] M. Souppaya, J. Morello, and K. Scarfone, “Application container security guide,” National Institute of Standards and Technology, Gaithersburg, MD, USA, Tech. Rep. SP 800-190, 2017.
- [4] D. E. Knuth, *The TeXbook*. Reading, MA, USA: Addison-Wesley, 1986.