Passage2Quizzes: Faithful Triple-Language Ports of a Local-LLM Essay-and-Quiz Pipeline

Shyamal Chandra
Sole Techincal Founder
Domino Data Systems
domino-data-systems.github.io
Email: dominodatasystems@gmail.comt

***Abstract*—Instructors and curriculum designers increasingly draft reading passages and comprehension quizzes with large language models (LLMs). A 52-line Bash prototype, `main.sh`, already performed that job on a local Ollama host: it read a topic list, asked Llama 3.2 for a five-paragraph essay and a ten-item quiz with an answer key, inserted LaTeX page breaks, and converted the Markdown packet to PDF with Pandoc. This industrial-track report describes a full-fidelity port of that pipeline to ISO C99 (with POSIX process control), Rust, and Swift. The ports are native programs, not shell wrappers. They preserve the original prompts, model name, Markdown layout, macOS `sed` page-break semantics, and operator-facing progress lines, while reading topics as newline-separated batch input and failing closed on missing tools. Deterministic stubs, golden files captured from Bash and BSD `sed`, black-box and UX suites, and a three-way A/B comparison show byte-identical `questions.md` across languages. On Apple Silicon, the local transform is $1.16\,\mu$s per page-break call; a three-topic stub pipeline finishes in $60\,$ms (C99/Rust) and $540\,$ms (Swift). A live Llama 3.2 run spent $8\,$s on the essay and $4\,$s on the quiz, confirming that generation, not porting overhead, dominates wall time.**

***Index Terms*—local LLM, quiz generation, language portability, golden-file testing, C99, Rust, Swift, Ollama, Pandoc**

I. Introduction

Compound AI systems glue a model to files, prompts, and downstream formatters [1]. In education, that glue is often a shell script: cheap to sketch, expensive to harden, and awkward to ship inside a C, Rust, or Swift product [2], [3]. Passage2Quizzes began as such a sketch. Given a topics file and a PDF path, `main.sh` (i) writes a Markdown header, (ii) loops over topics, (iii) calls `ollama run llama3.2` twice per topic, (iv) rewrites the quiz with two `sed` substitutions so Pandoc emits page breaks before the quiz body and the answer key, and (v) invokes Pandoc [4], [5].

The industrial problem is not "call an API." It is *fidelity*: operators already trust the Bash UX (progress lines, integer-second timers, a hard-coded `questions.md` in the working directory) and the layout quirks of BSD `sed` on macOS. A rewrite that "improves" those details silently breaks downstream PDF pagination. At the same time, a production port must not inherit Bash's lack of I/O checks or its `$(cat file)` globbing.

This paper reports a completed port of Passage2Quizzes to three deployment languages used at Domino Data Systems, with a test harness that treats Bash+`sed` as an oracle for bytes, not as a runtime dependency.

a) Contributions:

- Native C99+POSIX, Rust (stdlib), and Swift implementations of the full pipeline, including exact prompt strings and Markdown assembly.
- A two-pass, first-match-per-line page-break transform that reproduces macOS BSD `sed` replacement of `Here` and `Answer Key` by `\newpage` fences [6].
- A test matrix covering unit, golden-file regression, black-box, UX, A/B, and optional live-model runs.
- Measurements showing that port overhead is negligible beside local Llama 3.2 generation [7], [8].

II. Background

A. Local LLMs in the authoring loop

Hosted chat services are convenient but couple curriculum shops to network policy, retention, and per-token cost. Ollama [9] runs open weights on-device; Llama 3.2 is a small instruction model intended for edge deployment [7], [8]. Automatic item generation is a long-standing assessment topic [10]; LLMs change the cost of a first draft, not the need to paginate a packet that a student can sit down with.

B. Why three languages

C99 remains the lingua franca of embedded and POSIX tools [11], [12]. Rust is the default for new systems services that must not alias buffers [13]. Swift is the native CLI story on macOS [14]. Cross-language reimplementation is a classic empirical question [15], [16]: here the acceptance criterion is not "idiomatic X" but *observable equality* of artifacts.

C. The Bash prototype as a specification

`main.sh` is 33 lines of code (52 with blanks and comments). It is also an incomplete specification. Command substitution strips trailing newlines; `echo -e` interprets escapes in the structural wrappers but not, under Homebrew Bash 5.3, in the essay body; `sed` without `g` replaces the first match *per line*; the second `sed` reads the first's multi-line output. Those details were recovered by running the original commands on macOS 26 (Darwin 25.6) and storing golden files. That is closer to characterization testing of legacy behavior [17], [18] than to a clean-room redesign.

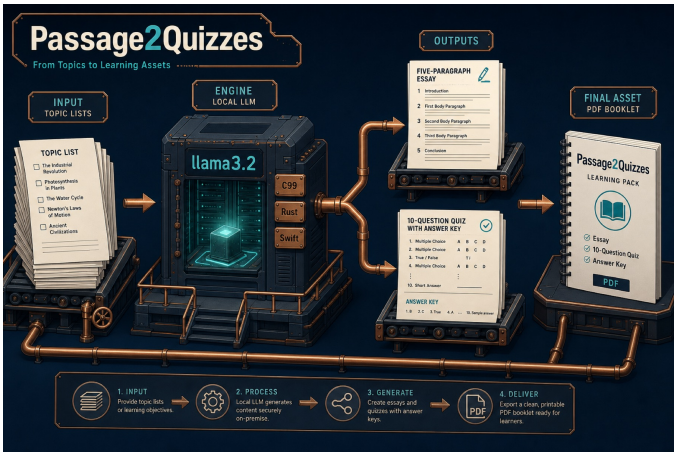

Fig. 1. Pipeline concept: newline-separated topics, local Llama 3.2, essay plus quiz, Markdown packet, Pandoc PDF.

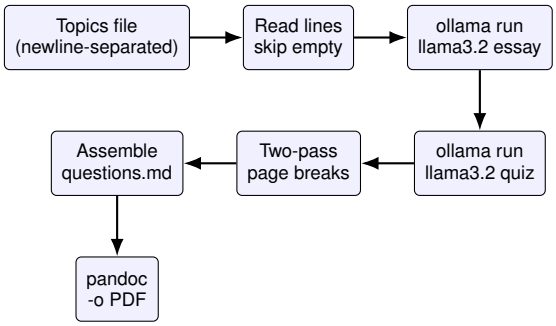

Fig. 2. Control flow of Passage2Quizzes. The inner loop (essay, quiz, page-break, append) repeats once per topic; Pandoc runs once.

III. Algorithm

Figure 2 states the data flow. Figure 1 shows the industrial metaphor used in the accompanying poster: topics enter a local generator; an essay and a quiz leave; Pandoc binds the packet.

A. CLI contract

Let $argv_0$ be the program name. If $|argv| \neq 3$, the process prints

`Usage:` $argv_0$ `<topics_file> <output_pdf_file>`

to standard output and exits 1. Otherwise the topics path is $argv_1$ and the PDF path is $argv_2$. The Markdown path is the constant `questions.md` in the current working directory, matching the prototype.

B. Topic scan

A topic is a maximal byte sequence not containing `LF` (`0x0A`). Empty sequences are discarded; sequences that contain only spaces are kept. A missing terminal newline still yields a topic. The file is read as bytes so UTF-8 proper nouns survive (`Newton's Laws of Motion`). This is the batch interface requested for production: many essays and quizzes from one list, without Bash pathname expansion.

Algorithm 1 Passage2Quizzes pipeline

Require: topics path T, PDF path P
$L \leftarrow$ nonempty lines of T
$M \leftarrow$ `"# Generated Essays and Quizzes\n\n"`
for topic t in L **do**
  announce essay; $e \leftarrow \text{Ollama}(p_e(t))$
  announce quiz; $q \leftarrow \text{Ollama}(p_q(e))$
  $q \leftarrow \text{PageBreak}(q)$
  $M \leftarrow M \cdot \text{Section}(t, e, q)$
end for
write M to `questions.md`
$\text{Pandoc}(\texttt{questions.md}, P)$
announce success

C. Generation

For each topic t, the program writes the progress line `Generating an essay on the topic:` t, records Unix seconds, and executes

`ollama run llama3.2` p_e

with a single argument

$p_e =$ `Write a five paragraph essay about the followi`

Standard output is captured; standard error is inherited so the Ollama spinner remains visible; standard input is `/dev/null`. Trailing newlines are stripped, reproducing Bash `$(...)`. The elapsed integer seconds are printed as `Essay generation took` n `seconds.` The quiz call is analogous with prefix

```
Write a quiz with 10 questions with the
answer key at the end about the following
text passage:
```

followed by the essay bytes. Arguments are passed by `exec` (`execvp`, `Command`, or `/usr/bin/env`), never by interpolating t into a shell string [19].

D. Page-break transform

Let $\text{Sed}_1(s)$ be: append a newline (as `echo` does); for each line, replace the first occurrence of `Here` with the eight-line fence

`\n\n\\newpage\n\nHere`

in the sense of BSD `sed` (literal backslash-`newpage`, real newlines); emit the line plus `LF`. Let Sed_2 do the same for `Answer Key`. The quiz body is

$$\text{StripNL}(\text{Sed}_2(\text{Sed}_1(q))).$$

StripNL deletes every trailing `LF`, matching command substitution. Because Sed_1 injects newlines, Sed_2 sees a different lineation—necessary to match the pipeline, not a single in-memory replace. The match is a byte substring: `Herein` is rewritten. Only the first hit per original line is rewritten, so `Here Here Here` yields one fence.

E. Markdown assembly

Each topic appends, in order: a `## Topic:` heading, a `### Text Passage` heading, the essay plus one newline, three blank-line newlines from `echo -e "\n\n"`, the transformed quiz, and a final newline. The header is written once. Empty L still writes the header and still calls Pandoc.

F. Failure policy

Unlike the prototype, a missing topics file, a non-zero Ollama status, or a non-zero Pandoc status exits 1 with a message on stderr. That is the production contract; happy-path stdout remains byte-compatible with Bash aside from the integer in the timing line.

IV. Implementation

Three codebases share the algorithm, not a library FFI.

a) C99: `p2q.c` is ISO C99 with `_POSIX_C_SOURCE` for `fork`, `execvp`, `pipe`, and `waitpid`. Pure ISO C99 cannot spawn processes [11]. Dynamic buffers double on growth. The CLI is 5 lines; the library is 538 lines of C plus a 42-line header.

b) Rust: `passage2quizzes` uses only `std`: `fs`, `process::Command`, `time::SystemTime`. Page-break search is byte-wise on UTF-8 views. 339 lines of Rust.

c) Swift: Foundation `Process` runs `/usr/bin/env` so `PATH` stubs work in tests. 217 lines of Swift. Extra process hops explain part of the stub-pipeline gap in Section V.

d) What was deliberately not cloned: Bash `$(cat)` globbing; `echo` treating a body of `-n` as a flag; continuing after a failed generator. Topics with quotation marks, dollars, and backticks are one `exec` argument.

V. Benchmarks and Evaluation

Experiments ran on Darwin 25.6, arm64, Apple clang 21, rustc 1.97.1, Swift 6.3.3, Ollama 0.32, Llama 3.2 3B. Pandoc was not installed; PDF bytes in automated tests come from a stub that copies the Markdown sidecar and writes a `FAKEPDF` marker. That isolates generator and formatter contracts [20].

A. Correctness

Eleven golden strings were captured from `echo | sed | sed` on this host, including start-of-line `Here`, `Answer Key` before `Here`, substring `Herein`, two input lines, and the empty string. Two Markdown goldens were captured from Bash `echo -e` assembly. All three ports match those files. Black-box tests assert six Ollama invocations for three topics, exact prompts, Pandoc argv `questions.md -o <pdf>`, and the success sentence. UX tests cover argc, missing files, empty topic lists, and special characters. The A/B harness runs C99, Rust, and Swift on the same fixtures and diffs `questions.md` plus stdout (timing integers normalized). The three binaries matched. Figure 3 is the poster rendering of that result.

Fig. 3. Three native ports, one artifact: A/B tests require byte-identical `questions.md`.

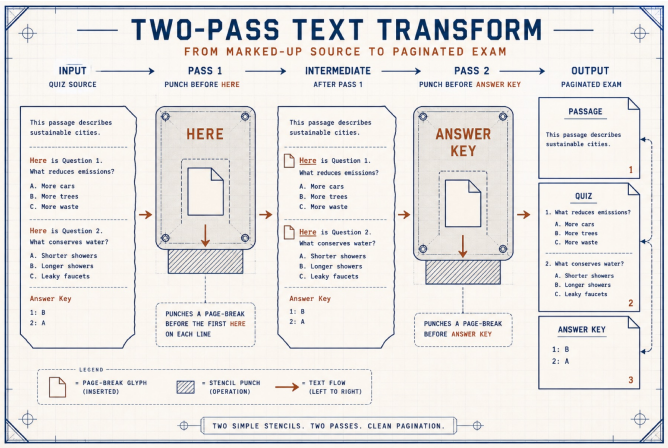

Fig. 4. Two-pass page-break: first `Here` per line, then first `Answer Key` per line, inserting `\newpage` fences for Pandoc.

B. Microbenchmarks

Table I reports C99 `-O2` means over 20 000 calls on a 7-line stub quiz. Table II reports five-run stub pipeline times for three topics (six model calls + Pandoc); the first run is discarded as warmup. Table III reports stripped release binaries.

C. Live model

With real `llama3.2` and a Pandoc stub, one topic (Photosynthesis) took 8 s for the essay and 4 s for the quiz. Local transforms are $< 10^{-4}$ of that budget. Figure 6 shows the intended packet; Figure 7 summarizes the test stations used to keep the three ports honest.

D. Threats

Goldens are host-specific (BSD `sed`, Bash `echo -e`). GNU `sed` would not be a valid oracle without regeneration. Live quality of essays and items is out of scope; we test the *harness*, not pedagogy [3]. Swift times include Foundation process setup and should not be read as a language shootout [15].

TABLE I
IN-PROCESS C99 TRANSFORMS (20 K ITERATIONS, `-O2`).

Operation	Mean time
Page-break transform	1.16 μs
Assemble 3-topic Markdown	0.38 μs

TABLE II
THREE-TOPIC STUB PIPELINE (MEDIAN OF 4 POST-WARMUP RUNS).

Port	Median wall time	vs. C99
C99 `-O2`	60 ms	1.0×
Rust release	60 ms	1.0×
Swift release	540 ms	9.0×

VI. FUTURE WORK

a) Item schema: Pin quiz output to JSON (identifier, stem, choices, key) so pagination does not depend on the model emitting the substrings `Here` and `Answer Key`.

b) Deterministic decoding: Pass Ollama sampling parameters and a seed when the engine supports them, enabling live A/B of text, not only of the harness.

c) True PDF in CI: Vendor a Pandoc + pdf engine or a pure-Rust Markdown-to-PDF path so layout tests can assert page count.

d) Windows: Replace POSIX spawn with `CreateProcess`; keep argv fidelity.

e) Human review queue: Bind generated packets to an editor before print, addressing academic integrity concerns [2], [3].

f) Shared spec crate: Generate C headers, Swift, and Rust from one grammar of the Markdown layout to prevent drift.

VII. CONCLUSION

Passage2Quizzes shows that a 33-line LLM shell pipeline can be lifted into C99, Rust, and Swift without giving up the bytes instructors already print. The hard part was not calling Ollama; it was cloning BSD `sed` lineation, Bash newline stripping, and `echo -e` Markdown spacing, then locking those clones with goldens and a three-language A/B gate. Once that gate is green, performance is irrelevant beside Llama 3.2: twelve seconds of generation dwarf sixty milliseconds of port overhead. The practical recommendation is: treat the shell script as an oracle, port the process graph into the language you ship, and never let the model call become an excuse to skip characterization tests.

ACKNOWLEDGMENT

The original Bash tool and product screenshots are confidential materials of Domino Data Systems.

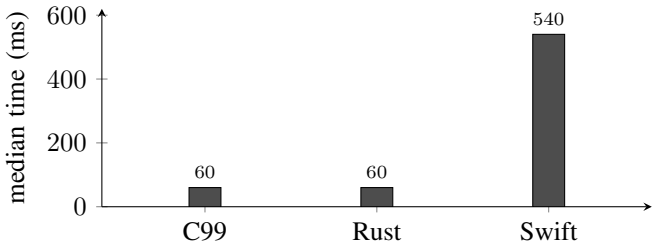

Fig. 5. Stub-pipeline wall time. Swift pays extra `Process`/env spawns; all three are noise compared with a live LLM call.

TABLE III
STRIPPED RELEASE BINARY SIZE (ARM64-APPLE-DARWIN).

Port	Bytes
C99	35 144
Swift	76 616
Rust	412 840

REFERENCES

[1] M. Zaharia *et al.*, "The shift from models to compound AI systems," *Berkeley Artificial Intelligence Research Blog*, 2024.

[2] E. Kasneci, K. Sessler, S. Küchemann, M. Bannert, D. Dementieva, F. Fischer, U. Gasser, G. Groh, S. Günnemann, E. Hüllermeier *et al.*, "ChatGPT for good? On opportunities and challenges of large language models for education," *Learning and Individual Differences*, vol. 103, p. 102274, 2023.

[3] D. R. E. Cotton, P. A. Cotton, and J. R. Shipway, "Chatting and cheating: Ensuring academic integrity in the era of ChatGPT," *Innovations in Education and Teaching International*, vol. 61, no. 2, pp. 228–239, 2024.

[4] J. MacFarlane, *Pandoc User's Guide*, 2024, https://pandoc.org/MANUAL.html.

[5] J. Gruber, "Markdown: Daring Fireball," 2004, https://daringfireball.net/projects/markdown/.

[6] "sed — stream editor," POSIX.1-2017 / The Open Group, 2017, iEEE Std 1003.1-2017.

[7] A. Dubey *et al.*, "The Llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[8] Meta AI, "Llama 3.2: Revolutionizing edge AI and vision with open, customizable models," https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/, 2024.

[9] Ollama, "Get up and running with large language models locally," https://ollama.com, 2024.

[10] X. Wang *et al.*, "Automatic question generation for educational assessment: A survey," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024, survey of neural and LLM-based item generation.

[11] *ISO/IEC 9899:1999 Programming Languages — C*. Geneva, Switzerland: International Organization for Standardization, 1999.

[12] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, 2nd ed. Englewood Cliffs, NJ: Prentice Hall, 1988.

[13] S. Klabnik and C. Nichols, *The Rust Programming Language*. San Francisco, CA: No Starch Press, 2019.

[14] Apple Inc., "The Swift programming language," https://docs.swift.org/swift-book/, 2024.

[15] L. Prechelt, "An empirical comparison of seven programming languages," *Computer*, vol. 33, no. 10, pp. 23–29, 2000.

[16] B. Ray, D. Posnett, V. Filkov, and P. Devanbu, "A large scale study of programming languages and code quality in GitHub," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 155–165.

[17] M. Feathers, "Working effectively with legacy code," in *Prentice Hall PTR*. Prentice Hall, 2004.

[18] G. J. Myers, C. Sandler, and T. Badgett, *The Art of Software Testing*, 3rd ed. Wiley, 2011.

[19] *IEEE Std 1003.1-2004, POSIX.1*. IEEE / The Open Group, 2004.

[20] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *Future of Software Engineering (FOSE '07)*. IEEE, 2007, pp. 85–103.

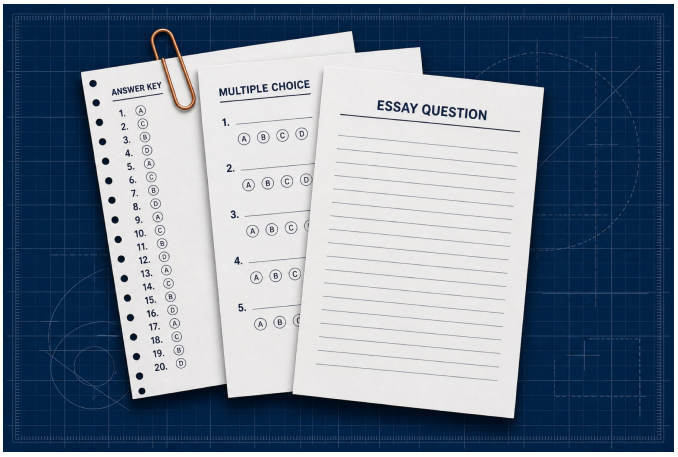

Fig. 6. Deliverable packet: passage, quiz, answer key, paginated for print.

Fig. 7. Evaluation stations: unit, black-box, UX, A/B, and live.