

From course materials to targeted practice exams

JEFF is an exam-forecasting and practice application built around Google's Gemini models. A student supplies course materials, such as previous assessments, notes, and review sheets. The system analyzes what those documents reveal about topic coverage and assessment style, proposes a blueprint for a likely next exam, and generates an original practice exam with solutions. Students can take that exam, review it, export it, or share a link.

The question that started it

A casual LinkedIn post suggesting that Gemini could anticipate exam questions from course material gave me the idea. I wanted to test it in a tool I would actually use. My goal was to generate the most relevant kinds of practice questions, not simply ask a chatbot for a generic quiz. The post was a starting point for an experiment, not evidence that exact future questions could be predicted.

Turning a model into a usable workflow

I built the application around the model: document handling, instructions, formatting, mathematical rendering, practice, export, and sharing. On 29 April 2026, I documented lessons from studying a publicly circulated Claude Code prompt leak, then refactored JEFF. I separated a short shared base prompt from task-specific instructions, established instruction priorities, and added schema-constrained JSON output with validation. These changes made the workflow more explicit and inspectable; they did not establish forecasting accuracy.

The current workflow goes further: analysis produces a blueprint with topics, points, format, difficulty, and time for each question. It distinguishes exams, solutions, notes, and review sheets rather than treating them as equivalent evidence. A separate generation step follows the plan. Checks examine structure and excessive source reuse, with repair and solution-verification stages. These later capabilities should not be confused with what the April refactor alone implemented.

What prediction means here

JEFF estimates likely emphasis and question style from the evidence supplied. It cannot know an instructor's unseen exam, and I have not established a reliable exact-question prediction rate. A plausible-looking question or a passing formatting test does not establish forecasting accuracy. The later evaluation pages explain what was tested and why stronger performance claims would be premature.

Development references: 29 April 2026 commits 18120c2, implementation lessons, and 51a2a3c, Gemini service refactor. The following pages examine use, generation, source reuse, rendering, and evaluation.

Course material to practice

I built JEFF because I wanted more active practice from my course materials. Before our physics midterm, classmates kept asking me for more tests. Sending imperfect PDF exports through Discord or Google Drive made distributing that practice cumbersome. I added built-in share links so classmates could open the questions directly, pass the link along, or export their own copy.

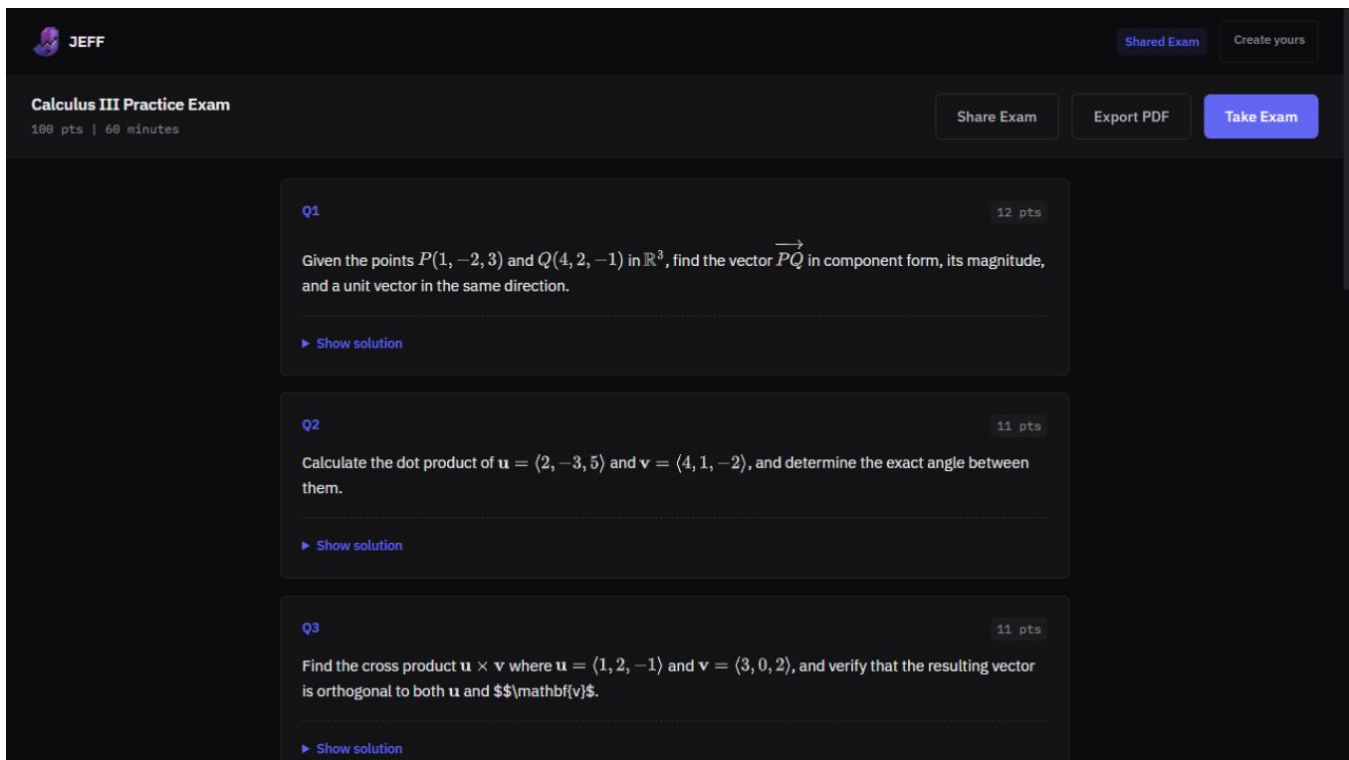


Figure 6. Actual user-supplied shared exam, captured 18 October 2026 from the deployed application. The visible Question 3 markup defect is retained. This is not a locally generated exam or a learning-outcome measurement.

What I built around the model

I built the workflow connecting source upload, analysis, generation, validation, practice, review, and export, using Next.js, React, Gemini, and Firebase. I did not train the underlying model. Copied questions, broken notation, and missing answers motivated the checks examined in this chapter.

A product decision driven by use

Sharing became part of the product rather than a file-distribution task I handled separately. JEFF remains the project I use most in daily life. That experience explains the feature, but does not establish learning gains or measured growth. The following pages examine generation, source overlap, rendering, and the limits exposed by development pilots.

From exam blueprint to model request

JEFF uses an exam blueprint to specify the structure of a generated assessment. Rather than asking only for questions on a topic, the generation instructions carry question count, point totals, timing, and slot-level requirements into the model request.

Generation instructions

```

79 Rules:
80 - Generate exactly ${formatQuestionCount(blueprint.topLevelQuestionCount)}.
81 - Set totalPoints to exactly ${blueprint.totalPoints}.
82 - Set timeLimit to exactly "${blueprint.timeLimitMinutes} minutes".
83 - Match every blueprint slot's number, points, topics, type, subpart count, difficulty, and style
    pattern.
84 - Question numbers must be sequential from 1.
85 - Write new questions. Do not copy or lightly paraphrase uploaded stems.
86 - Solutions must be complete, independently correct, and consistent with the stem.
87 - If a question has lettered subparts, solve every subpart.

```

lib/gemini/prompts.ts, lines 79-87. Local source view captured 18 October 2026; HEAD f002fda6.

Excerpt from buildGenerationPrompt. Interpolated values come from the exam blueprint; the rules express the requested structure and source-use constraints.

The API connection

```

149 const response = await getClient().models.generateContent({
150   model,
151   contents: [{ role: "user", parts: [{ text: prompt }, ...fileParts] }],
152   config: {
153     responseMimeType: "application/json",
154     responseSchema: opts.responseSchema,
155     thinkingConfig: { thinkingLevel: opts.thinkingLevel ?? DEFAULT_THINKING_LEVEL },
156     temperature: opts.temperature ?? 0.2,
157     maxOutputTokens: opts.maxOutputTokens ?? 12000,

```

lib/gemini/client.ts, lines 149-157. Local source view captured 18 October 2026; HEAD f002fda6.

The generation routine builds this prompt and invokes the shared Gemini client with EXAM_SCHEMA. The client sends the prompt as user content, alongside any supplied file parts, and requests JSON matching the response schema. These are generation instructions, not a separate API systemInstruction field.

Why the prompt is only one layer

A schema constrains the response format; it cannot establish originality or mathematical correctness. JEFF subsequently validates the candidate exam, checks source overlap, and can request repairs. Its solution-verification path works in batches and marks unresolved omissions as disputed. The next pages show why those checks were needed and where the evidence remains limited.

Source overlap detection

A generated question may reuse source wording while appearing to be new practice. I implemented a phrase-overlap check using eight-word sequences. The score is the proportion of candidate sequences that also appear in the source.

```
15 export function sourceOverlapRate(stem: string, sourceText: string, size = 8): number {
16   const source = new Set(grams(sourceText, size));
17   const candidate = grams(stem, size);
18   if (candidate.length === 0) return 0;
19   return candidate.filter((gram) => source.has(gram)).length / candidate.length;
  ... intervening source lines omitted ...
24   return exam.questions.flatMap((question, index) => {
25     const overlap = sourceOverlapRate(question.question, sourceText);
26     return overlap >= 0.35
27       ? [{
28         code: "SOURCE_STEM_OVERLAP",
29         severity: "error" as const,
30         path: `questions[${index}].question`,
31         message: `Question ${question.number} has ${Math.round(overlap * 100)}% phrase overlap
  with uploaded sources.`,
32         repairable: true,
33       }]
34       : [];
35   });
```

Source screenshot of lib/jeff/originality.ts, lines 15-19, 24-35. Omitted lines are marked.

Listing 5. Overlap at or above 0.35 produces SOURCE_STEM_OVERLAP with repairable set to true. The detector does not accept a high-overlap stem as an ordinary successful result.

Test cases

Case in originality.test.ts	Expected assertion
Source text compared with itself	Overlap rate equals 1
Related but newly worded inventory question	Overlap rate equals 0
Copied question passed to inspectOriginality	Issue code is SOURCE_STEM_OVERLAP

The test checks the issue code but does not separately assert the repairable flag. That flag is visible in the implementation above. Short stems with fewer than eight words produce no candidate sequences and return zero.

This heuristic can miss paraphrases and may flag repeated technical language. It is a control against long verbatim reuse, not a proof of originality or factual correctness.

Currency and mathematical rendering

A question containing two prices exposed a rendering error: MathJax interpreted the dollar signs around ordinary currency as mathematical delimiters. The text between the prices could therefore be rendered as an equation.

```

17 describe("protectCurrencyDollars", () => {
18   it("keeps multiple accounting prices out of MathJax", () => {
19     const text = "Beginning inventory was $50 each. The next purchase was $60 each.";
20     expect(protectCurrencyDollars(text)).toBe(
21       "Beginning inventory was \\$50 each. The next purchase was \\$60 each.",
22     );
23   });
  ... intervening source lines omitted ...
35   it("does not change genuine inline or display equations", () => {
36     expect(protectCurrencyDollars("Use $x = 50$ and $$y = 60$$")).toBe(
37       "Use $x = 50$ and $$y = 60$$.",
38     );
39     expect(protectCurrencyDollars("Compute $50 + 60 = 110$")).toBe(
40       "Compute $50 + 60 = 110$.",
41     );

```

Source screenshot of lib/math-formatting.test.ts, lines 17-23, 35-41. Omitted lines are marked.

Listing 6. Regression fixtures distinguish two accounting prices from genuine mathematical expressions. The expected strings protect currency without escaping equation delimiters.

Implementation change

Browser rendering and PDF export share text segmentation. On 21 September 2026, a recorded regression run failed on an eigenvalue list: the currency safeguard misread its comma, and the word "and" remained in math mode. After the rule was narrowed and prose separated, all eleven formatting tests passed.

Commit 42a25cd preserves the change. This sequence comes from contemporaneous development-session command output, not a reconstructed physical trial.

Verification scope

On 16 October 2026, a focused run of the current working copy passed all sixteen mathematical-text tests and both source-overlap tests. The mathematical-text fixtures cover prices, punctuation, escaped currency, genuine equations, prose around equations, and bare LaTeX commands.

A live check on 16 October loaded a saved Calculus III exam and exported seven PDF pages. Question 3 retained malformed mathematical markup in both views; the two inspected PDF pages also had oversized inline equations. The deployed revision was not identified. These observations limit the rendering claim despite the passing local tests.

Implementation references: lib/math-formatting.ts; app/components/MathRenderer.tsx; lib/pdf.ts. Historical test record: Codex development session, 21 September 2026, commit 42a25cd.

Evaluation and development pilots

I used an early Physics II comparison to investigate generated-question quality. I subsequently withdrew its ranking because the judging path was not sufficiently independent to support a comparative performance claim.

Later development studies

The replacement work progressed beyond a protocol. Benchmark 003 reports document a Linear Algebra pilot on 11 August 2026 and a Chemistry replication on 12 August. These studies compared generated exams with held-out assessments and retained failed attempts in their accounting.

Recorded study	Successful exams	Grading described in the report
Linear Algebra	One per candidate	GPT, Gemini, and one human pass
Chemistry	Three per candidate	Four passes across two model families

Table 3. Counts reported in benchmarks/v3/pilot-001.md and pilot-002.md. These are development-study records, not a new independent reanalysis of the raw grades.

A failure that changed the implementation

The first Linear Algebra attempt failed after the solution verifier omitted question 3 and the backup model had no available quota. The report records a subsequent successful attempt after batching was introduced. Current code verifies non-multiple-choice questions in groups of four, retries missing question numbers once, and marks any remaining omission as disputed.

Limits of the evidence

The Linear Algebra graders disagreed substantially; its human grader could not independently assess solution correctness. Chemistry retained three JEFF infrastructure failures outside the six-exam candidate packet. The equal-budget comparison remained unfinished, and public held-out exams may have appeared in model training data.

I use these pilots to identify implementation and evaluation weaknesses. They do not establish improved student learning or general superiority. The portfolio omits comparative scores and the withdrawn ranking.

References: Benchmark 003 pilot reports and protocol.json; lib/gemini/client.ts, verification routine. The repository designates the OCW-based suite as internal research. No course PDFs or generated exam text are reproduced here.