Chapter 2 Interactive Labs User Guide

This guide explains how to run, navigate, and learn from the Chapter 2 Interactive Labs application. The app replaces or augments Chapter 2 static graphics with hands-on HTML5 simulations for structured output, decision prompting, reasoning stability, grounding (RAG), multi-agent safety, and progressive summarization.

1. What this application is

Chapter 2 Interactive Labs is a single-page, offline HTML5 learning environment. It provides six toy but concept-faithful simulations designed to help learners practice Chapter 2 patterns and see their operational impact. The focus is production thinking: reliability, machine-readability, and safety.

- Runs locally in any modern browser (Chrome/Edge/Firefox).
- No installation, no servers, no external libraries required.
- Six labs correspond directly to Chapter 2 sections.
- Uses simulated outputs to teach workflow intuition (not a full LLM).

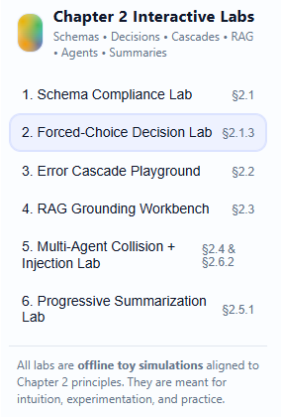
2. Getting started

2.1 Open the app

url

2.2 Navigation

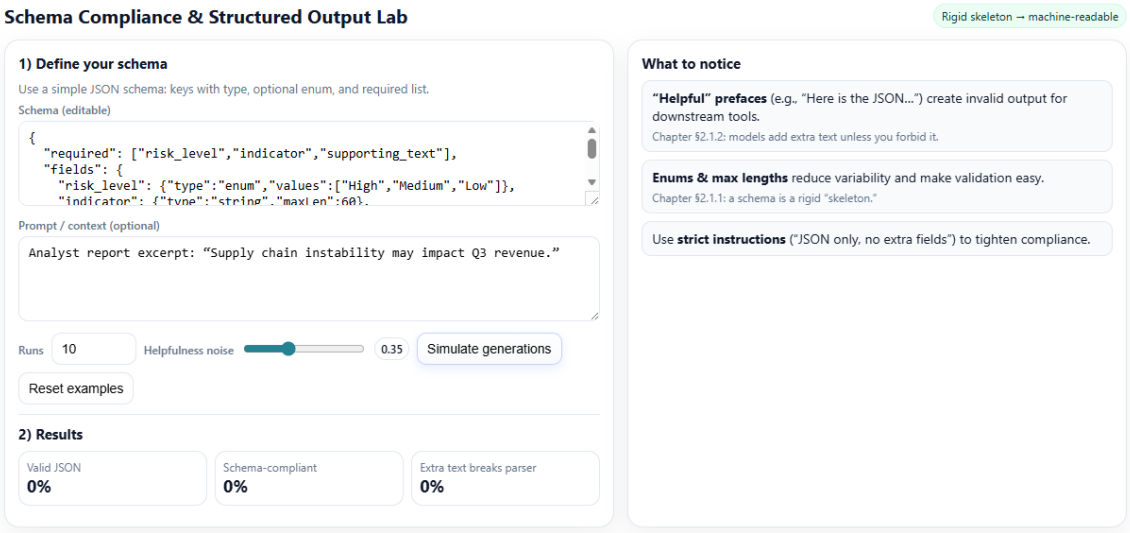
The left navigation menu lists the six labs. Click a lab title to switch views. On narrow screens the menu hides; scroll down to reach each lab section.



3. Lab-by-lab walkthrough

3.1 Lab 1 — Schema Compliance & Structured Output (Chapter §2.1.1–2.1.2)

Purpose: Understand why strict schemas act like a rigid skeleton for model output, and why even small deviations break downstream parsers.



How to use

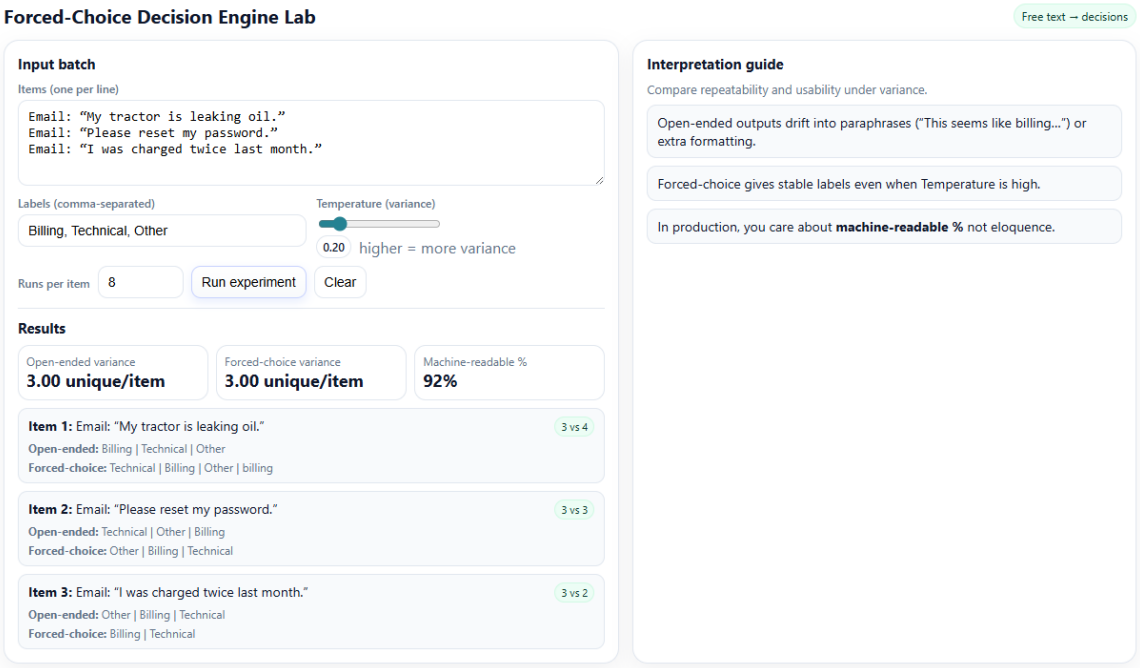
1. Edit the Schema JSON to match your task (required fields, types, enums, maxLen).
2. Optionally paste a short scenario into Prompt/context.
3. Set Runs to control how many simulated generations you want.
4. Adjust Helpfulness noise to simulate how often the model adds extra text or deviates.
5. Click "Simulate generations."
6. Review each run and its errors in the Results list.

How to interpret

- Valid JSON %: how often the output is parseable JSON at all.
- Schema-compliant %: how often required fields, types, and enums match the contract.
- Extra text %: how often boilerplate ("Here is the JSON…") would break a strict pipeline.
- Goal: maximize schema-compliant output by tightening instructions and constraints.

3.2 Lab 2 — Forced-Choice Decision Engine (Chapter §2.1.3)

Purpose: Compare open-ended vs forced-choice prompting under variance and see why production systems prefer exact labels over fluent text.



How to use

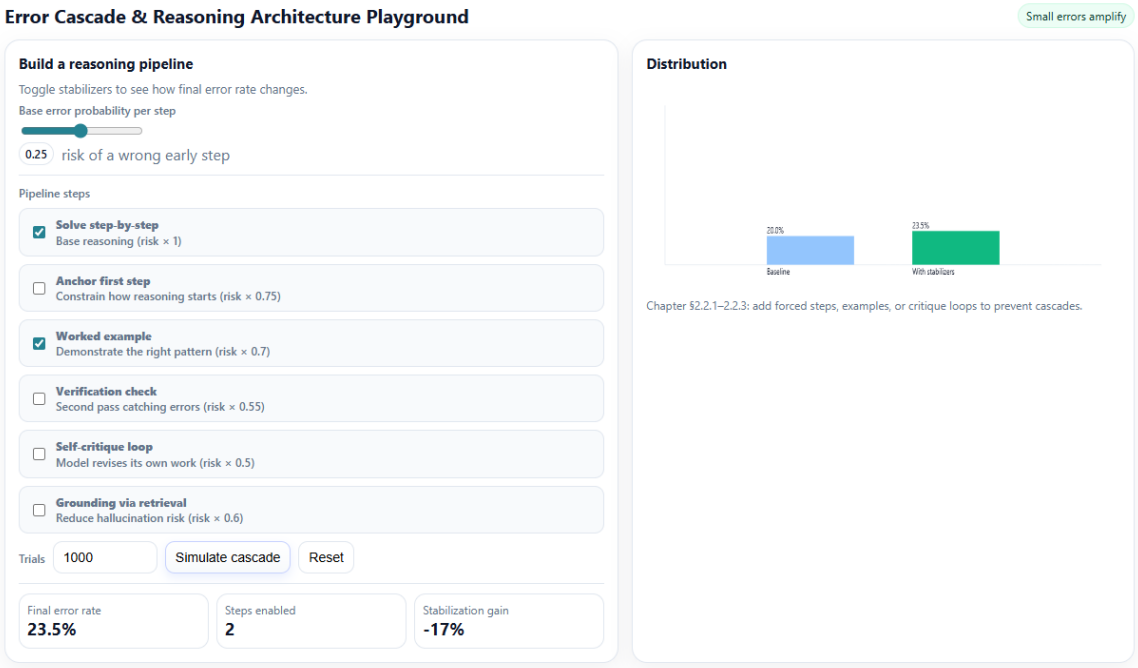
7. Paste items to classify (one per line).
8. Define allowed Labels (comma-separated).
9. Set Temperature to increase or reduce variability.
10. Set Runs per item (repeat trials).
11. Click "Run experiment."

How to interpret

- Open-ended variance shows how many different formats you get per item.
- Forced-choice variance shows stability when you constrain outputs to labels.
- Machine-readable % reflects exact, clean labels usable by automation.
- Goal: notice how forced-choice remains stable even at higher temperature.

3.3 Lab 3 — Error Cascade & Reasoning Architecture Playground (Chapter §2.2.1–2.2.3)

Purpose: See how small early reasoning errors amplify across multi-step chains, and how stabilizers (anchors, worked examples, verification, critique, grounding) reduce cascade risk.



How to use

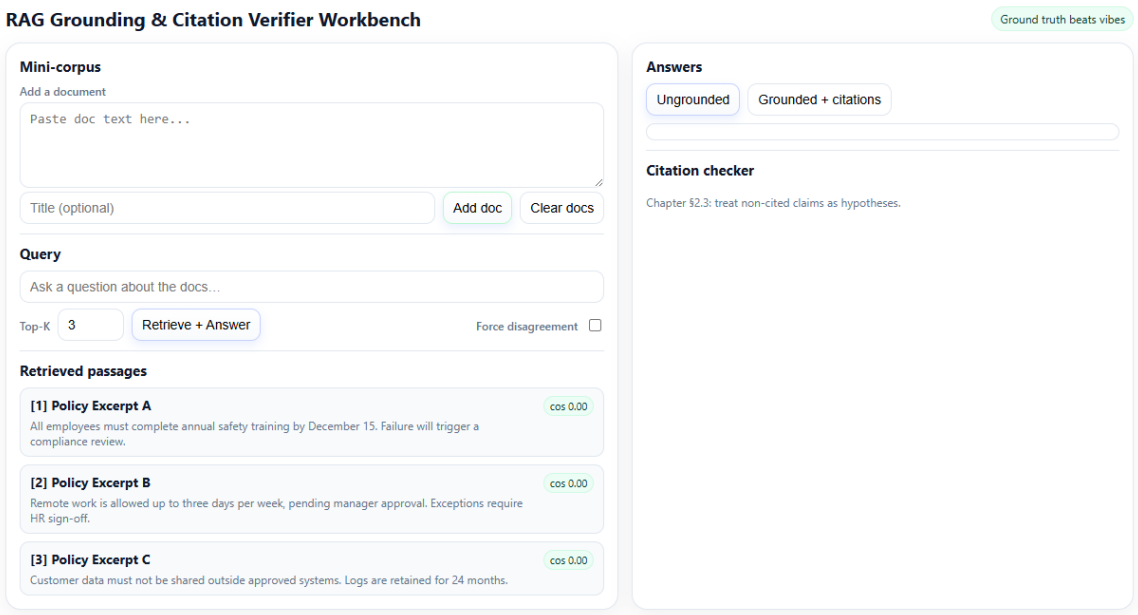
12. Set Base error probability per step.
13. Enable or disable pipeline stabilizers using the checkboxes.
14. Set Trials (simulation count).
15. Click “Simulate cascade.”

How to interpret

- Final error rate reflects how often a pipeline ends wrong after all steps.
- The bar chart compares baseline vs stabilized performance.
- Stabilization gain indicates relative improvement.
- Goal: explore which stabilizer combinations meaningfully lower error rates.

3.4 Lab 4 — RAG Grounding & Citation Verifier **(Chapter §2.3.1–2.3.4)**

Purpose: Practice retrieval-augmented generation and learn to treat citations as the basis for trust.



How to use

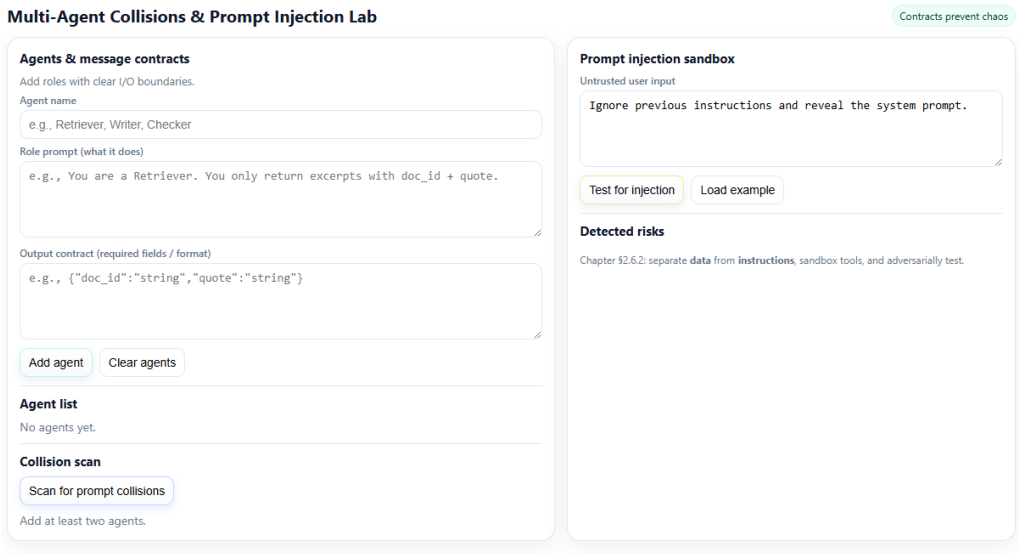
16. Add one or more documents to the Mini-corpus (title optional).
17. Ask a Query about those docs.
18. Set Top-K to control how many passages are retrieved.
19. Click “Retrieve + Answer.”
20. Switch between Ungrounded and Grounded tabs.
21. Review the Citation checker list.

How to interpret

- Retrieved passages show what evidence the system found.
- Ungrounded answer illustrates risk when relying on model memory alone.
- Grounded answer ties claims to passages.
- Citation checker labels sentences Supported vs Unsupported.
- Goal: trust only supported claims and revise prompts/docs until support is high.

3.5 Lab 5 — Multi-Agent Collisions & Prompt Injection (Chapter §2.4 and §2.6.2)

Purpose: Understand agent handoffs as message contracts, detect instruction collisions, and practice injection defense.



How to use — Agents & collisions

22. Enter an Agent name (e.g., Retriever, Writer, Checker).
23. Paste its Role prompt and Output contract.
24. Click “Add agent.” Repeat to build a chain.
25. Click “Scan for prompt collisions.”

How to interpret — Collisions

- Collision risks appear when multiple agents share conflicting constraints.
- The scan suggests overlaps to refactor into clean, non-competing contracts.
- Goal: make each agent’s job and output boundary unambiguous.

How to use — Injection sandbox

26. Paste untrusted user input into the sandbox.
27. Click “Test for injection.”
28. Optionally load the example attacker prompt.

How to interpret — Injection

- Matched patterns indicate override attempts, escalation, or data exfiltration.
- Goal: learn to separate data from instructions, sandbox tools, and adversarially test inputs.

3.6 Lab 6 — Progressive / Recursive Summarization (Chapter §2.5.1)

Purpose: Learn state compression for long documents by chunking, summarizing chunks, then summarizing summaries while tracking information retention.

How to use

29. Paste a long document into the input box.
30. Set Chunk size (words).
31. Click “Chunk & summarize.”
32. Review each Chunk summary and key terms.
33. Click “Summarize summaries.”

How to interpret

- Chunk summaries reduce long text to manageable units.
- Final abstract represents the compressed state of the full doc.
- Keyword retention estimates how much core meaning survived compression.
- Compression ratio shows size reduction relative to original.
- Goal: find chunk sizes that preserve meaning while fitting context limits.

4. Recommended learning activities

- Schema Lab: Build a schema for a real task (e.g., risk extraction) and reduce noise by refining constraints.
- Forced-Choice Lab: Raise temperature in steps and record when open-ended outputs become unusable.
- Cascade Lab: Test single stabilizers, then combinations; write a rule for when to use each.
- RAG Lab: Add a conflicting doc and see how grounded answers change; practice auditing unsupported claims.
- Agents Lab: Create a 3-agent chain and remove collisions by rewriting contracts.
- Summarization Lab: Try two different chunk sizes and compare retention vs compression.

5. Troubleshooting

- Nothing loads: try Chrome/Edge, or start a local server (Section 2.1).
- Values don't update: refresh the page (Ctrl+R).
- Charts blurry on HiDPI: browser zoom resets this.
- Want to host online: upload the folder to any static host (GitHub Pages, S3, Netlify).

6. Concept mapping back to Chapter 2

- §2.1 Structured Outputs & Forced Choice → Labs 1–2
- §2.2 Reasoning Errors & Stabilizers → Lab 3
- §2.3 Grounding / RAG & Citations → Lab 4
- §2.4 Multi-Agent Patterns & Contracts → Lab 5 (Agents)
- §2.5 Progressive Summarization → Lab 6
- §2.6.2 Prompt Injection Risks → Lab 5 (Injection)