

LESSONWORLD: AGENTIC ENVIRONMENT FOR INTERACTIVE EDUCATIONAL CONTENT GENERATION

Anonymous authors

Paper under double-blind review

Demos, Code and Data : LessonWorld.fly.dev

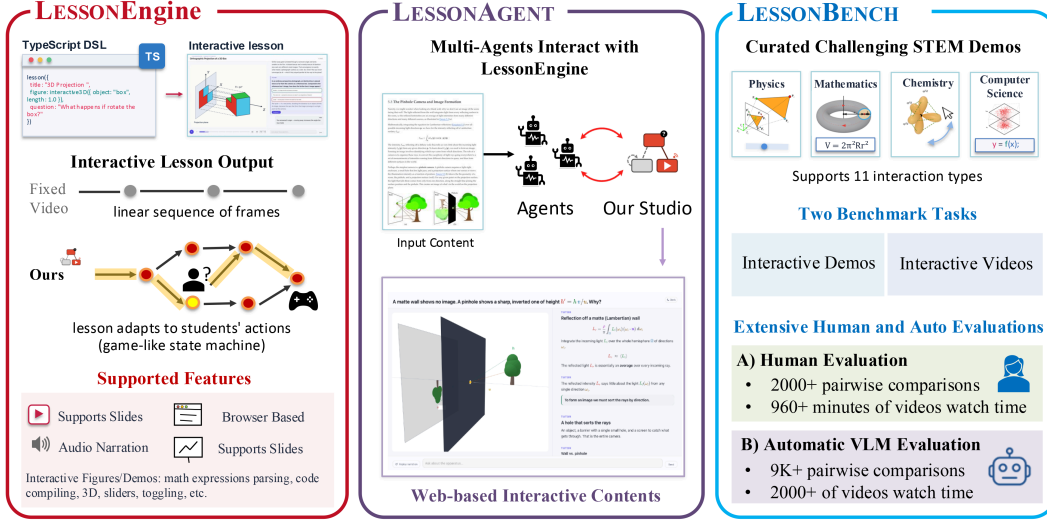


Figure 1: **LessonWorld Overview**. Unlike conventional educational content generation that produces fixed artifacts, LESSONENGINE represents lessons as interactive, stateful environments that respond to learner actions. LESSONAGENT uses a distributed agentic pipeline to transform instructional resources into these executable lessons through iterative generation and refinement. LESSONBENCH evaluates educational content generation across challenging STEM figures, demonstrations, videos, and interactive lessons using automated VLM-based and extensive human evaluation.

ABSTRACT

Agentic AI systems are increasingly used to generate educational content such as figures, slides, and videos, yet most systems still treat lessons as static artifacts that learners passively consume. We introduce **LESSONWORLD**, an environment for human-AI collaborative authoring of interactive, executable educational content. LESSONWORLD represents lessons as dynamic state machines in which students can manipulate 3D objects, write mathematical expressions, interact with visualizations, and ask questions, with the lesson pathway adapting to their actions. LESSONWORLD consists of three components: (1) **LESSONENGINE**, a TypeScript-based authoring platform with a stateful runtime and programmatic primitives for constructing and modifying interactive lessons; (2) **LESSONAGENT**, a distributed agentic pipeline that grounds lessons in instructional resources, generates interactive figures, composes lessons, and iteratively validates and adapts them; and (3) **LESSONBENCH**, a benchmark spanning challenging scientific demonstrations and spatially complex figures, with automated VLM-based pairwise evaluation and human evaluation of over 2,000 figure/demo pairs and 960 minutes of educational videos and interactive lessons. Across benchmark tasks, LESSONAGENT outperforms existing methods on figure generation and interactive content generation in both automated and human evaluations. Together, LESSONWORLD establishes interactive, executable lessons as a new paradigm for educational content creation. For demos and code please visit: LessonWorld.fly.dev.

1 INTRODUCTION

Over the past two years, generative AI has increasingly been used to produce educational content across modalities. For example, agentic systems can write Manim code to generate visually compelling explanatory videos in the style of 3Blue1Brown (Chen et al., 2026b; Ku et al., 2025; Jain et al., 2025; Joshi et al., 2026). Other work has explored generating slide decks (Zheng et al., 2025; 2026), narrated presentation videos with synthesized presenters (Zhu et al., 2025), and scientific figures and interactive demonstrations (Shao et al., 2026; Zhu et al., 2026b; Ji et al., 2026). Despite this progress, most existing systems ultimately generate a fixed sequence of educational content that is passively consumed after generation.

Real-world instruction is more responsive: an instructor can observe a learner’s responses, provide a different explanation or example, and adapt the next instructional step accordingly. Supporting this kind of interaction requires a lesson representation that can maintain state and be modified during execution. Yet current generation systems typically terminate in a finished artifact, while interactive educational environments largely rely on content and behaviors authored in advance.

Generating such interactive lessons from scratch presents another challenge. An agent must repeatedly implement the same runtime machinery, including state management, event handling, branching, synchronization, and session persistence. This makes generations long, difficult to coordinate, and hard to distribute across agents. These limitations motivate a reusable abstraction that separates lesson-specific content from the runtime machinery needed to make it interactive and adaptive.

We introduce  **LESSONWORLD**, an environment for authoring and running interactive educational content. Its central component, **LESSONENGINE**, is a TypeScript library and domain-specific language that serves as an authoring substrate for interactive lessons, analogous to Manim for Manimator (Jain et al., 2025) or PowerPoint for PPTAgent (Zheng et al., 2025). **LESSONENGINE** represents lessons as editable state machines, where nodes encode instructional content and transitions respond to learner and tutor events. By providing reusable, typed primitives, it eliminates the need to repeatedly implement lesson runtime machinery from scratch.

LESSONAGENT uses **LESSONENGINE** to construct lessons from instructional material, generating interactive figures and composing them into larger instructional sequences. Its typed, modular representation also enables generation to be distributed across smaller agents and composed afterward. In our measurements, authoring with **LESSONENGINE** requires up to $2\times$ fewer output tokens than generating the corresponding interactive implementation directly.

LESSONBENCH. To address the lack of dedicated benchmarks for educational interactive content, we introduce **LESSONBENCH**, a benchmark suite for generated figures and interactive lessons across four STEM domains. We evaluate generations along five dimensions: *interactivity*, *conceptual accuracy*, *pedagogical effectiveness*, *faithfulness*, and *usefulness*, using both human evaluators and VLM-based judge. Results show that existing methods struggle to consistently generate effective educational materials, while our approach substantially outperforms leading baselines across both tasks of educational interactive lesson and figure generation.

Our contributions are summarized as follows:

1. **LESSONENGINE:** our authoring platform implemented as a TypeScript library for constructing interactive, game-like educational lessons. It provides the stateful runtime, typed content representation, and authoring primitives needed to create and modify lessons programmatically.
2. **LESSONBENCH:** a benchmark for evaluating two tasks in interactive educational authoring, spanning challenging scientific demonstrations and spatially complex figures. We introduce automated and human evaluation protocols, including more than 2,000 pairwise comparisons with VLM judges and human participants, comprising 2050 pairwise figure comparisons and rubric-based evaluation of 120 educational videos and interactive lessons.
3. **LESSONAGENT:** an agentic authoring pipeline built around **LESSONENGINE**. The system decomposes lesson creation into specialized stages for grounding in instructional resources, generating interactive figures, composing lessons, and validating and adapting the resulting environment. **LESSONAGENT** achieves the best performance on across both tasks for automated and human evaluations of our benchmark.

2 RELATED WORK

Video Generation. Recent agentic systems increasingly generate educational videos using executable code as an intermediate representation. Code2Video (Chen et al., 2026b), TheoremExplainAgent (Ku et al., 2025), Manimator (Jain et al., 2025), and LLM2Manim (Joshi et al., 2026) use Manim-based representations for mathematical and educational animations, often with planning, critique, and refinement. Paper2Video (Zhu et al., 2025) extends this paradigm to lecture-length videos combining slides and narration. These systems demonstrate that agentic pipelines can produce structured instructional media, but the resulting artifact remains fixed after generation rather than continuing to respond to learner interaction.

Slide Generation. PPTAgent (Zheng et al., 2025) decomposes presentation creation into planning and slide-level construction, while DeepPresenter (Zheng et al., 2026) extends this paradigm toward richer web-based presentations. Their emphasis is on generating and revising the presentation itself rather than maintaining an executable lesson representation during use.

Interactive Figure and Visualization Generation. Recent work moves toward structured and editable visual representations. AutomaTikZ (Belouadi et al., 2024) uses code as an intermediate representation for scientific figures, while AutoFigure (Zhu et al., 2026b), LiveFigure (Shao et al., 2026), PaperBanana (Zhu et al., 2026a), and Paper2Figure (Han et al., 2026) develop increasingly explicit generation and editing pipelines. EduVisAgent (Ji et al., 2026) is particularly relevant to education, generating interactive HTML visualizations for pedagogical explanation. These systems move closer to executable media, but primarily target individual visualizations rather than complete lessons with persistent learner state and coordinated instructional flow.

Runtime-Interactive Environments. Educational platforms such as PhET and GeoGebra support direct learner interaction, while systems such as PaSSAGE (Thue et al., 2007), the drama manager of Yu and Riedl (Yu & Riedl, 2013), and *Left 4 Dead’s* AI Director (Booth, 2009) adapt experiences to user behavior during execution. These systems show the value of stateful runtime interaction, but their interaction logic and possible behaviors are largely specified in advance.

LESSONWORLD in Context. Prior systems typically cover only part of the design space: generation methods produce videos, slides, or visualizations, while interactive systems support runtime interaction within more constrained representations. LESSONWORLD combines multimodal generation with a stateful lesson representation that remains interactive during use and can be modified by agents at runtime. This allows generated content to function as part of a larger executable lesson rather than as an isolated artifact. Table 1 summarizes these differences across output modality, runtime directability, interaction, and evaluation.

Method	Venue	Backend	Output Modality				Properties		Evaluation
			Video	Figure	Web	Slides	RT	Interact.	
Code2Video (Chen et al., 2026b)	ICML’26	Manim (Python)	✓	×	×	×	×	×	VLM + Human
LiveFigure (Shao et al., 2026)	ICML’26	PPTX	×	✓	×	×	×	×	Human
PaperBanana (Zhu et al., 2026a)	ICML’26	Image model	×	✓	×	×	×	×	VLM
Paper2Figure (Han et al., 2026)	CVPR’26	FigScript	×	✓	✓	×	×	×	VLM (Rubric)
AutoFigure (Zhu et al., 2026b)	ICLR’26	SVG / image model	×	✓	×	×	×	×	VLM + Human
EduVisAgent (Ji et al., 2026)	ICLR’26	HTML	×	✓	✓	×	×	✓	VLM (Rubric)
DeepPresenter (Zheng et al., 2026)	ACL’26 Find.	HTML/CSS	×	×	×	✓	×	×	VLM + Rule-based
Manimator (Jain et al., 2025)	ICML’25	Manim (Python)	✓	×	×	×	×	×	LLM/VLM + Human
TheoremExplainAgent (Ku et al., 2025)	ACL’25	Manim (Python)	✓	×	×	×	×	×	LLM/VLM + Human
PPTAgent (Zheng et al., 2025)	EMNLP’25	PPTX (python-pptx)	×	×	×	✓	×	×	VLM + Auto.
Paper2Video (Zhu et al., 2025)	Preprint	Beamer (L ^A T _E X)	✓	×	×	×	×	×	VLM + Human
AutomaTikZ (Belouadi et al., 2024)	ICLR’24	TikZ (L ^A T _E X)	×	✓	×	×	×	×	Auto.
LESSONWORLD (Ours)	–	LESSONENGINE	✓	✓	✓	✓	✓	✓	Human + VLM Judge

Table 1: Comparison with recent systems for agentially generated educational content. **RT**: runtime-directable; **Interact.**: supports user interaction. **Evaluation**: evaluation method. LESSONWORLD supports all modalities with live, generative, and interactive outputs. Concurrent work finds that interactive AI-generated demos can still hide incorrect logic behind functional interfaces.

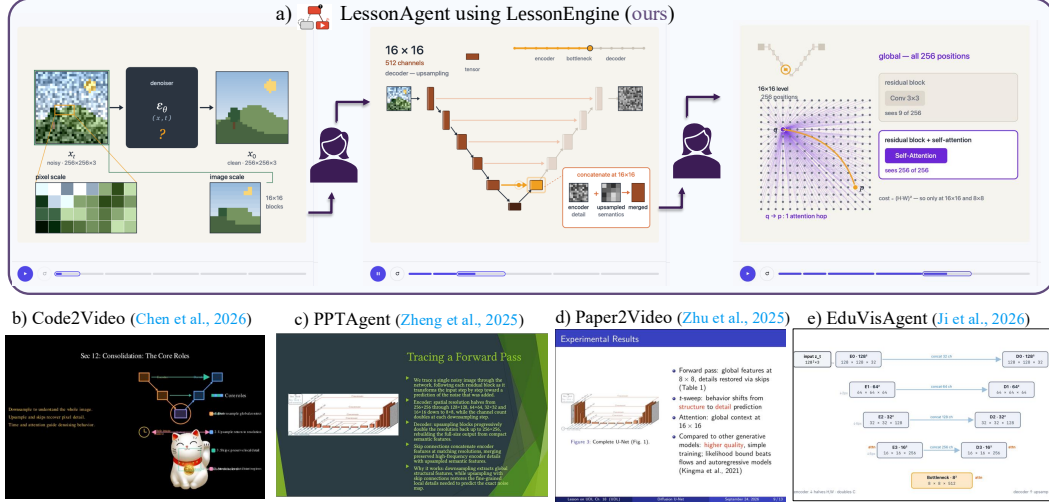


Figure 2: **From static educational artifacts to interactive lessons.** Our LESSONAGENT uses LESSONENGINE to generate interactive lessons that expose underlying concepts and adapt to learner actions at runtime. In contrast, existing methods primarily generate static educational videos (Chen et al., 2026b), slide presentations (Zheng et al., 2025), or narrated videos (Zhu et al., 2025).

3 LESSONWORLD

LESSONWORLD separates interactive lesson authoring from the mechanics of running a lesson. LESSONENGINE is the authoring substrate: a TypeScript library and domain-specific language (DSL) for defining lessons as executable state machines. LESSONAGENT uses this substrate to generate lessons from instructional material. This separation is important for interactive content. Rather than generating a complete application from scratch for every lesson, an agent writes a compact lesson specification while LESSONENGINE provides the runtime machinery needed to execute it.

3.1 LESSONENGINE

LESSONENGINE plays the same role for LESSONWORLD that Manim plays for Manimator (Jain et al., 2025) or PowerPoint for PPTAgent (Zheng et al., 2025): it provides the programming interface through which content is authored and executed. Unlike these substrates, LESSONENGINE is designed for browser-based lessons whose behavior can change during use.

A lesson as a state machine. A lesson is represented as a directed state machine whose nodes are typed *beats*: units of instruction or activity such as explanations, questions, interactive figures, demonstrations, or learner actions. Transitions connect beats and can depend on events such as a learner answer, completed interaction, or tutor command. The runtime maintains the learner’s position, interaction history, and collected learner-specific information. Lessons can therefore pause and resume, follow different paths for different learners, and return to earlier states without restarting the experience.

A minimal authoring language. Authors interact with this state machine through typed lesson primitives rather than implementing browser logic directly. The DSL specifies lesson structure, while LESSONENGINE handles recurring runtime functionality such as event handling, state updates, transitions, pausing and resuming, and synchronization between components. The DSL is implemented in TypeScript on standard browser technologies such as HTML, SVG, and JavaScript, so generated components run directly in the browser and remain inspectable by human authors. Full API details and additional examples are provided in Appendix D.

Why the abstraction matters? The representation moves repeated implementation work out of the model. Agents specify what a lesson should teach and how learners should interact with it, rather than repeatedly generating state management, event handling, and branching logic. More importantly, the typed state-machine representation gives the authoring process a stable intermediate representation: lesson structure can be fixed before content is written, components can be generated independently,

and assembled lessons can be checked automatically. In our measurements, DSL-based authoring requires up to $2\times$ fewer output tokens than generating the corresponding direct implementation. For example, the following declaration creates an interactive polar-curve figure with learner-editable input and worked alternatives, without requiring the author to implement event listeners or state bookkeeping:

```
explorable({ id: "reach", viz: "polar",
controls: [{ key: "curve", kind: "text",
suggestions: [{label: "a = 1.5", value: "1.5*(1+cos(x))"},
{label: "a = 2", value: "2*(1+cos(x))"}] }],
defaults: { curve: "2*(1+cos(x))" }, ask: true })
```

3.2 LESSONAGENT: STRUCTURED LESSON AUTHORING

LESSONAGENT authors executable lessons from textbook chapters, handouts, and figures using LESSONENGINE. It coordinates a Planner, Page Agent, Figure Agent, parallel Act writers, deterministic assembly, Editor, Tutor, and Review Agent over a shared typed representation. Rather than generating a lesson in one invocation, these components plan its structure, generate artifacts in parallel, verify and assemble them, and repair failures before producing the final executable lesson.

Planning and distributed generation. A planner specifies the lesson’s acts, beats, transitions, remediation paths, and learner-state branches, then validates content coverage and transition validity before generation. Incorrect-answer remediation is represented explicitly so later stages cannot silently alter the intended flow. Act writers independently generate narration, on-screen text, questions, feedback, and figure cues from this shared plan. They may modify content but not lesson structure, preserving globally consistent transitions.

The Figure Agent. Given a source figure, nearby instructional context, and a requested interaction specification, the Figure Agent produces a structured blueprint (visual elements, geometry, labels, controls, and guided states) and implements it as a browser-executable 2D or 3D figure. Named elements and states connect figure behavior to lesson narration and cues. Each candidate is rendered in a headless browser, checked for runtime and layout failures, and assessed by a multimodal critic (geometry, interactivity, faithfulness, labels, and conceptual accuracy). An orchestrator accepts the result or requests planning or implementation refinement for at most three attempts (one initial generation and two revisions). This development-time critic is distinct from the post hoc pairwise evaluator. Full details appear in Appendix E.

Verification, assembly, and repair. Artifacts are checked before and after assembly for mechanical, instructional, and visual failures, including invalid states, repeated narration, poor figure legibility, incorrect cues, and narration–visual misalignment. Failed components are regenerated from their failure signals rather than rebuilding the full lesson. The verified components are assembled into a LESSONENGINE lesson. An Editor with access to the complete lesson resolves cross-act inconsistencies, while the Tutor produces the runtime tutor brief. Finally, the Review Agent inspects the rendered storyboard and repairs act or page specifications when needed. Revisions are retained only if they pass verification. Full details appear in Appendix D.5.

Overall, LESSONWORLD replaces unconstrained application synthesis with structured authoring over an executable representation, enabling parallel generation while preserving global lesson structure and providing concrete failure signals for iterative refinement.

4 DATASET AND BENCHMARK DESIGN

4.1 DATASETS AND TASKS

We construct two task sets to evaluate LESSONWORLD: a *figure-level* set isolating generation quality for individual interactive artifacts, and a *lesson-level* set evaluating full authored sessions end-to-end.

The figure-level set comprises 250 tasks, split across four domains (Physics, Computer Science, Chemistry, Mathematics) and spanning both 2D and 3D visualizations. Each task specifies a single generation prompt paired with an interaction specification drawn from a fixed vocabulary of eleven primitives (*orbit*, *drag*, *hover*, *click*, *slider*, *toggle*, *button*, *state*, *animation*, *equation input*, *code editor*), so that every system under comparison is asked to produce the same interactive behavior

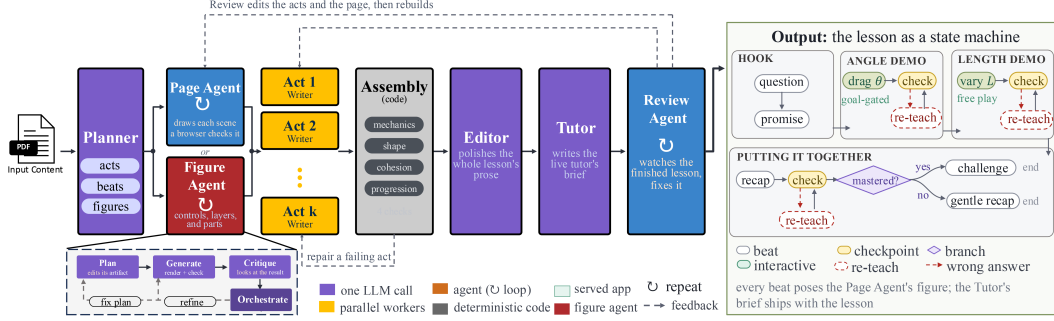


Figure 3: **The LESSONAGENT authoring pipeline.** A planner decomposes input content into acts and beats. Specialized agents generate pages and interactive figures, parallel writers draft the acts, and editor, tutor, and review agents assemble and refine the lesson. LESSONENGINE compiles the result into an executable state machine whose transitions respond to learner actions.

rather than being scored on self-selected functionality. The lesson-level set extends this coverage to full multi-beat sessions, used to compare LESSONWORLD against prior video- and slide-generation pipelines (Section 4.5.1).

4.2 DATA COLLECTION AND ANNOTATION

Prompt and interaction-spec generation. For each figure-level task, we first generate a candidate prompt and interaction specification using an LLM conditioned on the target domain and a target interaction primitive, then manually review every candidate to (i) verify the described interaction is unambiguous and achievable within a single artifact, (ii) confirm the underlying scientific content is correct, and (iii) remove or rewrite prompts that under-specify enough implementation freedom to make cross-system comparison noisy. This mirrors the “LLM-generate, human-verify” construction protocol used by recent benchmarks in this space (e.g., [Chen et al. 2026a](#)), and keeps the fixed-prompt, fixed-spec design controlled across the four domains.

Lesson-level tasks. We construct lesson-level tasks by extending a subset of the most complex figure-level tasks into multi-beat lessons. For each selected task, we use the instructional context of the corresponding source material (e.g., textbooks) and transform it into a standardized lesson specification that defines the learning objective, instructional context, and required lesson structure. This specification serves as the benchmark prompt, providing a common interface across the different models and systems evaluated. The resulting tasks preserve the conceptual and pedagogical intent of the source material while allowing each system to independently determine how to realize the lesson.

4.3 BENCHMARKING SETUP

We evaluate LESSONWORLD at two levels of granularity. We first evaluate complete authored lessons end-to-end against prior educational-content generation systems. We then isolate the figure-generation module using a controlled figure-level benchmark.

Lesson-Level Benchmark. To evaluate LESSONWORLD as a complete system, we compare full lesson sessions against NotebookLM, Manimator, Code2Video, PPTAgent, Paper2Video, EduVisAgent, and a coding-agent baseline. Each system is evaluated in its native output format using shared criteria for content quality, pedagogical effectiveness, and usability. Systems without native interaction are evaluated through their rendered outputs, while interactive systems retain their available controls. This evaluation measures the end-to-end value of LESSONWORLD’s stateful, learner-responsive lesson representation.

Figure-Level Benchmark. We separately evaluate the figure-generation module using the 250-task set described in Section 4.1. We fix the prompt and required interaction specification for every task so that differences between systems reflect generation quality rather than task ambiguity or self-selected functionality.

4.4 EVALUATION PROTOCOL

We use a shared evaluation protocol for figure- and lesson-level comparisons, combining automatic LLM-as-Judge scoring with human evaluation. **LLM-as-Judge.** Gemini 3.1 Pro is our primary judge (2,673 comparisons). **Human Evaluation.** We recruited 22 undergraduate participants across two studies: 12 participants completed 2050 blind pairwise figure comparisons, and 10 participants evaluated 120 videos and interactive lessons using a rubric-based protocol. Item order was randomized and system identity was blinded.

Pairwise comparison and ranking. Generated artifacts for the same prompt are compared blindly, with left/right order counterbalanced. We conduct pairwise comparisons for models within each domain and report each system’s implied win rate against an average competitor, with results pooled across domains for the overall average. Both benchmarks use this protocol, supplemented where applicable by automatic rubric-based scoring following prior checklist- and reference-anchored approaches (Chen et al., 2026a; Zhu et al., 2026a); further details are provided in Section B.

We use the LLM-as-Judge, blind pairwise comparison, and pairwise ranking protocol described in Section 4.4 to score both benchmarks, so that results are directly comparable on the same scale.

4.5 EXPERIMENTAL SETUP

We evaluate LESSONWORLD using the two benchmarks introduced in Section 4.3: a *figure-level* benchmark isolating generation quality for individual interactive artifacts (Section 4), and a *lesson-level* benchmark comparing authored sessions against prior video- and slide-generation systems.

On the figure-level benchmark, we evaluate four backbone models across three generation settings: **Baseline** (a single generation call with no planning), **No Iteration** (planning followed by one generation pass), and **Full Pipeline** (planning followed by up to three refinement iterations, our default configuration). The four backbones are gpt-5.5, kimi-k3, qwen3-8b-max, and gemini-3.1-pro, evaluated consistently across all three settings. For prior figure-generation baselines, we use the default model configurations reported by the corresponding papers.

On the lesson-level benchmark, we compare full LESSONWORLD sessions using the Full Pipeline setting against six prior systems for AI-generated educational content: NotebookLM, Manimator, Code2Video, PPTAgent, Paper2Video, and EduVisAgent. Since these baselines do not natively support the same runtime interaction as LESSONWORLD, we evaluate their rendered outputs using the same lesson-level evaluation protocol.

Human Evaluation. We recruited 22 undergraduate participants majoring in computer science, physics, or chemistry across the two human-evaluation studies. For figure-level evaluation, 12 participants completed 2050 randomly sampled pairwise comparisons, viewing outputs from two systems side by side and selecting the preferred figure. For lesson-level evaluation, 10 participants each completed approximately two hours of evaluation. Across participants, 120 videos were reviewed using a rubric-based protocol that measures multiple aspects of instructional quality.

4.5.1 RESULTS FOR LESSON-LEVEL

Automatic Evaluation Table 2 compares LESSONAGENT against the six prior systems described in Section 4.5. LessonAgent wins 91.0% of its pairwise comparisons against all other systems, the highest average of any method, and ranks first in three of the four domains (Physics, CS and Math). The Coding Agent, which uses the same underlying model (Claude Opus 5) without LessonAgent’s lesson design, is the closest competitor (87.0%) and edges ahead in Chemistry. Against every video-generation system (NotebookLM, Code2Video, Manimator, PPTAgent and Paper2Video), LessonAgent wins at least 97% of head-to-head comparisons, and it wins 73% against EduVisAgent, the strongest prior system.

Human Evaluation. Table 2 compares LESSONWORLD against the six prior systems described in Section 4.4. LESSONWORLD obtains the highest overall rating (7.80/10) and leads the strongest prior system on **all six dimensions**. The largest improvements occur in *Explanation* (8.00 vs. 7.10) and *Usefulness* (6.48 vs. 5.80), while *Concept Match* and *Learning* reach 8.90 and 7.86, respectively. Together, these results indicate that LESSONWORLD improves not only the quality of generated graphics, but also their perceived pedagogical value.

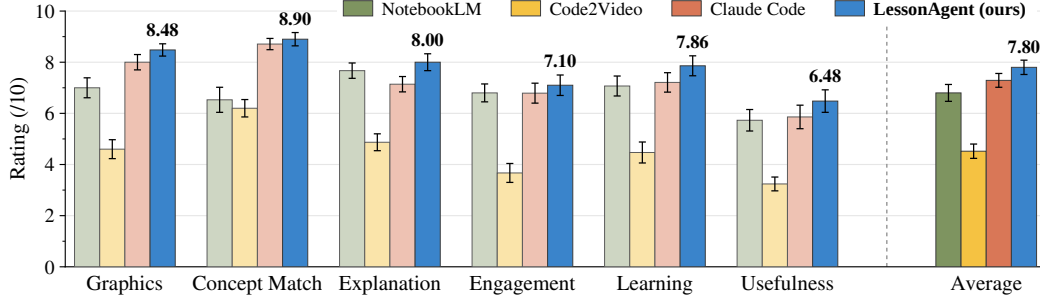


Figure 4: **Human evaluation of lessons generated by each system.** Participants rated each lesson on six dimensions on a 1–10 scale; higher is better. *Concept Match*: how well graphics correspond to the concept; *Learning*: whether participants understood the concept better afterwards; *Usefulness*: whether they would use the lesson to study. *Avg.* is the mean of each answer sheet’s six-item mean. Error bars: ± 1 bootstrap standard error over 10,000 domain-stratified resamples of answer sheets.

Model	Physics (%)	CS (%)	Chemistry (%)	Math (%)	Avg. (%)
NotebookLM	26.0 $\pm$ 4.6	37.5 $\pm$ 4.3	28.6 $\pm$ 2.7	38.7 $\pm$ 3.6	32.7 $\pm$ 1.9
Code2Video (Chen et al., 2026b)	53.8 $\pm$ 3.4	48.3 $\pm$ 4.4	48.3 $\pm$ 2.8	41.9 $\pm$ 2.1	48.1 $\pm$ 1.6
EduVisAgent (Ji et al., 2026)	84.0 $\pm$ 14.3	84.2 $\pm$ 13.3	71.4 $\pm$ 9.2	79.2 $\pm$ 15.3	79.7 $\pm$ 6.6
Manimator (Jain et al., 2025)	54.0 $\pm$ 4.5	56.9 $\pm$ 3.8	61.3 $\pm$ 4.2	62.5 $\pm$ 5.0	58.7 $\pm$ 2.2
PPTAgent (Zheng et al., 2025)	17.3 $\pm$ 1.4	4.8 $\pm$ 2.7	16.7 $\pm$ 0.8	4.8 $\pm$ 2.7	10.9 $\pm$ 1.0
Paper2Video (Zhu et al., 2025)	5.8 $\pm$ 3.3	12.5 $\pm$ 4.2	5.2 $\pm$ 3.1	15.0 $\pm$ 5.4	9.6 $\pm$ 2.1
Coding Agent	84.3 $\pm$ 4.8	88.7 $\pm$ 3.3	92.9 $\pm$ 2.9	82.3 $\pm$ 5.8	87.0 $\pm$ 2.2
LessonAgent (Ours)	92.3 $\pm$ 2.5	89.5 $\pm$ 3.7	89.1 $\pm$ 2.0	93.3 $\pm$ 2.6	91.0 $\pm$ 1.4

Table 2: Full comparison of LessonWorld against existing notebook, code-generation, and agentic video-generation systems across four subject domain ; $\pm$ denotes bootstrap standard error (10,000 resamples). Best results are shown in bold.

4.5.2 RESULTS FOR ABLATION STUDIES

Effect of generation pipeline and backbone. Table 3 compares three generation settings across four backbone models over 250 tasks spanning Physics, Computer Science, Chemistry, and Mathematics. The results isolate the effects of planning, iterative refinement, and backbone choice on figure-generation quality.

Setting	Model	Physics (%)	CS (%)	Chemistry (%)	Math (%)	Avg. (%)
Baseline	Gemini 3.1 Pro	63.8 $\pm$ 2.9	57.7 $\pm$ 3.4	70.0 $\pm$ 3.1	62.7 $\pm$ 3.0	63.5 $\pm$ 1.6
	Kimi K3	43.9 $\pm$ 3.2	51.9 $\pm$ 3.4	47.7 $\pm$ 3.3	45.4 $\pm$ 3.1	47.2 $\pm$ 1.6
	Qwen3-8B Max	41.2 $\pm$ 3.1	39.4 $\pm$ 3.3	41.9 $\pm$ 3.3	42.2 $\pm$ 3.1	41.3 $\pm$ 1.6
	GPT-5.5	55.2 $\pm$ 3.2	62.1 $\pm$ 3.3	60.6 $\pm$ 3.3	62.6 $\pm$ 3.0	60.1 $\pm$ 1.6
With Planner	Gemini 3.1 Pro	46.6 $\pm$ 3.0	35.8 $\pm$ 3.0	44.6 $\pm$ 3.3	38.9 $\pm$ 2.9	41.5 $\pm$ 1.5
	Kimi K3	40.4 $\pm$ 3.0	42.1 $\pm$ 3.2	38.5 $\pm$ 3.2	40.0 $\pm$ 3.1	40.3 $\pm$ 1.5
	Qwen3-8B Max	31.0 $\pm$ 2.8	28.1 $\pm$ 2.9	26.5 $\pm$ 2.9	31.2 $\pm$ 2.7	29.5 $\pm$ 1.4
	GPT-5.5	44.5 $\pm$ 3.2	<u>63.3</u> $\pm$ 3.1	48.0 $\pm$ 3.2	53.5 $\pm$ 3.0	52.2 $\pm$ 1.6
Structured Pipeline	Gemini 3.1 Pro	58.4 $\pm$ 3.0	43.3 $\pm$ 3.2	52.6 $\pm$ 3.2	53.4 $\pm$ 3.0	52.1 $\pm$ 1.5
	Kimi K3	58.0 $\pm$ 3.0	52.9 $\pm$ 3.2	55.3 $\pm$ 3.0	55.8 $\pm$ 3.1	55.4 $\pm$ 1.5
	Qwen3-8B Max	51.1 $\pm$ 3.2	46.0 $\pm$ 3.2	46.4 $\pm$ 3.3	47.3 $\pm$ 3.0	47.9 $\pm$ 1.6
	GPT-5.5	66.0 $\pm$ 3.0	76.2 $\pm$ 2.6	<u>68.0</u> $\pm$ 3.0	67.1 $\pm$ 2.9	69.2 $\pm$ 1.4

Table 3: Figure-level results across three generation settings and four backbone models over 250 tasks. Cells report pairwise win rates (%) within each domain; Avg. pools comparisons across domains. $\pm$ for bootstrap standard error (10,000 resamples). Best results are bold; second-best are underlined.

Comparison with prior figure-generation systems. Table 4 compares LESSONAGENT against prior figure-generation systems and coding-agent baselines using the same figure-level benchmark and evaluation protocol. LESSONAGENT with Claude Opus 5 achieves the highest average win rate and the strongest performance across all four domains.

System	Physics (%)	CS (%)	Chemistry (%)	Math (%)	Avg. (%)
LiveFigure (Shao et al., 2026)	35.9 $\pm$ 2.5	47.4 $\pm$ 2.6	33.4 $\pm$ 2.6	47.8 $\pm$ 2.5	41.3 $\pm$ 1.3
PaperBanana (Zhu et al., 2026a)	57.7 $\pm$ 2.6	42.9 $\pm$ 2.6	56.5 $\pm$ 2.7	53.1 $\pm$ 2.5	52.6 $\pm$ 1.3
AutoFigure (Zhu et al., 2026b)	29.7 $\pm$ 2.4	33.4 $\pm$ 2.5	24.7 $\pm$ 2.3	26.6 $\pm$ 2.3	28.8 $\pm$ 1.2
Coding Agent (Codex GPT-5.5)	34.9 $\pm$ 3.5	45.5 $\pm$ 3.6	53.7 $\pm$ 3.6	38.0 $\pm$ 3.7	43.1 $\pm$ 1.8
Coding Agent (Claude Opus 5)	71.6 $\pm$ 3.5	53.0 $\pm$ 3.7	66.3 $\pm$ 3.5	60.5 $\pm$ 3.6	62.7 $\pm$ 1.8
LESSONAGENT (GPT-5.5)	45.4 $\pm$ 2.5	50.1 $\pm$ 2.6	50.4 $\pm$ 2.7	50.6 $\pm$ 2.5	49.1 $\pm$ 1.3
LESSONAGENT (Claude Opus 5)	73.3 $\pm$ 2.3	75.7 $\pm$ 2.3	66.8 $\pm$ 2.6	73.4 $\pm$ 2.2	72.1 $\pm$ 1.2

Table 4: Figure-level comparison with prior generation systems and coding-agent baselines. Cells report pairwise win rates (%) against an average competitor; $\pm$ denotes bootstrap standard error (10,000 resamples). Best results are bold.

Human evaluation of Generated Figures. Figure 5 summarizes human preferences across the four subject domains. Participants compare figures from two systems side by side and select the preferred output. LESSONWORLD achieves the highest average preference rate, with particularly strong results in Computer Science and Chemistry.

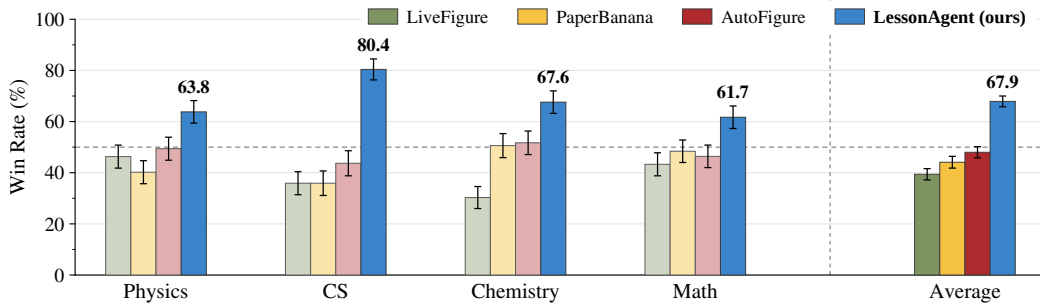


Figure 5: **Human evaluation of generated figures.** Participants compared figures from two systems side by side and chose the one they preferred. We report each system’s pairwise win rate (%) per subject domain and averaged across domains. The dashed line at 50% marks equal preference. LESSONENGINE achieves an average win rate of **67.9%**, compared with 39.4–48.0% for the baselines. Error bars denote ± 1 bootstrap standard error over 10,000 domain-stratified resamples.

5 DISCUSSION

Rather than generating a rendered artifact and stopping, LESSONWORLD maintains an executable lesson representation with explicit instructional structure, learner state, and interaction behavior. Across our evaluations, this yields gains at both levels: the full pipeline achieves a 69.2% average win rate with GPT-5.5, versus 60.1% for the single-call baseline, while LESSONAGENT with Claude Opus 5 reaches 72.1% against prior figure-generation systems. At the lesson level, LESSONAGENT achieves a 91.0% average automatic win rate and the highest human-rated instructional quality.

Interactivity, however, does not guarantee correctness. Consistent with the superficial-competence failure mode reported for interactive scientific demonstrations Chen et al. (2026a), we find cases where an artifact implements an interaction correctly but encodes an incorrect scientific relationship. For example, a slider may update the intended quantity while the equation or visual relationship it exposes is wrong. Keeping content, state, and behavior explicit makes such errors easier to inspect, but does not replace reliable scientific reasoning. Improving factual verification remains an important direction for future work.

More broadly, our results point to a distinction between *generating educational media* and *generating an educational environment*. The former ends when an artifact is rendered; the latter requires a representation that remains executable, inspectable, and responsive after generation. LESSONWORLD takes the latter approach, providing a foundation for future systems with richer learner models, adaptive tutoring, and longer-horizon interaction.

BIBLIOGRAPHY

- Jonas Belouadi, Anne Lauscher, and Steffen Eger. AutomaTikZ: Text-guided synthesis of scientific vector graphics with TikZ. In *The Twelfth International Conference on Learning Representations (ICLR)*, Vienna, Austria, 2024.
- Michael Booth. The ai systems of left 4 dead. In *Artificial Intelligence and Interactive Digital Entertainment Conference at Stanford, 2009*, 2009.
- Qiaosheng Chen, Yang Liu, Lei Li, Kai Chen, Qipeng Guo, Gong Cheng, and Fei Yuan. Interactsience: Programmatic and visually-grounded evaluation of interactive scientific demonstration code generation. *ICML 2026*, 2026a.
- Yanzhe Chen, Kevin Qinghong Lin, and Mike Zheng Shou. Code2video: A code-centric paradigm for educational video generation. *ICML 2026*, 2026b.
- Siwei Han, Haonian Ji, Siyang Xin, Juanquan Shi, Shi Qiu, Xinyu Ye, Peng Xia, Jiaqi Liu, Zhaorun Chen, Yiyang Zhou, Linjie Li, Lijuan Wang, and Huaxiu Yao. Paper2figure: A multi-agent collaborative system for figure generation towards academic research paper. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 29157–29166, 2026.
- Vyoman Jain, Shiva Golugula, Motamarri Sai Sathvik, et al. Animator: Transforming research papers and mathematical concepts into visual explanations. *arXiv preprint arXiv:2507.14306*, 2025.
- Haonian Ji, Shi Qiu, Siyang Xin, Siwei Han, Zhaorun Chen, Dake Zhang, Hongyi Wang, and Huaxiu Yao. From eduvisbench to eduvisagent: A benchmark and multi-agent framework for reasoning-driven pedagogical visualization. In *International Conference on Learning Representations (ICLR)*, 2026.
- Aastha Joshi, Hongyi Ke, Meet Gajjar, Aaron Christian, Qi Wang, and Jun Chen. Llm2manim: Pedagogy-aware ai generation of stem animations. *arXiv preprint arXiv:2604.05266*, 2026.
- Max Ku, Cheuk Hei Chong, Jonathan Leung, Krish Shah, Alvin Yu, and Wenhua Chen. Theorem-ExplainAgent: Towards video-based multimodal explanations for LLM theorem understanding. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.332.
- Chenyang Shao, Jiahe Liu, Fengli Xu, and Yong Li. Livefigure: Generating editable scientific illustration with vlm agents. *arXiv preprint arXiv:2605.23527*, 2026.
- David Thue, Vadim Bulitko, Marcia Spetch, and Eric Wasylishen. Interactive storytelling: A player modelling approach. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, volume 3, pp. 43–48, 2007. doi: 10.1609/aiide.v3i1.18780.
- Hong Yu and Mark O. Riedl. Data-driven personalized drama management. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2013.
- Hao Zheng, Xinyan Guan, Hao Kong, Wenkai Zhang, Jia Zheng, Weixiang Zhou, Hongyu Lin, Yaojie Lu, Xianpei Han, and Le Sun. PPTAgent: Generating and evaluating presentations beyond text-to-slides. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 14413–14429, Suzhou, China, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.emnlp-main.728.
- Hao Zheng, Guozhao Mo, Xinru Yan, Qianhao Yuan, Wenkai Zhang, Xuanang Chen, Yaojie Lu, Hongyu Lin, Xianpei Han, and Le Sun. Deeppresenter: Environment-grounded reflection for agentic presentation generation. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 31545–31558, 2026.
- Dawei Zhu, Rui Meng, Yale Song, Xiyu Wei, Sujian Li, Tomas Pfister, and Jinsung Yoon. Paperbanana: Automating academic illustration for ai scientists. *arXiv preprint arXiv:2601.23265*, 2026a.

Minjun Zhu, Zhen Lin, Yixuan Weng, Panzhong Lu, Qiujie Xie, Yifan Wei, Sifan Liu, QiYao Sun, and Yue Zhang. Autofigure: Generating and refining publication-ready scientific illustrations. In *International Conference on Learning Representations (ICLR)*, 2026b.

Zeyu Zhu, Kevin Qinghong Lin, and Mike Zheng Shou. Paper2video: Automatic video generation from scientific papers. *arXiv preprint arXiv:2510.05096*, 2025. Presented at the Scaling Environments for Agents (SEA) Workshop, NeurIPS 2025.

APPENDIX

This appendix contains additional results, dataset examples, tables, figures, prompts, and implementation details used throughout the paper. We include these materials to support reproducibility and provide further analysis beyond the main text.

A	Additional Examples	14
B	Figure-Level Pairwise Evaluation Protocol	14
B.1	Shared Pairwise Judge Instruction	15
B.2	Dimension Prompts	15
B.3	Runtime Message Templates	17
B.4	Aggregation Prompt	17
B.5	Bradley–Terry Ranking	18
B.6	Deterministic Render Validation	18
B.7	Evaluation Reproducibility and Scope	18
C	Human Study Protocol	18
C.1	Systems and materials	19
C.2	Blinding and counterbalancing	19
C.3	Study platform	19
C.4	Participants and procedure	20
C.5	Questionnaire	20
C.6	Behavioural logging	20
C.7	Analysis	21
D	LESSONAGENT: Full Details	22
D.1	Pipeline at a glance	22
D.2	Structured stages and the repair protocol	22
D.3	Planning	24
D.4	Figures: the Page Agent	24
D.5	Act writers	25
D.6	Assembly and the four checks	25
D.7	Editor	27
D.8	Tutor brief	27
D.9	Review Agent	27
D.10	At play time: the live tutor	28
D.11	Prompt design and disclosure	28
E	Figure Generation Pipeline	30
E.1	Pipeline Overview	30
E.2	Inputs and Context Construction	31

648	E.3 Planner Blueprint	31
649	E.4 Generation and Scaffold Contracts	32
650	E.5 Deterministic Browser Verification	32
651	E.6 Development-Time Critic	33
652	E.7 Orchestrator Prompt and Refinement Policy	33
653	E.8 Experimental Conditions and Stopping	33
654	E.9 Methodological Scope	34
655		
656		
657		
658		
659		
660		
661		
662		
663		
664		
665		
666		
667		
668		
669		
670		
671		
672		
673		
674		
675		
676		
677		
678		
679		
680		
681		
682		
683		
684		
685		
686		
687		
688		
689		
690		
691		
692		
693		
694		
695		
696		
697		
698		
699		
700		
701		

A ADDITIONAL EXAMPLES

This section shows additional examples of generated lessons. Because the lessons are interactive, static frames capture only part of them; we recommend visiting the anonymous project page at <https://lessonworld.fly.dev>, where all 20 benchmark lessons can be played and compared side by side with the output of every baseline.

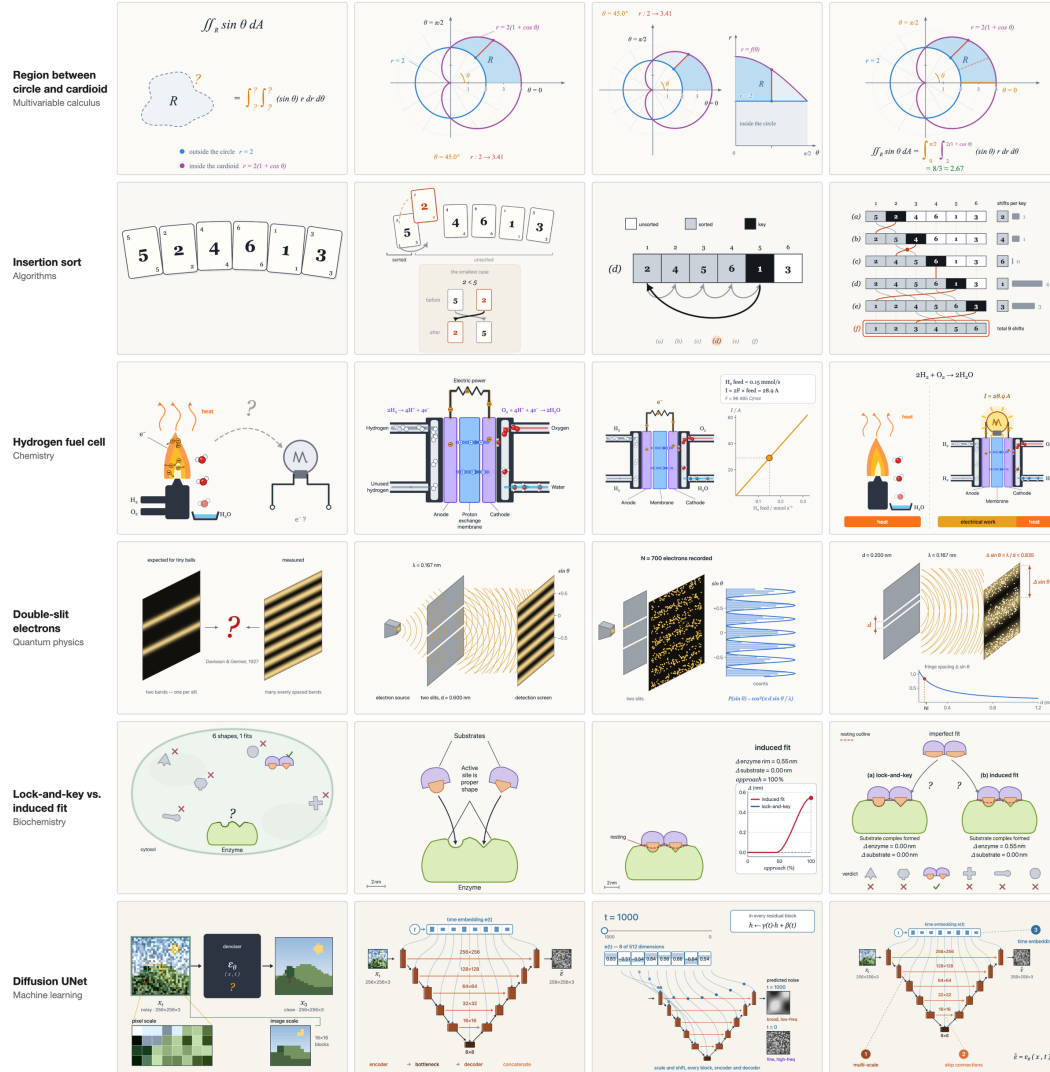


Figure 6: **Storyboards of lessons generated by LESSONAGENT.** Each row is one lesson and each column one moment from it, in the order the learner meets them. The six lessons span six subjects: a polar double integral, insertion sort, a hydrogen fuel cell, the electron double-slit experiment, lock-and-key versus induced-fit enzyme binding, and a diffusion U-Net. Each lesson opens on a question, builds the figure up piece by piece, lets the learner vary it, and closes on the result. Every frame is the lesson’s own interactive figure in the state set by that step of the lesson. All 20 lessons can be played at <https://lessonworld.fly.dev>.

B FIGURE-LEVEL PAIRWISE EVALUATION PROTOCOL

This section documents the complete prompt-based evaluation protocol used for the figure-level benchmark in Section 4.4. We follow the prompt-disclosure practice of prior work on automated scientific and educational visualization by reporting the judge instructions, evidence available to

each judge, output contract, aggregation rule, and ranking procedure. Prompt wording is reproduced from the implementation used for the evaluation; line wrapping and typographic punctuation are normalized for typesetting.

Execution. For each source figure, two generated candidates are mapped to anonymized *Figure 1* and *Figure 2* positions using one random assignment. The assignment is held fixed across five dimension-specific judges and the aggregator, after which the positional winner is mapped back to the generating system. The five dimension calls execute in parallel. Our primary evaluator is Gemini 3.1 Pro. Each call has an 8,192-token output limit. The provider adapter does not explicitly set temperature, top- p , or a random seed.

Judge-specific evidence. The geometry/structure and interactivity judges receive both candidates' complete HTML/JavaScript. The faithfulness and label judges receive the source textbook image and default-state screenshots of both candidates. The concept judge receives the numbered source chapter and both candidates' HTML/JavaScript. This specialization prevents a judge from relying on evidence that is not relevant to its assigned dimension.

B.1 SHARED PAIRWISE JUDGE INSTRUCTION

The following instruction and output contract are prepended to every 3D dimension prompt:

```
You are a strict evaluator comparing two Three.js 3D figure implementations (Figure 1 and
Figure 2) of the same textbook figure.
Be critical and honest. Only call "tie" when both figures are genuinely indistinguishable on
this dimension -- default to picking the better one.
Your rationale must be comparative: explain specifically what the winner does better AND what
the loser does worse -- not just why the winner is good in isolation.
Respond ONLY with valid JSON -- no explanation, no markdown:
{"winner":"1"|"2"|"tie","confidence":0.0-1.0,"rationale":"<one sentence comparing both figures
>"}
```

For 2D figures, the first sentence is replaced with:

```
You are a strict evaluator comparing two inline-interactive 2D figure implementations (SVG.js
/ Chart.js / Mermaid) of the same textbook figure (Figure 1 and Figure 2).
```

The remaining instructions and JSON contract are unchanged.

B.2 DIMENSION PROMPTS

B.2.1 GEOMETRY AND STRUCTURAL ACCURACY

For 3D candidates:

```
DIMENSION: Geometric accuracy -- which figure better reconstructs the 3D geometry of the
original.
Evaluate: correct 3D shapes/primitives, spatial relationships, element counts, proportions,
depth/perspective, and initial camera viewpoint matching the source.
Watch for: incorrect 3D perspective, wrong shapes for the concept, initial viewpoint differing
from the source, proportions noticeably off. Take special note of 2D canvases in 3D
space - this automatically loses.
Figure 1 and Figure 2 are provided as HTML/JavaScript source code.
```

For 2D candidates:

```
DIMENSION: Structural accuracy -- which figure better reconstructs the 2D structure of the
original.
Evaluate: axis ranges, tick marks and gridlines, the shape and trend of the data, node-and-
edge topology and counts, relative proportions and layout, and line styling (dashed vs
solid, stroke weight, arrowheads, fill vs outline).
Watch for: the wrong mark type for the concept (a scatter drawn as a connected line, a bar
chart drawn as an area, a directed edge missing its arrowhead), missing series or nodes,
proportions noticeably off.
These are 2D figures: do NOT reward or penalise depth, perspective, or 3D vantage point.
Figure 1 and Figure 2 are provided as HTML/JavaScript source code.
```

B.2.2 INTERACTIVITY AND USABILITY

For 3D candidates:

DIMENSION: Interactivity and usability -- which figure provides better developer-built interactions.
 CRITICAL: OrbitControls (mouse drag to rotate/zoom) does NOT count as a meaningful interaction. Meaningful = buttons, sliders, toggles, step-through animations, parameter controls built by the developer.
 Evaluate: number of meaningful interactions, whether they are functional, and whether they are pedagogically useful.
 Watch for: controls present in code but non-functional, only OrbitControls with no developer-built interactions.
 Figure 1 and Figure 2 are provided as HTML/JavaScript source code.

For 2D candidates:

DIMENSION: Interactivity and usability -- which figure provides better developer-built interactions.
 CRITICAL: panning/zooming the figure does NOT count as a meaningful interaction, and a hover tooltip on its own is weak. Meaningful = sliders, toggles, buttons, demo-step players, equation inputs, or code-editor workbenches built by the developer whose handlers actually redraw the figure.
 Evaluate: number of meaningful interactions, whether they are functional, and whether they are pedagogically useful.
 Watch for: controls present in code but non-functional, handlers that never redraw, a figure that is entirely static.
 Figure 1 and Figure 2 are provided as HTML/JavaScript source code.

B.2.3 VISUAL FAITHFULNESS

For 3D candidates:

DIMENSION: Visual faithfulness -- which figure's rendered output more closely matches the original textbook figure.
 Evaluate: color match, compositional similarity, proportions, overall visual resemblance at a glance.
 Watch for: colors that don't match the original, elements present that don't appear in the original, proportions noticeably off.
 You will receive the original textbook figure image, plus rendered screenshots of Figure 1 and Figure 2.

For 2D candidates:

DIMENSION: Visual faithfulness -- which figure's rendered output more closely matches the original textbook figure.
 Evaluate: color match, compositional similarity, proportions, overall visual resemblance at a glance.
 Judge the screenshots in their default (step-0) state -- each is meant to be a drop-in replacement for the static source figure.
 Watch for: colors that don't match the original, elements present that don't appear in the original, proportions noticeably off.
 You will receive the original textbook figure image, plus rendered screenshots of Figure 1 and Figure 2.

B.2.4 LABEL QUALITY

For 3D candidates:

DIMENSION: Label quality -- which figure has better text labels and annotations matching the original.
 Evaluate: presence of all required labels, correctness of label text, readability, size, placement, and freedom from clutter.
 Watch for: important annotations absent, labels not present in the original figure.
 You will receive the original textbook figure image, plus rendered screenshots of Figure 1 and Figure 2.

For 2D candidates:

DIMENSION: Label quality -- which figure has better text labels and annotations matching the original.
 Evaluate: presence of all required labels, correctness of label text, readability, size, placement, and freedom from clutter.

2D figures are NOT screened by an automated overlap checker, so actively penalise labels that collide with each other or with the artwork, escape the plot/diagram area, are clipped at the edge, or are too small to read comfortably at inline size.
 Watch for: important annotations absent, labels not present in the original figure.
 You will receive the original textbook figure image, plus rendered screenshots of Figure 1 and Figure 2.

B.2.5 CONCEPT GROUNDING

The concept prompt is shared across 2D and 3D candidates:

1. EXTRACT: list the key concepts taught by this textbook section (1-4 concepts)
 2. FOR EACH CONCEPT: find the book lines that define it; assess how well Figure 1 and Figure 2 each teach it via their labels, tooltips, interactions, and geometry
 3. IDENTIFY MISMATCHES: claims in either figure that contradict or ignore the textbook
 4. DECIDE: which figure better grounds its interactions and labels in the textbook in a clear manner.
 The numbered textbook lines as well as Figure 1 and Figure 2 are provided as HTML/JavaScript source code.

The shared JSON contract requests winner, confidence, and rationale. Although the implementation can preserve optional structured concept details when a model emits them, those optional fields are not required by the executed prompt and are not used in the reported ranking.

B.3 RUNTIME MESSAGE TEMPLATES

The geometry and interactivity judges receive:

Figure 1 source code:
 [FIGURE_1_HTML]

 Figure 2 source code:
 [FIGURE_2_HTML]

 Output ONLY the JSON object.

The faithfulness and label judges receive:

Reference textbook figure:
 [SOURCE_IMAGE]

 Screenshot of Figure 1:
 [FIGURE_1_SCREENSHOT]

 Screenshot of Figure 2:
 [FIGURE_2_SCREENSHOT]

 Output ONLY the JSON object.

The concept judge receives:

Numbered textbook QMD:
 [NUMBERED_CHAPTER_TEXT]

 Figure 1 source code:
 [FIGURE_1_HTML]

 Figure 2 source code:
 [FIGURE_2_HTML]

 Output ONLY the JSON object.

B.4 AGGREGATION PROMPT

The aggregator receives only the five dimension outputs; it does not receive the original image, candidate screenshots, source code, or chapter text.

You receive pairwise preferences from five independent dimension agents that each compared Figure 1 vs Figure 2.

```

Synthesize a final judgment. Up-weight agents with higher confidence.
Only call "tie" if the dimension votes are genuinely split with no clear overall winner.
Your explanation must be comparative: cite specific strengths of the winner over the loser and
specific weaknesses of the loser relative to the winner -- not just a list of the winner
's merits in isolation.
Respond ONLY with valid JSON -- no explanation, no markdown:
{"winner": "1"|"2"|"tie", "confidence": 0.0-1.0, "explanation": "<two to three sentences comparing
both figures>"}

```

The runtime message summarizes each dimension as `dimension: winner=[1|2|tie], confidence=[0-1], rationale="[TEXT]"`. Confidence weighting is therefore performed by the aggregator model through this instruction; it is not a deterministic numerical weighted vote.

B.5 BRADLEY-TERRY RANKING

Each aggregate judgment contributes one pairwise outcome. A win contributes 1 to the winner, a loss contributes 0, and a tie contributes 0.5 to each system. Let W_i be the resulting total wins for system i , n_{ij} the number of comparisons between systems i and j , and $p_i > 0$ the latent system strength. The implementation applies the minorization-maximization update

$$p_i \leftarrow \frac{W_i}{\sum_{j \neq i} n_{ij} / (p_i + p_j)}. \quad (1)$$

Strengths are initialized to one and normalized to sum to one after each update. Iteration stops when the largest normalized parameter change is below 10^{-8} or after 500 iterations. Systems are ranked in descending order of p_i . We compute the overall machine ranking from aggregator winners and separate rankings from each dimension’s winners. Aggregator confidence does not enter the Bradley-Terry fit.

B.6 DETERMINISTIC RENDER VALIDATION

The generation pipeline also renders generated figures in a headless browser before accepting them as mechanically valid candidates. For 3D outputs, checks cover uncaught exceptions, error-overlay visibility, render-loop liveness when instrumented, GPU draw calls, scene content, non-finite geometry, blank or zero-sized canvases, horizontal overflow, off-screen controls, and equation overflow. Console errors, vertical overflow, clipped controls, and label placement/size/overlap are recorded as warnings. These checks assess executability and layout validity, not semantic quality, and do not contribute additional LLM votes.

B.7 EVALUATION REPRODUCIBILITY AND SCOPE

The generation pipeline also uses a development-time critic whose feedback can change subsequent outputs. We treat that critic as part of the generation method rather than as an independent final benchmark judge. Likewise, user-triggered absolute rubric scores are distinct from the pairwise observations used for the Bradley-Terry results above.

To audit a pairwise run, we retain the candidate-system identities, randomized Figure 1/Figure 2 assignment, five dimension outputs, aggregate judgment, evaluator model, and evaluation timestamp. The current implementation codes ties as half-wins and does not fit a tie-specific parameter or use judge confidence in the ranking. It also does not natively estimate rank uncertainty or enforce a minimum number of comparisons per pair.

C HUMAN STUDY PROTOCOL

We ran a blind, within-subjects comparison in which participants worked through four lessons on the same topic and rated each one. This appendix describes the materials, the study platform, the procedure, the questionnaire, the inclusion rules, and the analysis.

C.1 SYSTEMS AND MATERIALS

The human study compares four of the systems from the automated evaluation, on the same 20 generation briefs and textbook figures (Table 5). Two produce narrated videos and two produce interactive web pages, so participants saw each system in the format it naturally produces. Unlike the automated judge, which saw silent videos, participants heard every system’s narration.

System	Format	What participants received
LESSONAGENT (ours)	interactive lesson	The lesson running in the LESSONENGINE player, with ElevenLabs narration and the live tutor (question box) enabled.
Coding Agent	interactive page	The page Claude Code built from the brief with no harness, served as built, with its own narration.
NotebookLM	video	The generated video, with its original narration.
Code2Video (Chen et al., 2026b)	video	The generated video, with its original narration.

Table 5: The four systems in the human study.

During the study, LESSONAGENT’s runtime tutor ran on `gemin-3.5-flash` rather than the model listed in Table 9. Narration was pre-synthesized and cached, so it did not depend on the tutor model.

C.2 BLINDING AND COUNTERBALANCING

Each topic’s four versions appear under the neutral labels A, B, C and D. The system behind each label is fixed per topic and balanced across topics: over the 20 topics, every system appears under every label exactly five times. Consequently no system is consistently the first one a participant opens, and a participant who completes several topics cannot carry a letter-to-quality association from one topic to the next. The mapping is identical for all participants on a topic, and it is recorded only in the exported data. No page title, URL, file name or on-screen text names any system, and the site itself is titled “Lesson Study”. Participants could see whether a version was a video or an interactive lesson, because the format cannot be hidden.

C.3 STUDY PLATFORM

The study ran as a single web application deployed on Fly.io (one `shared-cpu-1x` machine with 1 GB of memory and a persistent volume for logs and cached narration). Participants signed in with their institutional email address, which had to be on a pre-registered list of invitees; the address served as both credential and participant identifier. After signing in, a participant chose a topic from the list of 20, shown with its subject and a neutral title (Table 6).

The topic page shows one version at a time, selected by tabs A–D, with the questionnaire in a resizable panel beside it. The questionnaire panel follows the selected tab, showing the questions for that version. Switching away from a version pauses all of its playback (video, audio and speech), and switching back resumes it, so two lessons never play at once. Answers are saved as they are given and can be changed at any time until the participant presses *Finish this topic*, which marks the answer sheet as submitted. The on-screen instructions read:

Each topic has four versions of the same lesson, labelled A, B, C and D. Pick a topic, watch or work through all four in any order, then answer the questionnaire about it. You can come back and do another topic afterwards.

and, on the topic page:

Four versions of the same lesson — some are videos, some are interactive, and each one says which above it. Go through them in any order, as often as you like, and answer the **Questionnaire** as you go.

ID	Subject	Topic as shown to participants
p1	Calculus	Proving $\lim_{x \rightarrow 0} \sin x/x = 1$ by squeezing
p2	Calculus	Volume of a tetrahedron by double integration
p3	Calculus	Setting up a polar double integral
p4	Computer science	How insertion sort works, step by step
p5	Computer science	Tracing the Bellman-Ford algorithm
p6	Calculus	Fluid force on a submerged surface
p7	Calculus	The cross product of two vectors
p8	Chemistry	Inside a hydrogen-oxygen fuel cell
p9	Chemistry	The Rutherford gold foil experiment
p10	Chemistry	Electron diffraction and wave-particle duality
p11	Biochemistry	Lock-and-key versus induced fit enzyme binding
p12	Physics	The simple pendulum
p13	Physics	The cylindrical capacitor
p14	Machine learning	The U-Net behind diffusion models
p15	Machine learning	Edge graphs in graph neural networks
p16	Organic chemistry	Steric bulk and inversion in an S_N2 reaction
p17	Machine learning	Fitting a line to data: slope, intercept and error
p18	Optics	Why there is no picture on the wall: the pinhole camera
p19	Computer vision	Orthographic projection of a box onto a plane
p20	Optics	Lenses in a telescope

Table 6: The 20 study topics.

C.4 PARTICIPANTS AND PROCEDURE

Participants were recruited through an advertisement email. The study was conducted with the required institutional review board (IRB) approval, and every participant gave informed consent before taking part. Twenty respondents were registered by their university email address, and 10 of them completed at least one topic. Participants chose their own topics. A participant could complete any number of topics, and several participants could complete the same topic. Within a topic, participants could open the four versions in any order and return to any of them while answering.

C.5 QUESTIONNAIRE

Each answer sheet has three blocks: questions asked once per version, comparison questions asked once per topic, and questions about the study site itself (Table 7). Participants were told to rate each version while it was fresh and to answer the comparison questions after seeing all four. Follow-up text questions appear only when the preceding answer calls for them (for example, “what went wrong?” appears only if a problem was reported). The interactivity question is asked only of the two interactive systems, and the question about the tutor only of LESSONAGENT, the only system with one. These restrictions follow the system, not the letter, so they move with each topic’s label assignment.

C.6 BEHAVIOURAL LOGGING

Alongside the answers, the platform keeps an append-only journal per participant. It records which version was opened and when; every play, pause, seek and end of a video; a tick every ten seconds naming the version on screen and whether the browser tab was in the foreground; every change to an answer, in order; and, for LESSONAGENT, the lesson’s own beat-by-beat history. We report viewing time as foreground time, which is accurate to about ten seconds per version. Across included answer sheets, the median participant spent about 20 minutes in the foreground on one topic.

Item	Wording	Response
<i>Per version (asked of each of A–D)</i>		
graphics*	Visual quality of the graphics	1–5 (poor–excellent)
figure_faithful*	The figure matched the concept being taught	1–5 (not at all–completely)
errors	Did you notice anything wrong — a wrong label, a wrong formula, a figure that contradicts the narration?	nothing wrong / minor slips / serious errors / not sure; then free text
explanation*	Quality of the explanation	1–5 (poor–excellent)
follow*	How easy was it to follow?	1–5 (hard–effortless)
hook*	The opening made me want to keep going	1–5 (not at all–very much)
engaging*	How engaging was it overall?	1–5 (dull–gripping)
understood*	I understood the concept better afterwards	1–5 (not at all–a lot better)
pace	Its length and pacing were	too short ... too long (5 options)
technical	Technical problems (stutters, broken animation, audio out of sync, nothing loading)	none / minor glitches / major problems; then free text
interaction	The parts you could click, drag or control helped you learn	1–5; interactive systems only
chat	Being able to ask the lesson a question	did not try it / not / somewhat / very helpful; LESSONAGENT only
study_from*	I would use this to study for an exam	1–5 (never–definitely)
notes	Anything that stood out, good or bad?	free text
<i>Comparison (once per topic)</i>		
Best overall; weakest overall; best graphics; best opening; clearest explanation; most engaging; most accurate (fewest errors); the one you would actually study from		pick one of A–D
Rank them best to worst		order A–D
For this topic, what suited it better?		video / interactive / depends / neither
Why that order?; why that format?; what did every version miss?		free text
<i>Study site</i>		
The study site itself was easy to use		1–5 (confusing–effortless)
Did you go back and forth between versions while answering?		often / once or twice / no
Anything broken or confusing about the site?; anything else?		free text

Table 7: The questionnaire. Items marked * form the overall score (§C.7).

C.7 ANALYSIS

The unit of analysis is the answer sheet. The *overall score* of a version is the mean of the eight 1–5 items marked * in Table 7; the interactivity and tutor items are excluded because they are not asked of every system.

- **Ratings.** For each item and system we report the mean over sheets with a 95% percentile bootstrap confidence interval (4,000 resamples of sheets).
- **Paired comparisons.** For LESSONAGENT against each other system, we take the within-sheet difference in overall score and test whether its mean is zero with a two-sided sign-flip permutation test (20,000 permutations). Pairing within a sheet controls for the participant and the topic.

- **Picks.** For each comparison question we count the sheets that chose each system.
- **Rankings.** We report each system’s mean rank and how often it was ranked first. For every pair of systems we compute the share of sheets that ranked one above the other, with a Wilson 95% interval and an exact two-sided binomial (sign) test on the sheets that ranked the pair decisively.

All p -values are reported without correction for multiple comparisons. Given the sample of 30 sheets, we treat the study as a complement to the automated evaluation rather than a standalone test.

D LESSONAGENT: FULL DETAILS

This appendix describes LESSONAGENT completely: every stage, what it is given, what it returns, what accepts or refuses it, and every model and agent setting. Section D.11 summarizes each stage’s prompt contract and disclosure artifacts; the resulting lesson structure is illustrated in Figure ??.

D.1 PIPELINE AT A GLANCE

Stage	Kind	Produces	Accepted by	If refused
Planning	one LLM call	acts, typed beats, every transition, skills granted by checkpoints	plan validation, coverage of the source, pedagogical shape	resubmit with the failures
Page Agent	coding agent	one browser page drawing one picture per scene	headless-browser oracle	same session resumed with the remaining problems; the run fails if they persist
Act writers (k)	one LLM call per act, 3 at a time	the content of every beat in one act	assembly, mechanics, and shape, on the act’s own beats	resubmit
Assembly	code	the complete lesson	the four checks	the failing act is rewritten
Editor	one LLM call	prose edits	mechanics, shape, and cohesion, no worse than before	resubmit
Tutor	one LLM call	the runtime tutor’s brief and an offline fallback	length and format checks	resubmit
Review Agent	coding agent	edits to the acts, the figure page, and the tutor brief	rebuild through assembly, the four checks, and the oracle	revision discarded; last passing lesson kept
Emit	code	the lesson directory	zero errors	nothing emitted

Table 8: The stages of LESSONAGENT, in order.

D.2 STRUCTURED STAGES AND THE REPAIR PROTOCOL

Five stages are single LLM calls. Each submits one JSON object through a tool call, and code accepts or refuses it; no stage is graded by another model. The checks were written for the engine, not for the model, so a refusal states a fact about the lesson (“a learner who lands here can never reach an ending”) rather than an opinion. On refusal the model’s own submission is returned as its previous turn, followed by:

```

That was refused. Every problem below is from a checker that will run again, so fix each
one:
* <problem>
* <problem>

Resubmit the WHOLE object with '<tool>'. Change only what the problems name; keep everything
else byte for byte.

```

Setting	Value
<i>Structured stages (planning, figure, act writers, editor, tutor)</i>	
Model	claude-opus-5
Maximum output tokens	16,000 per call
Output format	one forced tool call per response; the tool’s input schema is the stage’s output format (§D.11)
Repairs	2 per stage (at most 3 submissions); a refusal returns the checker’s failures verbatim
Replies with no tool call	answered with a reminder, up to 2 times, not counted as repairs
Provider errors	retried up to 2 times, not counted as repairs
Act writers in parallel	3 at a time
Repair rounds after assembly	2 with the Page Agent, 1 without
<i>Page Agent</i>	
Agent	Claude Code (claude CLI, headless), model claude-opus-5
Turns	at most 120 per session; at most 3 sessions (the first, then 2 resumptions of the same session)
Tools	Read, Write, Edit, Glob, Grep; shell restricted to <code>npx tsx, node, ls</code> ; web search and fetch disabled; no user settings or MCP servers loaded
Workspace	a fresh directory holding only the brief and its inputs; the agent may write nowhere else
<i>Review Agent</i>	
Agent	Claude Code, model claude-opus-5
Turns	at most 400; resumed once if it stops on an error
Tools	Read, Write, Edit, Glob, Grep; shell restricted to the rebuild script, <code>npx tsx, ls</code>
<i>Runtime tutor (at play time, not authoring)</i>	
Model	claude-opus-5, adaptive thinking, at most 12,000 output tokens per turn

Table 9: Model and agent settings.

Returning the submission as the model’s own turn makes the model repair its text rather than redraft from the brief, which tends to fix the named problem without breaking something else.

D.3 PLANNING

The planner decides the lesson’s shape and writes no content. It receives the request (objective, scene plan, constraints, source text) with the draft cut into numbered units, and returns a plan: acts, each with an objective and the draft units it covers, and in each act an ordered list of beats. Each beat has an id, a registered type, a one-line *intent* (the assignment its writer works from), and every transition leaving it: the next beat, where a wrong answer goes, the two arms of a branch and its condition, and the skill a checkpoint grants.

The plan owns every transition because act writers run in parallel and cannot see one another: if writers set transitions, two would target the same id and a third would rename one. Assembly merges the plan’s transitions over whatever a writer returns. The skill a checkpoint grants and the branch that tests it usually sit in different acts, so both ends are the planner’s.

The plan is refused unless every transition lands on a declared beat, every draft unit is covered by exactly one act, and the plan already has the shape of an interactive lesson (the shape codes in §D.6 that can be judged from structure alone). This happens before any content is written, so a plan with no checkpoint costs one call rather than a whole lesson.

D.4 FIGURES: THE PAGE AGENT

The Page Agent writes the lesson’s figure as one self-contained HTML page with a different picture for each scene of the request. What it draws is its own choice. Its brief fixes only how the lesson drives the page:

- a `scene` slider selects the picture; other sliders, toggles, and text fields call `global update_key` and `toggle_key` functions;
- `window.LS_FIGURE` declares the scenes, the *layers* each scene can reveal one at a time, and the named *parts* the narration can point at;
- `update_emphasize(parts)` keeps the named parts at full strength and dims everything else;
- the picture is a function of the controls and the clock, so the same pose always draws the same picture; all text is SVG or HTML so it can be measured; nothing is loaded from the network.

The oracle. The agent runs a headless-browser oracle as often as it likes. The oracle freezes the page’s clock, so every screenshot is reproducible, then refuses the page if: the scene slider and declarations disagree, or there are fewer scenes than the request needs; there are fewer than two layers or three parts; two scenes draw the same picture; a control, a layer, or a part changes nothing; a text field’s examples draw the same picture, or bad input breaks it; text runs off the stage, overlaps other text, or is too small to read; text is drawn on a canvas, where it cannot be measured; or the page throws. It saves a screenshot of every scene with all layers on and all off, and the brief tells the agent to look at each one and judge it against a written quality bar: a distinct idea per scene, drawn to scale from the physics, legible, with one colour per quantity.

Acceptance. When the agent stops, the pipeline runs the oracle itself. If problems remain, the same session is resumed with them, up to twice; a page that still fails ends the run instead of being used. The accepted page’s declared scenes, layers, and parts become the vocabulary the act writers use to pose it.

Without the Page Agent. A single LLM call writes the figure as data instead: elements whose numeric attributes can be bound to controls, grouped into layers switched by toggles. It is refused if a binding names a missing control, if any element leaves the stage or evaluates to NaN at any control extreme, if a layer does not hide what it lists, or if colours are illegible in either theme.

D.5 ACT WRITERS

One writer per act, three at a time. Each receives the request, its act’s slice of the plan (beat ids, types, intents, and the transitions already fixed), the draft units the act covers, and the figure’s declared scenes, layers, and parts. It returns the parameters of every beat it was given: narration, on-screen text, questions, choices with the misconception each wrong choice reveals, feedback, the controls and goal of an interactive beat, and the figure’s pose and narration cues. Writers cannot add, rename, or reorder beats or set transitions. Assembly discards anything structural a writer returns, and a missing beat becomes a visible placeholder. A writer’s output is accepted when the lesson, assembled with it, passes the mechanics and shape checks on that act’s own beats.

D.6 ASSEMBLY AND THE FOUR CHECKS

Assembly is code. It merges the plan and the acts into one lesson specification, expands every figure reference against the figure, and runs four checks: *mechanics* (the lesson compiles, runs, and renders), *pedagogical shape* (it asks, re-teaches, forks, and invites questions), *cohesion* (it reads as one lesson), and *visual progression* (it reveals its figure in order and shows all of it). Every failure code is listed at the end of this section. Errors block the lesson; warnings are reported.

Each error names a beat, and the plan says which act owns it. That act alone is regenerated, with its previous output and the errors appended to its prompt, for up to two rounds. One fault is routed specially: a layer the Page Agent drew that no beat ever shows is assigned to the first act that poses the scene containing it, because only that act can bring it on screen. Errors that concern the lesson as a whole name no act and are reported rather than routed, since handing “the lesson has no checkpoint” to one writer would add a checkpoint in an arbitrary place.

Every failure code.

Plan validation (`compose/plan.ts`). Run on the plan, before any content is written.

BAD_COVERAGE	An act claims a draft unit that does not exist.
BAD_ID	The lesson id is not a lowercase slug.
BAD_VERSION	The plan’s version is not 1.
DANGLING_TARGET	An edge points at a beat id no beat in the plan declares.
DUPLICATE_ACT_ID	Two acts share an id.
DUPLICATE_BEAT_ID	Two beats share an id.
NO_ACTS	The plan has no acts.
NO_BEATS	An act has no beats.
NO_INTENT	A beat has no intent, so its writer would invent an assignment.
TOPOLOGY_LEAK	A writer-owned field appearing in <code>topology</code> , or vice versa.
UNGRANTED_SKILL	A branch forks on a skill no beat in this plan grants.
UNKNOWN_BEAT_TYPE	A beat names a type the engine has not registered.

1350	Mechanics (<code>compose/verify.ts</code>). Does the lesson run: compile, terminate, and render at every	
1351	extreme.	
1352	BAD_BINDING	A figure value is bound to a control that does not exist.
1353	BAD_GUARD	A branch condition names learner state the engine does not record.
1354	COMPILE	The lesson does not compile into a state machine.
1355	DEAD_END_REMEDIATION	A wrong-answer detour never returns to the checkpoint it came from.
1356	GOAL_NO_CONTROL	A goal is set on a value no control on the beat writes, so the gate never opens.
1357		
1358	GOAL_UNREACHABLE	A goal cannot be met by any value its control can take.
1359	INLINE_FN	Beat parameters are not plain data and would not survive serialization.
1360	MISSING_PARAM	A beat lacks a parameter its type requires.
1361	NAN_SAMPLE	Sampling the figure at some time or control extreme fails or yields NaN.
1362	NO_EXIT	An interactive beat has no way to continue.
1363	NO_TERMINAL	A learner who reaches this beat can never reach an ending.
1364	OFF_PALETTE	A colour is not a named palette colour, so no theme can re-map it.
1365	OUT_OF_STAGE	A figure element leaves the stage at some time or control extreme.
1366	REPLAY	Replaying the lesson from its own event log does not reproduce the same run.
1367		
1368	RESERVED_PROP	A control collides with a property the renderer reserves.
1369	UNCHECKED_VIZ	Reported, not refused: a beat poses hand-written figure code that no check can sample.
1370		
1371	UNKNOWN_PROP	A visual is passed a property it does not accept.
1372	UNKNOWN_VISUAL	A beat names a visual that is neither registered nor built.
1373		
1374	Pedagogical shape (<code>author/pedagogy.ts</code>). Is it an interactive lesson: it asks, re-teaches, forks, and	
1375	invites questions.	
1376	ACT_NO_CHECKPOINT	A whole act goes by without one.
1377	CHECKPOINT_NO_SKILL	A checkpoint that grants no skill, so nothing downstream can fork on having passed it.
1378		
1379	EVERY_DEMO_GATED	Every demo is gated, so there is nowhere the learner may simply look and move on.
1380		
1381	FORK_ARMS_EQUAL	A fork whose arms land in the same place, which is not a fork.
1382	FORK_ON_UNTAUGHT	A fork on a skill or misconception no checkpoint in this lesson can ever set.
1383	FUSSY_GOAL	A gate the learner has to thread a needle to open.
1384	GATE_NO_TASK	A gated demo that never says what to do.
1385	MCQ_NOT_ONE_CORRECT	An mcq without exactly one correct choice.
1386	MCQ_NO_MISCONCEPTION	Wrong choices that carry no misconception, so a wrong answer teaches the lesson nothing.
1387		
1388	NOTHING_PLAYS	Nothing carries advance, so nothing plays: the "video" is a deck the learner clicks through.
1389		
1390	NO_ADAPTIVE_PATH	Every learner gets the same beats in the same order, whatever they showed they knew.
1391		
1392	NO_ASK	Nowhere invites a question, so the tutor is wired up and unreachable.
1393	NO_CHECKPOINT	Nothing in the lesson asks the learner to commit to an answer.
1394	NO_GATE	Every demo in the lesson can be skipped without touching anything.
1395	NO_INTERACTION	Nothing the learner can touch.
1396	NO_NARRATION	A beat with no narration — in a narrated video, a silent beat is a gap in the soundtrack.
1397		
1398	NO_REMEDIATION	A checkpoint whose wrong answer walks on rather than detouring.
1399	NO_WRONG_FEEDBACK	A checkpoint that says nothing when the answer is wrong.
1400	REMEDIATION_NO_RETURN	A detour that never comes back to the question it was sent from.
1401	SUCCESS_NO_GOAL	A success message on a demo with no goal — it can never be rendered.
1402		
1403	UNREACHABLE	A beat nothing routes to.

Cohesion (`author/cohesion.ts`). Does it read as one lesson rather than k acts written apart.

CUE_UNANCHORED	A cue is anchored to a phrase the beat's narration does not say, so it can never fire.
HOOK_UNANSWERED	The lesson never comes back to the question it opened with.
NARRATION_ECHOES_TEXT	A beat's narration is its own prose read aloud, rather than what a voice would say over it.
PROMISED_INTERACTION	Prose tells the learner to do something the beat gives them no way to do.
REPEATED_LINE	Two beats say the same thing in almost the same words.
UNKNOWN_SYMBOL	Prose tags a symbol the lesson's colour key does not define — the tag is inert.

Visual progression (`author/figure_kit.ts`). Does the lesson use its figure: reveal it in order, and show all of it.

ALL_AT_ONCE	Every beat shows every layer, so nothing is revealed in order.
BAD_BINDING	A figure value is bound to a control that does not exist.
LAYER_UNUSED	A layer the figure draws is never shown by any beat.
NEVER_COMPLETE	No beat ever shows the whole diagram, so the recap never assembles it.
NEVER_POINTS	The figure declares named parts and no beat ever highlights one.
NO_FIGURE	A figure was built and no beat poses it.
OFF_PALETTE	A colour is not a named palette colour, so no theme can re-map it.
OUT_OF_STAGE	A figure element leaves the stage at some time or control extreme.
OVERLOADED_OPENING	The first figure beat already shows most of the layers.
UNPACED_BEAT	A beat brings up several new layers at once with no narration cue pacing them.

D.7 EDITOR

The editor is the only stage that reads the whole lesson, and it exists because parallel writing has one cost: no writer has read the lesson. It receives the lesson in playing order and the cohesion findings, and returns a list of edits, each a beat, a prose field, and new text. It may change prose only; structure, goals, skills, misconceptions, and the figure are refused. Every narration cue phrase must survive an edit word for word, because the figure is listening for it. The edits are applied and the lesson rechecked: the edits are refused if they introduce any mechanics or shape error, break a narration cue, or raise the number of cohesion findings. Accepted edits are folded into the act specifications once, so later stages and rebuilds start from the edited text.

D.8 TUTOR BRIEF

This stage writes the subject-matter half of the runtime tutor's system prompt: what the lesson establishes, so the tutor cannot contradict it; what the figure can and cannot show, so the tutor builds a new figure rather than describing one the learner cannot see; and where the lesson's scope ends. It also writes a one-paragraph fallback shown when no model is available. The brief must be at least 300 characters and contain no markdown headings; the fallback must be 80 to 900 characters.

D.9 REVIEW AGENT

Every earlier stage sees part of the lesson; the Review Agent sees the lesson itself. After emission, LESSONAGENT renders a *storyboard*: for every beat, a screenshot of the figure as the beat opens and after each narration cue fires, beside the narration (with each cue's position marked), the on-screen text, and the question or controls. The agent reads the storyboard and every frame, looking for faults no check can see: a picture that contradicts the words, a thing named but not on screen, the wrong scene for a beat, a demonstration whose control does not visibly drive what the narration says to watch, a remediation that repeats the original explanation, a mislabelled or cluttered figure, and questions with more than one right answer or untrue feedback.

It may edit any beat's prose, cues, and figure pose; the figure page; and the tutor brief, so the tutor does not contradict a fact it corrected. It may not change beat ids, types, transitions, skills, misconceptions,

or goals. After each batch of edits it runs a rebuild script that reassembles the lesson with every model stage reused and no model called, reruns the four checks and the oracle, emits the lesson, and re-renders the storyboard. If the agent’s final lesson has errors, it is discarded and the last lesson that passed is kept.

D.10 AT PLAY TIME: THE LIVE TUTOR

The emitted lesson carries the tutor brief. When a learner asks a question, the runtime tutor, a separate model, receives a fixed teaching brief, the engine’s command reference, and the lesson’s tutor brief as its system prompt (§D.11.6). Each turn it also receives an observation of the learner’s state, and it answers by issuing engine commands: speak, focus or annotate part of the figure, move a control, revisit or jump to a beat, add a beat, or do nothing. The engine adjudicates every command and reports refusals in the next observation.

D.11 PROMPT DESIGN AND DISCLOSURE

This section summarizes the prompts used by the structured authoring stages and the runtime tutor. Rather than reproducing one task-specific run in full, we report the information supplied to each stage, its output contract, and the constraints that determine acceptance. The complete verbatim prompt templates, JSON schemas, and the captured prompts for the pendulum example are included in the anonymized code release. Prompt templates are generated from the same registries used by the runtime, so the advertised beat types, drawing primitives, guards, and command vocabulary remain synchronized with the implementation.

D.11.1 SHARED PROMPT CONSTRUCTION

Each authoring prompt contains four task-specific inputs: (1) the learning objective, (2) a scene plan, (3) implementation and presentation constraints, and (4) numbered source units extracted from the instructional material. Structured stages additionally receive only the artifacts they need from prior stages. The planner receives no generated lesson content; act writers receive one act and the figure’s interface; the editor receives the assembled lesson; and the tutor-brief stage receives the finalized lesson. This division prevents parallel writers from silently changing lesson topology.

All structured stages return one forced tool call whose input schema is also the stage’s output schema. On rejection, the implementation returns the submitted object and the deterministic failure messages, then requests a complete corrected object. The model is not shown a scalar reward. The repair budget, provider retry policy, and model settings are listed in Table 9.

D.11.2 PLANNER

The planner is instructed to decide lesson structure without writing instructional prose. Its output is a `LessonPlan` containing acts, typed beats, source-unit coverage, transitions, remediation paths, and skills granted by checkpoints. It owns all edges because act writers run independently. The core instruction is:

Plan the lesson as acts and typed beats. For every beat, provide an id, type, one-line intent, and every outgoing edge. Do not write lesson content. Every source unit must be covered exactly once. Every checkpoint must route an incorrect answer through remediation and back to the checkpoint.

The prompt enumerates registered beat types and stock guards directly from the engine. It requires at least one checkpoint, remediation path, meaningful interaction, learner-state fork, gated demonstration, and opportunity to ask the tutor a question. The plan is rejected for invalid coverage, unknown beat types or guards, dangling transitions, unreachable endings, or missing pedagogical structure (§D.6).

D.11.3 PAGE AGENT AND DECLARATIVE FIGURE FALLBACK

The Page Agent receives the task, source figure, accepted plan summary, a minimal contract-compliant example, and a brief defining the browser interface. It writes one self-contained HTML page with no

network access. The page must expose: (1) a scene selector, (2) global handlers for learner controls, (3) an `LS_FIGURE` declaration of scenes, layers, and named parts, and (4) an emphasis handler used by narration cues. The brief asks the agent to run the browser oracle, inspect every rendered scene, and repeat until both the oracle and visual inspection pass.

The design requirements are concise but explicit: each scene must express a distinct instructional idea; geometry and motion must follow the source material; labels must remain legible at the target viewport; narrative text belongs outside the figure; concepts must be revealable in layers; and demonstrations must respond visibly to learner controls. The agent may not load remote assets or draw measurable text on an uninspectable canvas.

When the Page Agent is disabled, one structured call produces the same figure as declarative scene data. The fallback prompt defines the available primitives, binding language, palette, and layer contract. It requires every control extreme to remain finite and on stage, every layer to change visibility, and every control binding to name a declared control.

Representative output schema. The following abridged schema shows the typed contract between figure generation and the downstream act writers. The full schema, including control and scene-node definitions, is provided with the released prompt artifacts.

```
{
  "type": "object",
  "properties": {
    "stage": {"type": "object"},
    "initial": {"type": "array", "items": {"type": "object"}},
    "controls": {"type": "array", "items": {"type": "object"}},
    "defaults": {"type": "object"},
    "layers": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "key": {"type": "string"},
          "label": {"type": "string"},
          "shows": {"type": "string"},
          "nodes": {"type": "array", "items": {"type": "string"}}
        },
        "required": ["key", "label", "shows", "nodes"]
      }
    },
    "symbols": {
      "type": "object",
      "description": "symbol -> {tex, means, dark, light}; shared by figure and prose"
    },
    "notes": {
      "type": "array",
      "items": {"type": "string"},
      "description": "one line per figure part for downstream writers"
    }
  },
  "required": [
    "stage", "initial", "controls", "defaults", "layers", "notes"
  ]
}
```

D.11.4 ACT WRITERS

Each act writer receives the task, one planned act, the source units assigned to that act, and the accepted figure vocabulary. It supplies only the parameters of the beats already planned: narration, on-screen text, questions, choices, feedback, controls, goals, figure poses, and narration cues. The central contract is:

Write the content of the assigned beats. Do not add, remove, rename, or reorder beats, and do not set transitions. Return parameters for every assigned beat. A narration cue must quote an exact phrase from that beat's narration.

The prompt also specifies voice and interaction rules: narration must be spoken rather than copied from on-screen text; every explorable needs a reachable exit; at most one demonstration is gated unless the plan explicitly requests otherwise; wrong choices must identify misconceptions; and

questions must not reveal their own answers. Assembly discards structural fields returned by a writer and checks the act as described in §D.6.

D.11.5 EDITOR, TUTOR BRIEF, AND REVIEW AGENT

The editor receives the complete lesson in playing order and any cohesion findings. It may return a small list of prose-only edits. It is asked to enforce one voice, one term per concept, continuity between acts, spoken narration, and an ending that resolves the opening. It cannot change structure, learner state, figure behavior, skills, goals, or misconceptions. An edit that removes a phrase used by a narration cue is rejected.

The tutor-brief stage writes the subject-matter context used at play time and an offline fallback. Its brief must state what the lesson establishes, what the current apparatus can and cannot show, and where the lesson’s scope ends. It is explicitly told not to invent facts or unseen figure capabilities.

The Review Agent receives the rendered storyboard, every storyboard frame, the task, source figure, assembled acts, figure page, and tutor brief. It checks visual–verbal alignment, figure correctness, legibility, control behavior, question validity, remediation quality, and tutor consistency. It may edit prose, cues, figure poses, the figure page, and the tutor brief, but not lesson topology or learner-state semantics. Every revision is rebuilt and rechecked; a failing revision is discarded in favor of the last passing lesson.

D.11.6 RUNTIME TUTOR

At play time, the tutor receives a fixed teaching policy, the lesson-specific tutor brief, the engine command reference, and a per-turn observation of learner state. It is instructed to answer briefly, act through engine commands, reuse existing beats and figures when possible, and avoid interrupting a learner who is not stuck. Available operations include speaking, revisiting or moving to a beat, setting controls, focusing or annotating the figure, adding or patching a beat, and changing a transition. Commands are adjudicated atomically by the engine: if any command in a turn is invalid, none are applied and the tutor receives the refusal in the next observation.

The fixed policy emphasizes three constraints: never invent a beat identifier, never strand the learner without a reachable ending, and never assume a command succeeded before reading the next observation. Runtime-authored beats must be pure JSON so that the complete interaction remains replayable.

D.11.7 REPRODUCIBILITY

The released prompt artifacts include the exact template version, generated registry sections, output schema, task inputs, accepted stage outputs, refusal messages, model identifier, and capture timestamp. For the reported experiments, prompts were not manually edited between tasks. Task-specific source text and scene plans vary, while the stage contracts and generated engine vocabulary remain fixed. This separation allows a run to be audited without embedding repeated source material and thousands of lines of task-specific prompt text in the paper.

E FIGURE GENERATION PIPELINE

This section specifies the executable pipeline used to transform a source textbook figure into an interactive educational artifact. Unlike the pairwise benchmark evaluator in Section B, the critic described here is *inside* the optimization loop: its feedback changes later generations and is therefore part of the method rather than an independent final evaluator.

E.1 PIPELINE OVERVIEW

Given a source figure image x , textbook context q , task prompt u , required interaction specification r , and a maximum of T outer attempts, the full pipeline executes

$$P_0 = \text{Plan}(x, q, u, r), \quad H_t = \text{Generate}(x, P_t, H_{t-1}, C_{t-1}), \quad (2)$$

where P_t is a structured interaction blueprint and H_t is executable HTML/JavaScript. Each candidate is rendered before semantic critique:

$$V_t = \text{Verify}(H_t), \quad C_t^{\text{vlm}} = \text{Critic}(x, H_t, \text{shot}(H_t)), \quad (3)$$

and blocking verifier findings are merged into the critic’s generation action items to form C_t . The critic returns five integer scores

$$\mathbf{s}_t = (s_{\text{geometry}}, s_{\text{interaction}}, s_{\text{faithfulness}}, s_{\text{labels}}, s_{\text{concept}}) \in \{1, \dots, 5\}^5,$$

failure modes, generation-level action items, and plan-level action items. The reported development-time overall score is the unweighted mean

$$\bar{s}_t = \frac{1}{5} \sum_{k=1}^5 s_{t,k}. \quad (4)$$

An LLM orchestrator chooses

$$D_t \in \{\text{pass}, \text{fix_plan}, \text{refine_generation}\}. \quad (5)$$

If $D_t = \text{fix_plan}$, the planner revises P_t using critic feedback. If $D_t = \text{refine_generation}$, the plan is retained and the generator edits the previous payload. Blocking verifier errors override `pass` and `fix_plan` and force `refine_generation`. The loop terminates on `pass`, a stage failure, or the attempt limit. Importantly, passage is an orchestrator decision; the implementation does not apply a fixed numerical pass threshold.

E.2 INPUTS AND CONTEXT CONSTRUCTION

Each benchmark task supplies the source image, nearby instructional context, a manually reviewed generation prompt, and a required interaction specification. The interaction specification is held fixed across systems and ablations. In the structured conditions, the planner receives these inputs and expands them into an implementation blueprint; it does not replace the required interaction with a self-selected alternative.

Both tracks share the following interaction vocabulary:

orbit, drag, hover, click, slider, toggle, button, state, animation, equation_input, and code_editor.

Every interaction is represented by a typed identifier and a short statement of what the learner should understand from using it. Type-specific fields record values such as slider ranges, toggle options, state names, equation variables, or the restricted API exposed by a code editor.

E.3 PLANNER BLUEPRINT

The planner converts the image, nearby text, task prompt, and required interaction specification into a compact implementation blueprint. It is instructed to preserve the source’s visible elements and labels, implement every required interaction, and state the pedagogical role of each control. Its common output fields are:

- the concept and key learning insight;
- an exhaustive inventory of visual elements and labels;
- typed interactions, including identifiers, defaults, ranges or states, and what each interaction teaches;
- optional guided-demo steps and their target interaction states; and
- reconstruction notes covering layout, scale, colors, orientation, and formulas required for correctness.

For 3D figures, the blueprint additionally records projection type (perspective, axonometric, or oblique), camera framing, world-space geometry, and depth relationships. For 2D figures, it records a figure type used to route rendering to SVG.js, Chart.js, or Mermaid. Planner output is JSON parsed by the pipeline and then injected into the generator prompt.

E.4 GENERATION AND SCAFFOLD CONTRACTS

Both primary generators fill fixed scaffold regions rather than rewriting the runtime. The required output is:

```
<!-- @FIGURE_UI_BEGIN -->
[HTML inserted inside the scaffold's #ui element]
<!-- @FIGURE_UI_END -->
// @FIGURE_CODE_BEGIN
[non-empty drawing and interaction JavaScript]
// @FIGURE_CODE_END
```

The model must emit both complete marker pairs and no prose or Markdown. The backend rejects missing markers and empty JavaScript, then merges the payload into the versioned scaffold.

3D generation. The Three.js scaffold provides the scene, renderer, camera, controls, resize handling, label helpers, camera-fitting functions, hover/click explanations, equation parsing, and sampled 3D plotting. The generator is instructed to:

- construct every visible element as actual meshes, lines, points, or sprites rather than pasting the source image onto a plane;
- preserve the source’s geometry, projection mechanism, initial view, labels, proportions, and colors;
- build static geometry before setting the camera, labels, and interactions;
- implement every planned control and every guided-demo state;
- use a sandboxed Web Worker, operation limits, and replayable state frames for learner-editable algorithm code.

The initial 3D runtime message injects textbook context, projection guidance, elements, geometry notes, interactions, demo steps, and camera framing from the plan. A refinement call additionally receives the source image, previous rendered screenshot, previous marker payload, all critic scores, failure modes, notes, and action items.

2D generation and deterministic library routing. The 2D scaffold loads SVG.js, Chart.js, Mermaid, and an expression parser. The library is selected from the blueprint:

- Mermaid for `flow_chart`;
- Chart.js for line, scatter, bar, contour, and multi-panel plots;
- SVG.js for other diagrams;
- SVG.js whenever `equation_input` or `code_editor` is requested.

The default state must be a faithful inline replacement for the source, including mark types, axis ranges, data shape, labels, colors, and line styles. Explanatory content is placed in interaction affordances rather than added inside the static figure. A refinement call includes the previous screenshot and payload in addition to the full original generation contract.

E.5 DETERMINISTIC BROWSER VERIFICATION

Verification renders each candidate in a real headless browser before critique. The 3D track evaluates 1366×768 and 1280×800 viewports. Blocking checks include uncaught exceptions, visible error overlays, render-loop liveness when instrumented, GPU draw calls, scene content, non-finite geometry, blank or zero-sized canvas, horizontal overflow, off-screen controls, and equation overflow. Console errors, vertical overflow, clipped controls, and label size/placement/overlap are warnings.

The 2D track evaluates 900×560 and 640×520. It blocks on page errors, absent or nearly blank visual output, horizontal overflow, or a missing or clipped primary visualization. Console messages, excessive height, and an undetected explanatory affordance are warnings. The verifier does not establish scientific correctness, source fidelity, or that every control functions. Verifier infrastructure exceptions currently fail open and allow critique to continue.

E.6 DEVELOPMENT-TIME CRITIC

The critic receives the source image, the verifier’s screenshot, and the generated marker payload. It outputs:

```
{
  "discrepancies": ["0-6 high-impact differences from the source"],
  "failure_modes": ["selected canonical failure modes"],
  "geometry_accuracy": 1,
  "interactivity_usability": 1,
  "faithfulness": 1,
  "label_quality": 1,
  "concept_accuracy": 1,
  "notes": "one concise summary",
  "action_items": ["2-4 generation-level fixes"],
  "plan_action_items": ["0-2 plan-level fixes"]
}
```

The canonical failure vocabulary is:

Depth-Wrong, Missing-Labels, Wrong-Primitives,
Interaction-Broken, Interaction-Missing, Camera-Wrong,
Scale-Wrong, Color-Wrong, Hallucination, and
Concept-Misunderstood.

For 2D figures, depth and camera failure modes are removed and geometric accuracy is interpreted as structural fidelity of axes, marks, topology, proportions, and line styles. Scores are rounded and clamped to [1, 5]. Generation action items target implementation defects; plan action items target missing concepts, elements, or interaction structure.

E.7 ORCHESTRATOR PROMPT AND REFINEMENT POLICY

The routing prompt is:

```
You are the orchestration agent for an iterative figure-generation loop.

You will receive the critic evaluation, including failure modes, scores, notes, and action
items.
Decide which part of the system is most responsible:
- planner: the interaction plan or conceptual decomposition is wrong, incomplete, or missing
  the right elements
- generator: the plan is close to right, but implementation, labels, layout, interactivity, or
  execution is broken
- none: the figure is good enough to stop iterating

Use failure modes and action items as primary evidence. Do not decide from raw score
thresholds alone.
Prefer planner for concept misunderstanding, missing core elements, wrong primitives, or wrong
interaction structure.
Prefer generator when the concept is right but implementation details or controls are broken.

Return ONLY:
{
  "next_step": "pass | fix_plan | refine_generation",
  "rationale": "one concise sentence"
}
```

Invalid or unknown orchestrator output defaults to `refine_generation`. In the 2D loop, an orchestrator API failure also defaults to generation refinement. Blocking verifier errors take precedence over the LLM route. Warnings do not block passage.

Plan refinement receives the previous plan, plan-level and generation-level action items, overall and concept scores, and failure modes. Generation refinement receives the original generation prompt plus the previous payload, rendered screenshot, scores, failure modes, notes, and concrete action items.

E.8 EXPERIMENTAL CONDITIONS AND STOPPING

The three figure-generation conditions differ only in which stages are enabled:

Baseline. One direct generation call from the source image, task prompt, and fixed interaction specification; no planner and no refinement.

No Iteration. The planner first produces the structured blueprint, followed by one generation call; no critic-driven refinement is allowed.

Full Pipeline. Planning is followed by generation, browser verification, multimodal critique, and orchestration for at most three generation attempts (the initial candidate and up to two revisions).

All three conditions use the same source figure, authored task prompt, required interaction specification, and evaluated backbone. Thus the ablation changes pipeline structure rather than task content. A structured-pipeline run stops when the orchestrator returns `pass` and no blocking verifier error is present, or when the three-attempt budget is exhausted.

E.9 METHODOLOGICAL SCOPE

The deterministic verifier establishes mechanical renderability and selected layout invariants, not conceptual truth. The development-time critic is not an independent evaluator because it participates in optimization. A bounded refinement loop guarantees termination, not successful correction. We therefore report mechanical completion, verifier status, orchestrator acceptance, and independent pairwise quality evaluation as separate outcomes.

AI Use Statement. We used large language models and AI coding agents throughout the research and writing process. Specifically, AI tools were used to aid and polish writing, retrieve and discover related work, support research ideation and execution, draft sections of the paper, and generate synthetic datasets used in our experiments. Details of these uses, including the roles of AI systems in our research pipeline and dataset generation, are described in the paper and appendix. All generated content, experimental procedures, analyses, and reported results were reviewed and verified by the authors.