LESSONWORLD: AGENTIC ENVIRONMENT FOR INTERACTIVE EDUCATIONAL CONTENT GENERATION

Anonymous authors
Paper under double-blind review

Demos, Code and Data : LessonWorld.fly.dev

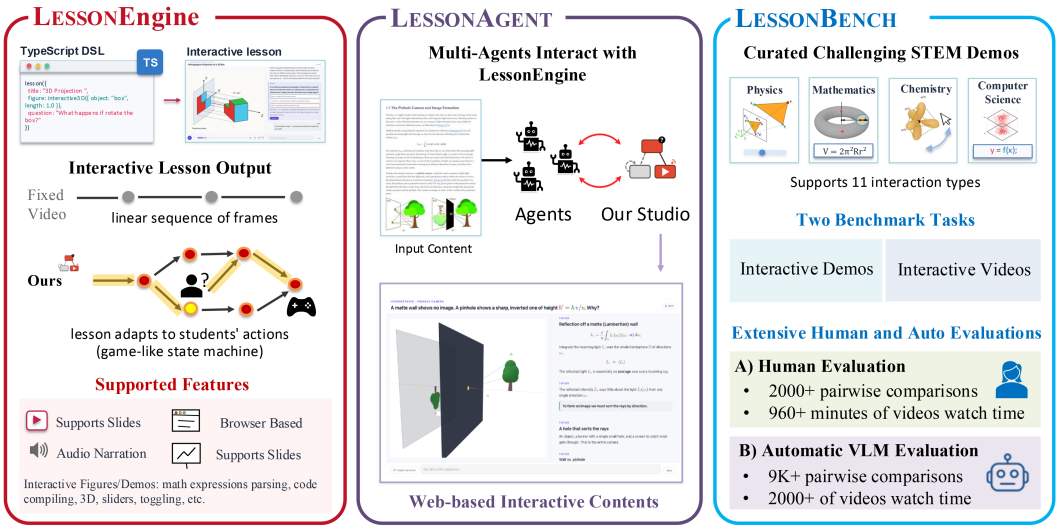

Figure 1: **LessonWorld Overview**. Unlike conventional educational content generation that produces fixed artifacts, LESSONENGINE represents lessons as interactive, stateful environments that respond to learner actions. LESSONAGENT uses a distributed agentic pipeline to transform instructional resources 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 after generation. 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.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

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 550 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.

			Output Modality				Properties		
Method	**Venue**	**Backend**	Video	Figure	Web	Slides	RT	Interact.	**Evaluation**
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 (LaTeX)	✓	×	×	×	×	×	VLM + Human
AutomaTikZ (Belouadi et al., 2024)	ICLR'24	TikZ (LaTeX)	×	✓	×	×	×	×	Auto.
LessonWorld (Ours)	–	**LessonEngine**	✓	✓	✓	✓	✓	✓	**Human + VLM Judge**

Table 1: Comparison with recent systems for agentically 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.

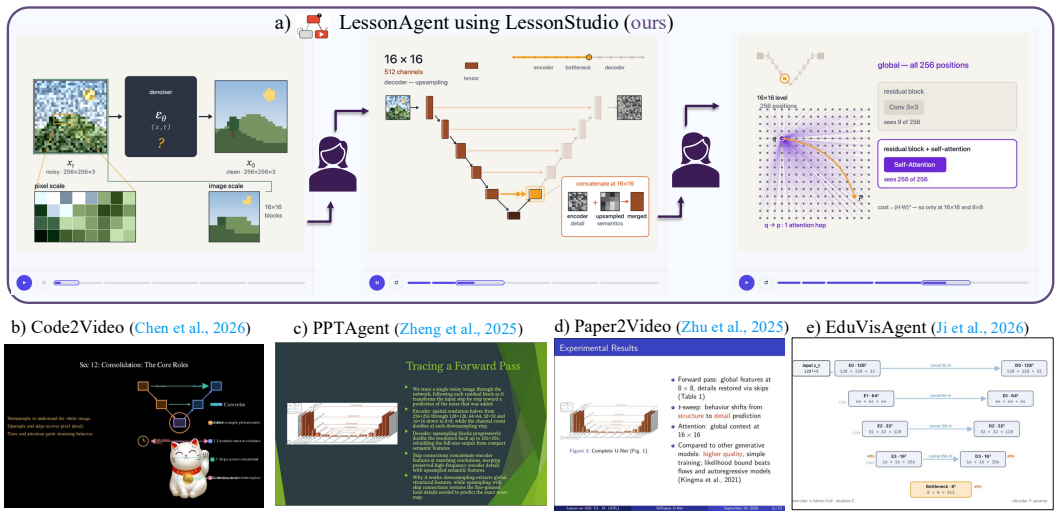

Figure 2: **From static educational artifacts to interactive lessons.** Our LESSONAGENT uses LESSONENGINE to generate interactive lessons that expose underlying concepts and adapt to learner actions at runtime. In contrast, existing methods primarily generate static educational videos (Chen et al., 2026b),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

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× 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 transforms textbook chapters, handouts, and figures into executable lessons using LESSONENGINE. Rather than generating a lesson in one invocation, it decomposes authoring into specialized stages over the same typed representation, allowing lesson structure to be planned first, components to be generated in parallel, and failures to be checked automatically.

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. Specialized agents then generate components from the shared plan: a Figures Agent creates interactive browser-based figures with visual structure, controls, guided states, and learning goals, while Act writers independently generate narration, on-screen text, questions, feedback, and figure cues. They may modify content but not lesson structure, preserving globally consistent transitions. Full details appear in Appendices D and C.

Verification, assembly, and repair. Artifacts are checked before and after assembly. Deterministic checks detect failures in *mechanics*, *instructional structure*, *cohesion*, and *visual progression*, including invalid states, repeated narration, and elements appearing before introduction. Failed components are regenerated from the corresponding failure signal rather than regenerating the full lesson. Visual and temporal properties, including figure legibility, cue correctness, and narration–visual alignment, are checked on rendered outputs.

The verified components are assembled into a single LESSONENGINE lesson. An Editor, the only stage with access to the complete lesson, resolves cross-act inconsistencies such as terminology mismatches, broken transitions, or inadequate recaps. A Tutor stage then produces the brief for the runtime tutor. Finally, the Review Agent inspects the rendered storyboard and can repair act specifications or figure pages when assembly reveals issues. Revisions are retained only if they continue to pass verification. Details appear in Appendix C.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 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: a *figure-level* benchmark isolating the quality of individual interactive artifacts our pipeline generates, and a *lesson-level* benchmark evaluating the full authored session against prior video- and slide-generation systems.

Figure-Level Benchmark. Using the 250-task figure-level set described in Section 4.1, we fix the prompt and interaction specification per task so that differences in model output reflect generation quality rather than task ambiguity.

Lesson-Level Benchmark. To evaluate LessonWorld as a complete system rather than a figure generator in isolation, we compare full lesson sessions against prior video- and presentation-generation pipelines: `NotebookLM`, `Manimator`, `Code2Video`, `PPTAgent`, `Paper2Video`, as well as `EduVisAgent`. Where a system does not natively support interactivity (all baselines except LessonWorld), we evaluate its rendered output as the static degenerate case described in Section 1. This isolates the comprehension and usability value LessonWorld's live, agent-directable representation adds on top of content quality that is otherwise held comparable across systems.

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 A.

We use the LLM-as-Judge, blind pairwise comparison, and pairwise ranking protocol described in Section 4.4 to score both benchmarks, so that figure- and lesson-level 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 complete 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.

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

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

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.

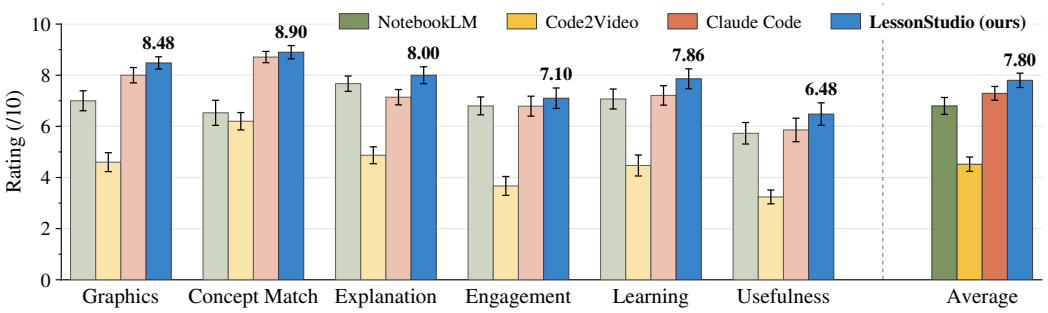

Figure 3: **Human evaluation of lessons generated by each system.** Participants rated each lesson on six dimensions on a 1–10 scale; higher is better. *Concept Match*: how well graphics correspond to the concept; *Learning*: whether participants understood the concept better afterwards; *Usefulness*: 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.

4.5.2 Results for Ablation Studies

We conduct three ablation studies on the interactive figure generation component. We report results with three evaluation set-ups.

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

Human evaluation of Generated Figures. Figure 4 summarizes human preferences across the four subject domains. Participants compare figures from two systems side by side and select the preferred output. LessonStudio achieves the highest average preference rate, with particularly strong results in Computer Science and Chemistry.

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

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

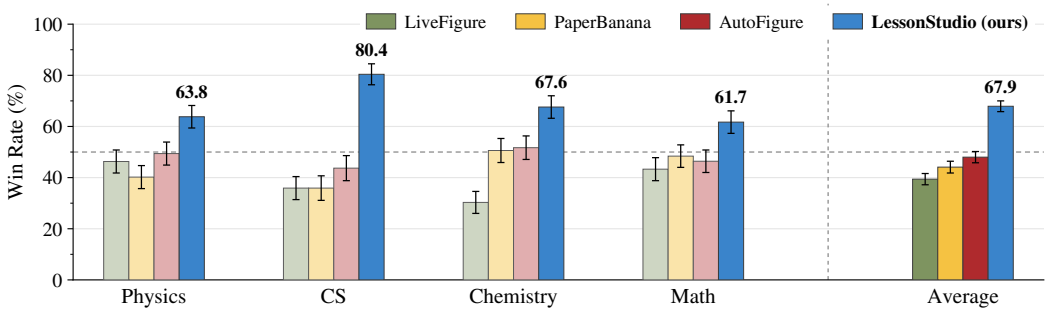

Figure 4: **Human evaluation of generated figures.** Participants compared figures from two systems side by side and chose the one they preferred. We report each system's pairwise win rate (%) per subject domain and averaged across domains. The dashed line at 50% marks equal preference. LESSONENGINE 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

LESSONWORLD suggests that the representation used to generate educational content can matter as much as the generation model itself. Instead of generating a rendered artifact and stopping there, LESSONWORLD maintains an executable lesson representation where instructional structure, learner state, and interaction behavior remain explicit. Across our evaluations, this leads to gains at both the figure and lesson levels: the full structured pipeline achieves a 69.2% average win rate with GPT-5.5, compared with 60.1% for the corresponding 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. Interactscience: 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. Manimator: 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 Wenhu Chen. TheoremExplainAgent: 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 Figure-Level Pairwise Evaluation Protocol **13**

A.1 Shared Pairwise Judge Instruction . . . 13
A.2 Dimension Prompts . . . 13
A.3 Runtime Message Templates . . . 15
A.4 Aggregation Prompt . . . 16
A.5 Bradley–Terry Ranking . . . 16
A.6 Deterministic Render Validation . . . 16
A.7 Evaluation Reproducibility and Scope . . . 16

B Human Study Protocol **17**

B.1 Systems and materials . . . 17
B.2 Blinding and counterbalancing . . . 17
B.3 Study platform . . . 17
B.4 Participants and procedure . . . 18
B.5 Questionnaire . . . 18
B.6 Behavioural logging . . . 19
B.7 Analysis . . . 20

C LESSONAGENT: Full Details **20**

C.1 Pipeline at a glance . . . 20
C.2 Structured stages and the repair protocol . . . 22
C.3 Planning . . . 22
C.4 Figures: the Page Agent . . . 22
C.5 Act writers . . . 23
C.6 Assembly and the four checks . . . 23
C.7 Editor . . . 25
C.8 Tutor brief . . . 25
C.9 Review Agent . . . 25
C.10 At play time: the live tutor . . . 26
C.11 Prompts . . . 26

D Figure Generation Pipeline **58**

D.1 Pipeline Overview . . . 58
D.2 Inputs and Context Construction . . . 59
D.3 Planner Blueprint . . . 59

D.4 Generation and Scaffold Contracts 59
D.5 Deterministic Browser Verification 60
D.6 Development-Time Critic 60
D.7 Orchestrator Prompt and Refinement Policy 61
D.8 Experimental Conditions and Stopping 61
D.9 Methodological Scope 62

A 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.

A.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.

A.2 Dimension Prompts

A.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.
```

A.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.
```

A.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.
```

A.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.
```

A.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.

A.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.
```

A.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.

A.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

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

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.

A.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.

A.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.

B 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.

B.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 `gemini-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.

B.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.

B.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 $\lim \sin x/x = 1$ 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.

B.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.

B.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.

Item	Wording	Response
Per version (asked of each of A–D)		
`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
Comparison (once per topic)		
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
Study site		
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 (§B.7).

B.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.

B.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.

C 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 C.11 reproduces every prompt verbatim, as sent for one real run, the pendulum lesson whose state machine appears in Figure **??**.

C.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.

Setting	Value
Structured stages (planning, figure, act writers, editor, tutor)	
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 (§C.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
Page Agent	
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 `npx tsx`, `node`, `ls`; 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
Review Agent	
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, `npx tsx`, `ls`
Runtime tutor (at play time, not authoring)	
Model	`claude-opus-5`, adaptive thinking, at most 12,000 output tokens per turn

Table 9: Model and agent settings.

C.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.
```

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.

C.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 §C.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.

C.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.

C.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.

C.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 `topology`, 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.

Mechanics (`compose/verify.ts`). Does the lesson run: compile, terminate, and render at every extreme.

`BAD_BINDING`	A figure value is bound to a control that does not exist.
`BAD_GUARD`	A branch condition names learner state the engine does not record.
`COMPILE`	The lesson does not compile into a state machine.
`DEAD_END_REMEDIATION`	A wrong-answer detour never returns to the checkpoint it came from.
`GOAL_NO_CONTROL`	A goal is set on a value no control on the beat writes, so the gate never opens.
`GOAL_UNREACHABLE`	A goal cannot be met by any value its control can take.
`INLINE_FN`	Beat parameters are not plain data and would not survive serialization.
`MISSING_PARAM`	A beat lacks a parameter its type requires.
`NAN_SAMPLE`	Sampling the figure at some time or control extreme fails or yields NaN.
`NO_EXIT`	An interactive beat has no way to continue.
`NO_TERMINAL`	A learner who reaches this beat can never reach an ending.
`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.
`REPLAY`	Replaying the lesson from its own event log does not reproduce the same run.
`RESERVED_PROP`	A control collides with a property the renderer reserves.
`UNCHECKED_VIZ`	Reported, not refused: a beat poses hand-written figure code that no check can sample.
`UNKNOWN_PROP`	A visual is passed a property it does not accept.
`UNKNOWN_VISUAL`	A beat names a visual that is neither registered nor built.

Pedagogical shape (`author/pedagogy.ts`). Is it an interactive lesson: it asks, re-teaches, forks, and invites questions.

`ACT_NO_CHECKPOINT`	A whole act goes by without one.
`CHECKPOINT_NO_SKILL`	A checkpoint that grants no skill, so nothing downstream can fork on having passed it.
`EVERY_DEMO_GATED`	Every demo is gated, so there is nowhere the learner may simply look and move on.
`FORK_ARMS_EQUAL`	A fork whose arms land in the same place, which is not a fork.
`FORK_ON_UNTAUGHT`	A fork on a skill or misconception no checkpoint in this lesson can ever set.
`FUSSY_GOAL`	A gate the learner has to thread a needle to open.
`GATE_NO_TASK`	A gated demo that never says what to do.
`MCQ_NOT_ONE_CORRECT`	An mcq without exactly one correct choice.
`MCQ_NO_MISCONCEPTION`	Wrong choices that carry no misconception, so a wrong answer teaches the lesson nothing.
`NOTHING_PLAYS`	Nothing carries `advance`, so nothing plays: the ”video” is a deck the learner clicks through.
`NO_ADAPTIVE_PATH`	Every learner gets the same beats in the same order, whatever they showed they knew.
`NO_ASK`	Nowhere invites a question, so the tutor is wired up and unreachable.
`NO_CHECKPOINT`	Nothing in the lesson asks the learner to commit to an answer.
`NO_GATE`	Every demo in the lesson can be skipped without touching anything.
`NO_INTERACTION`	Nothing the learner can touch.
`NO_NARRATION`	A beat with no narration — in a narrated video, a silent beat is a gap in the soundtrack.
`NO_REMEDIATION`	A checkpoint whose wrong answer walks on rather than detouring.
`NO_WRONG_FEEDBACK`	A checkpoint that says nothing when the answer is wrong.
`REMEDIATION_NO_RETURN`	A detour that never comes back to the question it was sent from.
`SUCCESS_NO_GOAL`	A `success` message on a demo with no `goal` — it can never be rendered.
`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.

C.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.

C.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.

C.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.

C.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 (§C.11.8). 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.

C.11 Prompts

Every prompt below is exactly what the model or agent received for the pendulum lesson (request `p12`). The structured stages were captured by running each stage with a stub model that records its request and stops. The act-writer prompt is for the *Angle demo* act. The Page and Review Agent briefs are the files the run wrote for them. Sections in capitals inside the prompts (beat types, drawing vocabulary, pedagogy rules) are generated from the engine's registries, so the prompts always describe the engine as it is. For typesetting, lines are wrapped at 92 characters and a few non-ASCII symbols are spelled out (e.g. `theta`, `->`, `-`); nothing else is changed.

C.11.1 Planning

System prompt.

```
You are planning an interactive lesson. It plays like a short narrated film and STOPS,
over and over, to hand the learner something to do -- a slider to push, a question to
answer, a  choice about where to go next. You are not writing it. You are deciding its shape.

WHAT YOU PRODUCE is a LessonPlan: acts, and inside each act an ordered list of beats. For

every beat you give an id, a registered TYPE, a one-line INTENT (the assignment its writer will
work from), the figure it poses if any, and every EDGE leaving it. You do not write a word of content.

WHY THE SPLIT: act writers run in parallel and cannot see each other. If they owned the edges,
two of them would point at the same id and a third would rename one, and the lesson would not
compile. So the topology is yours and only yours -- the assembler merges your edges OVER  whatever
a writer returns, and the edge params are exactly: onWrong, onTimeout, then, else, exits,  whenRef.

## THE PLAN OBJECT
  id        a lowercase slug -- it becomes a directory and a URL
  title     the lesson's title, as a learner would see it
  version   1
  draftUnits  how many units the draft has (you are told)
  acts[]    {id, title, objective, covers:[unit ids], beats:[...]}
  beats[]   {id, type, intent, next, topology?, viz?, grants?}
            next: omit to fall through to the following beat, a beat id to jump, null to
                END.
            topology: {onWrong?, then?, else?, whenRef?, exits?} -- edges only, never
                content.
            viz: name the figure this beat poses (`declarative` for the built figure), or
                omit.
            grants: on a checkpoint, the skill a correct answer earns. A `branch` on
              `stock.mastered:<skill>` is only real if some beat grants that exact string --
                  and
              the two are nearly always in different acts, whose writers cannot see each
                  other.
              So both ends are yours: name the skill on the checkpoint AND in the guard.

## COVERAGE
  Every act lists the draft unit ids it is accountable for in `covers`. Every coverable unit
      must
  be claimed by exactly one act -- an unclaimed unit is a scene the lesson silently dropped,
      and it
  is measured and reported. Claim in order; do not reorder the draft.

## BEAT TYPES (the `type` + `params` of an `addBeat` spec)
  explain
```

```
  Prose (markdown + $math$) with an optional visual. The plainest beat; `say` is sugar for
      one.
    * text             markdown + $math$; the words the learner reads
      narration        spoken variant, synthesized on demand
      viz              {name, props?, persistent?} -- a registered visual to pose
      advance          "narration" (move on when the spoken clip ends) or ms to dwell -- a
          run of beats carrying it plays like a film and stops at the first one that does
          not
      exits            [{label, to}] -- offer SEVERAL ways out instead of one Continue; `to:
          null` ends the lesson
      e.g. {"text":"The rays cross **at** the hole, so the image arrives inverted."}
  mcq
    A multiple-choice checkpoint. Marks the beat a checkpoint, so it scores and gates
        progress.
    * prompt           the question; markdown + $math$
    * choices          [{text, correct?, misconception?}] -- exactly one should be `correct:
        true`
      correctFeedback  prose shown after a right answer (default 'Correct!')
      wrongFeedback    prose shown after a wrong answer
      onWrong          beat id to detour to on a wrong answer
      skill            a correct answer raises mastery of this skill name
      narration        spoken variant, synthesized on demand
      e.g. {"prompt":"Doubling $u$ while holding $v$ fixed does what to the
          image?","choices":[{"text":"Halves it","correct":true},{"text":"Doubles
          it","misconception":"m-proportional-to-u"}],"correctFeedback":"Right -- $m=v/u$,
          so $u$ is in the denominator."}
  branch
    An invisible fork: routes to `then` when the named guard passes, `else` otherwise.
        `whenRef` must name a guard the LESSON already registered -- you cannot define one,
        since runtime-authored beats are pure JSON. Prefer `explain` with `exits` to offer
        the learner a choice; use `branch` only to reuse a guard the lesson exposes.
    * whenRef          name of an already-registered guard
    * then             beat id taken when the guard passes
    * else             beat id taken when it does not
      e.g. {"whenRef":"masteredMagnification","then":"hard-q","else":"recap"}
  scene
    A NEW declarative figure, drawn and animated from primitives. Pure JSON, so you can
        author one for a question the lesson never anticipated -- see the DRAWING vocabulary
        for node kinds, animatable properties and easings.
    * storyboard       {duration, initial: SceneNode[], tweens: Tween[], stage?: {w,h},
        camera?: CameraKey[]}
      narration        spoken script; plays alongside the motion, not synchronized to it
      e.g. {"storyboard":{"duration":1200,"stage":{"w":400,"h":200},"initial":[{"id":"ray","
          kind":"line","x":20,"y":40,"x2":380,"y2":160,"stroke":"#f59e0b","opacity":0},{"id"
          :"cap","kind":"label","x":20,"y":175,"text":"one ray, one direction","size":18}],"
          tweens":[{"target":"ray","property":"opacity","to":1,"start":0,"duration":600,"eas
          ing":"smooth"}]}}
  freeResponse
    A typed-answer checkpoint, graded against `accept` after normalization (case, spacing,
        articles, plurals; numbers by value; the phrase inside a short answer). Only for
        answers with a short canonical form -- use `mcq` otherwise.
    * prompt           the question; markdown + $math$
    * accept           [string] -- every spelling you will accept; matched after
        normalization
      hint             shown as the wrong-answer feedback when `wrongFeedback` is absent
      correctFeedback  prose shown after a right answer
      wrongFeedback    prose shown after a wrong answer
      onWrong          beat id to detour to on a wrong answer
      skill            a correct answer raises mastery of this skill name
      misconception    a wrong answer bumps this misconception's strength
      e.g. {"prompt":"With $u=10$ and $v=5$, what is the magnification
          $m$?","accept":["0.5","1/2",".5"],"hint":"$m = v/u$."}
  explorable
    An INTERACTIVE demo: controls the learner drags, wired to a visual. Two ways to name the
        visual -- an existing registered one (see the VISUALS list for names and accepted
        props), or `{name:"declarative"}` with a storyboard whose numbers are BINDINGS to
        the control keys, which is how you build a brand-new interactive figure without
        registering code.
    * controls         [{key, label, kind: slider|toggle|button|choice|text, min?, max?,
        step?, unit?, options?, placeholder?, suggestions?, rows?}] -- `text` is a typed
        field (a formula r = f(theta)); its value is the string, and `suggestions` [{label,
        value}] are one-click alternatives. For CODE give `rows` (e.g. 8): it becomes a
        multi-line editor
    * viz              {name, props?, persistent?} -- control values are merged over
        `props`, so a control named `u` sets prop `u`
      defaults         {key: number|boolean|string} -- starting values (else sliders start
          at min, toggles false, text empty)
      note             prose shown beside the demo
      goal             {key, equals?|min?|max?, tolerance?} -- hide Continue until the
          learner reaches it
      task             what to do; the instruction, shown until a `goal` (if any) is met
```

```
      success          shown once the `goal` is met, in place of `task`
      ask              true to show a free-text question box on this demo
      narration        spoken variant, synthesized on demand
      e.g. {"controls":[{"key":"hole","label":"Hole width","kind":"slider","min":1,"max":40,
          "step":1,"unit":"px"},{"key":"__next","label":"Continue","kind":"button"}],"defaul
          ts":{"hole":6},"viz":{"name":"declarative","props":{"storyboard":{"duration":0,"st
          age":{"w":400,"h":220},"initial":[{"id":"gap","kind":"rect","x":200,"y":{"$sub":[1
          10,{"$div":[{"$ref":"hole"},2]}]},"w":4,"h":{"$ref":"hole"},"fill":"#fbbf24"},{"id
          ":"blur","kind":"circle","x":330,"y":110,"r":{"$mul":[{"$ref":"hole"},0.9]},"fill"
          :"#38bdf8","opacity":0.5}],"tweens":[]}}},"task":"Widen the hole and watch the
          spot on the wall."}
  (* = required)

## GUARDS (the `whenRef` of a `branch` beat -- you may NAME one, never define one)
  stock.mastered:<skill>          the learner answered a checkpoint carrying that `skill`
      correctly
  stock.misconception:<name>     the learner has tripped that named misconception at least
      once
  stock.scoreAtLeast:<n>        the running score is at least n
  stock.struggling                    the learner model says they are struggling
  Any other name is refused before the lesson is emitted. Prefer `explain` with `exits` when
      the
  LEARNER should choose the path; use a branch only when the lesson should choose for them.

## WHAT MAKES THIS A LESSON (every one of these is checked before the lesson is emitted)
  * ASK. At least one `mcq` or `freeResponse` checkpoint, and one in every act that runs
      three
    beats or more. Put it where the learner has just done something, not at the end.
  * CATCH. Every checkpoint carries `topology.onWrong` pointing at a beat that re-teaches,
      and
    that beat's `next` chain must come BACK to the checkpoint -- the engine re-asks it, so
        the
    loop is a second chance rather than a trap.
  * FORK. The lesson must have at least one real fork: a `branch` on a stock guard, or an
    `explain` with two or more `exits`. A branch's `then` and `else` must differ.
  * COUPLE THE FORK TO THE EVIDENCE. A `branch` on `stock.mastered:<skill>` is only real if
      some
    checkpoint GRANTS that exact string. Put `grants: "<skill>"` on that checkpoint -- the
        plan
    owns both ends, because the fork and the checkpoint are usually in different acts and
        neither
    writer can see the other. A plan whose fork reads a skill nothing grants is refused
        outright.
  * LET THEM DO SOMETHING. At least one `explorable`, and it should be the beat the lesson
      turns
    on rather than a decoration after the explanation.
  * GATE ONE DEMO, NOT ALL OF THEM. A `goal` hides Continue until the learner reaches a
      value, and
    on the beat the lesson turns on that is the difference between doing and skipping. On
        every
    demo it is a toll booth. The rest get a plain `__next` they can press whenever they
        like.
  * INVITE THE QUESTION. At least one beat with `ask: true`, so the tutor is reachable where
      the
    learner is most likely to be stuck.
  * SPEAK. Every rendering beat gets `narration`; a run of beats that carry `advance` plays
      as a
    film and stops at the first one that does not, which is how a narrated video is paced
        here.

## SHAPE OF A GOOD RUN
  * One act per scene in the request, unless two scenes are one idea. 3-6 beats an act.
  * Open with a filmed run: two or three `explain`/`scene` beats that play and land on the
    apparatus. Then stop and hand it over -- that first `explorable` is the lesson.
  * A `Demonstration` scene is an `explorable`. Say in its intent whether it is GATED -- the
      one
    demo the lesson turns on should be, so the learner advances by doing; the others should
        not,
    or the lesson becomes a series of small puzzles about finding the value the text named.
  * Check what the demonstration was for, right after it, while it is still on screen.
  * The remediation beat is not a repeat. It is the same idea shown from the one angle the
      wrong
    answer proves they were missing -- usually by pointing at the figure at a different
        pose.
  * End with the fork: a `branch` on the skill the checkpoint granted, sending a learner who
      has
    it to something genuinely harder and everyone else to the recap. Both arms must end.

## WHAT HAPPENS TO WHAT YOU SEND
  It is compiled by the real engine and run through a battery that has no opinions and no
      mercy:
```

```
  every beat must reach an ending, every `goal` must be attainable on its own control's
      lattice,
  every binding must name a control that exists, every storyboard is sampled at five times
      and at
  every slider extreme for NaN and for nodes off the stage, every guard name must be real,
      and the
  whole thing must survive being replayed from its own event log. Anything it refuses comes
      back
  to you as a list, and you fix it and resend. Aim at passing the first time.
```

User prompt.

```
## THE REQUEST
Create a narrated educational video that teaches the textbook figure
    'CNX_UPhysics_15_05_SimplePend' (university physics volume 1). Target length: 60-120
    seconds of narrated animation.
  Pacing: the request asks for 60-120 seconds of narrated animation.

## LEARNING OBJECTIVE
Identify the forces acting on a simple pendulum and describe how the restoring force depends
    on the displacement angle. Determine the period of a simple pendulum and explain how it
    is dictated by the length of the string and the acceleration due to gravity.

## SCENE PLAN (the spine; transcribe it, do not replace it)
  1. Hook -- Have you ever wondered why grandfather clocks keep such precise time regardless
      of how heavy their swinging weights are? The secret lies in the physics of a simple
      pendulum, where the time it takes to complete one swing depends only on its length and
      gravity.
  2. Construction -- Draw a fixed pivot point at the top of the screen and suspend a string
      of length L from it. Attach a spherical pendulum bob of mass m to the bottom of the
      string, and draw a dashed vertical line down from the pivot to represent the
      equilibrium position. Displace the bob to the right by an angle theta, drawing the arc
      length s, the downward weight vector mg, the tension vector along the string, and the
      restoring force vector -mg sin(theta) pointing back toward equilibrium.
  3. Demonstration -- Drag the pendulum bob to change the initial angle and watch the
      restoring force vector dynamically scale in magnitude. As the angle theta increases
      from 0 to 15 degrees, the component of gravity acting along the arc, -mg sin(theta),
      grows larger, providing a stronger restoring force toward the equilibrium position.
  4. Demonstration -- Adjust a slider to modify the string length and observe how the period
      of the pendulum's oscillation changes. Increasing the string length L from 0.5 meters
      to 2.0 meters causes the period T to increase, as the period is equal to 2 pi times
      the square root of L over g.
  5. Consolidation -- The motion of a simple pendulum is driven by a restoring force from
      gravity that scales with the sine of the displacement angle. For small angles, this
      results in simple harmonic motion where the period is completely determined by the
      length of the pendulum and the local acceleration due to gravity.

## CONSTRAINTS (rules on HOW, not content to teach)
  - Render in 2D unless the figure is inherently 3D. Keep every axis, label and symbol
      legible at 720p; no gratuitous camera motion.
  - Name the physical quantities and their units; keep the geometry quantitatively honest.
  - Each demonstration must actually play out on screen: sweep the parameter through its
      range and show the figure responding.
  - Narration must be synchronised to the visual it describes.

## THE DRAFT -- 13 units. Every one must be claimed by exactly one act's `covers`.
[unit 0]
# UPhysics Simple Pend

[unit 1]
## What the learner should be able to do

### What the learner should be able to do
[unit 2]
Identify the forces acting on a simple pendulum and describe how the restoring force depends
    on the displacement angle. Determine the period of a simple pendulum and explain how it
    is dictated by the length of the string and the acceleration due to gravity.

[unit 3]
## Scene 1 -- Hook

### Scene 1 -- Hook
[unit 4]
Have you ever wondered why grandfather clocks keep such precise time regardless of how heavy
    their swinging weights are? The secret lies in the physics of a simple pendulum, where
    the time it takes to complete one swing depends only on its length and gravity.

[unit 5]
## Scene 2 -- Construction
```

```
### Scene 2 -- Construction
[unit 6]
Draw a fixed pivot point at the top of the screen and suspend a string of length L from it.
    Attach a spherical pendulum bob of mass m to the bottom of the string, and draw a dashed
    vertical line down from the pivot to represent the equilibrium position. Displace the
    bob to the right by an angle theta, drawing the arc length s, the downward weight vector
    mg, the tension vector along the string, and the restoring force vector -mg sin(theta)
    pointing back toward equilibrium.

[unit 7]
## Scene 3 -- Demonstration

### Scene 3 -- Demonstration
[unit 8]
Drag the pendulum bob to change the initial angle and watch the restoring force vector
    dynamically scale in magnitude. As the angle theta increases from 0 to 15 degrees, the
    component of gravity acting along the arc, -mg sin(theta), grows larger, providing a
    stronger restoring force toward the equilibrium position.

[unit 9]
## Scene 4 -- Demonstration

### Scene 4 -- Demonstration
[unit 10]
Adjust a slider to modify the string length and observe how the period of the pendulum's
    oscillation changes. Increasing the string length L from 0.5 meters to 2.0 meters causes
    the period T to increase, as the period is equal to 2 pi times the square root of L over
    g.

[unit 11]
## Scene 5 -- Consolidation

### Scene 5 -- Consolidation
[unit 12]
The motion of a simple pendulum is driven by a restoring force from gravity that scales with
    the sine of the displacement angle. For small angles, this results in simple harmonic
    motion where the period is completely determined by the length of the pendulum and the
    local acceleration due to gravity.

Plan the lesson. Set `draftUnits` to 13 and `id` to "p12".
```

Output schema.

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "string",
      "description": "lowercase slug; use exactly the one you were given"
    },
    "title": {
      "type": "string",
      "description": "the lesson's title as a learner sees it"
    },
    "version": {
      "type": "number",
      "description": "1"
    },
    "draftUnits": {
      "type": "number",
      "description": "how many units the draft has"
    },
    "acts": {
      "type": "array",
      "description": "the acts, in the order they are played",
      "items": {
        "type": "object",
        "properties": {
          "id": {
            "type": "string",
            "description": "short slug, unique in this plan"
          },
          "title": {
            "type": "string"
          },
          "objective": {
            "type": "string",
            "description": "what the learner can do at the end of this act that they could
                not at the start"
          },
```

```
        "covers": {
          "type": "array",
          "items": {
            "type": "number"
          },
          "description": "draft unit ids this act is accountable for"
        },
        "beats": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "id": {
                "type": "string",
                "description": "short slug, unique across the whole plan"
              },
              "type": {
                "type": "string",
                "description": "a registered beat type"
              },
              "intent": {
                "type": "string",
                "description": "one line: the assignment its writer works from"
              },
              "next": {
                "type": "string",
                "description": "beat id to go to, or \"end\" to finish the lesson; omit to
                    fall through to the next beat"
              },
              "viz": {
                "type": "string",
                "description": "name the figure this beat poses -- \"declarative\" for the
                    lesson's own figure; omit for none"
              },
              "grants": {
                "type": "string",
                "description": "checkpoint only: the skill a correct answer grants. A
                    `branch` on `stock.mastered:<skill>` is only real if some beat grants
                    that exact string, and the two are usually in different acts -- so you
                    name it here, on both ends"
              },
              "topology": {
                "type": "object",
                "description": "edges only",
                "properties": {
                  "onWrong": {
                    "type": "string",
                    "description": "checkpoint: beat id to detour to on a wrong answer"
                  },
                  "whenRef": {
                    "type": "string",
                    "description": "branch: the stock guard to fork on"
                  },
                  "then": {
                    "type": "string",
                    "description": "branch: beat id when the guard passes"
                  },
                  "else": {
                    "type": "string",
                    "description": "branch: beat id when it does not"
                  },
                  "exits": {
                    "type": "array",
                    "description": "explain: several ways out instead of one Continue",
                    "items": {
                      "type": "object",
                      "properties": {
                        "label": {
                          "type": "string"
                        },
                        "to": {
                          "type": "string",
                          "description": "beat id, or \"end\""
                        }
                      },
                      "required": [
                        "label",
                        "to"
                      ]
                    }
                  }
```

```
                  }
                }
              },
              "required": [
                "id",
                "type",
                "intent"
              ]
            }
          }
        },
        "required": [
          "id",
          "title",
          "objective",
          "covers",
          "beats"
        ]
      }
    }
  },
  "required": [
    "id",
    "title",
    "version",
    "draftUnits",
    "acts"
  ]
}
```

C.11.2 PAGE AGENT

The agent is started with the single instruction "Read BRIEF.md in this directory and do exactly what it says." Its directory holds the request (`task.txt`), the textbook figure image, the plan summary below (`plan.md`), a minimal example page, and the brief.

Brief (`BRIEF.md`).

```
# Draw the pictures for one interactive lesson

You are writing ONE self-contained HTML file, `figure.html`, in this directory. It is the
    stage of a
narrated, interactive lesson: a voice explains, the learner watches, then drags controls and
    answers
questions. The lesson engine (not you) plays the narration, shows captions, asks the
    questions and
decides which picture is on screen. Your page draws the pictures and responds to controls.

## Read first
- `task.txt` -- the request: objective, scene plan, constraints, source text. The
    physics/maths is in here.
- `figure.jpg` -- the textbook figure the lesson teaches. LOOK at it before you start.
- `plan.md` -- what each scene is for, and the beats the lesson will play over your
    pictures.
- `example.html` -- a minimal page that follows the contract exactly. Copy its skeleton;
    replace
  every drawing. It is deliberately plain -- yours must not be.

## What makes it good (this is the bar)
1. **A distinct visual idea per scene.** 5 scenes, one picture each, chosen for what that
   scene has to make someone feel or see. A hook opens on the question in the world (a
       clock, a
   bridge, a thrown ball) -- not on the finished textbook diagram. A construction builds the
       diagram.
   A demonstration shows the relationship live (the apparatus moving, and often a graph
       beside it
   tracking the same quantity). A check can reuse an earlier picture as its evidence. The
       recap puts
   the whole thing back together.
2. **Callbacks.** When an image recurs -- the hook's objects returning as the check's
    evidence --
   draw it the same way both times, so the learner recognises it.
3. **True, and to scale.** Compute geometry from the actual formulas in `task.txt` and the
   controls. Motion comes from the physics (closed-form in the clock), not from a looping
       tween.
4. **Motion where motion is the idea.** Animate with `requestAnimationFrame` and read the
    time from
   its timestamp or `performance.now()`. The picture must be a FUNCTION of (controls, time):
```

```
      never
   accumulate state frame by frame, and reset everything a scene owns when `update_scene` is
      called.
5. **Legible at a glance.** The stage is 960x720 (4:3); use all of it, with margins. Labels
   >= 20px, math-like symbols in an italic serif, one consistent palette (a force is the
      same colour
   in every scene). No text overlaps text or runs off the stage. Arrow heads sized to the
      arrow.
6. **Pictures, not paragraphs.** Captions and prose are the engine's. Put only labels,
    symbols, short
   equations and numeric read-outs on the stage -- never a sentence the narration will
      already say.
7. **Revealable.** Split each scene's diagram into LAYERS that start OFF where the lesson
    should
   build them up (weight, then tension, then the components) -- and declare PARTS for every
      thing the
   narration will name, so a cue can light it while the voice says it.
8. **Touchable.** The demonstration's quantity is a slider control. If the learner could
    naturally
   drag something on the picture (the bob, a point), let them -- and have the drag call the
      GLOBAL
   `window.update_<key>(value)` so the lesson records it exactly as if the slider moved.

## The contract (the oracle enforces all of it)
- `<body data-ui-layout="right">` with the drawing in `#stage` and every control inside
  `<div id="ui">` between `<!-- @FIGURE_UI_BEGIN -->` and `<!-- @FIGURE_UI_END -->`. The
      lesson
  hides `#ui` and gives `#stage` the whole frame; nothing outside `#ui` may be a control
      panel.
- Each control is `<label>Words <input ...></label>`:
  - a slider: `<input type="range" id="<key>Input" min max step value
      oninput="update_<key>(+this.value)">`
  - a layer: `<input type="checkbox" id="<key>Input" onchange="toggle_<key>(this.checked)">`
  - a text field, when the request has the learner TYPE something (a formula r = f(theta),
      an expression):
    `<input type="text" id="<key>Input" value="2*(1+cos(theta))"
       data-examples="4*cos(theta)|3*(1+cos(theta))" onchange="update_<key>(this.value)">`.
    Do not substitute a slider for a formula the request asks the learner to edit.
       `update_<key>`
    may receive a number when the text looks like one, so `String(v)` it. Compile the
       formula once per
    edit with only the names it may use in scope (theta, pi, sin, cos, sqrt...). Bad input
       -- half a formula,
    nonsense, empty, a division by zero -- must never throw: keep drawing the last good
       curve and say
    so briefly on the stage. `data-examples` lists 2-3 values that draw visibly different
       pictures;
    the learner gets them as one-click suggestions and the oracle checks each one.
- A slider with key `scene`, `min="0" max="4" step="1"`, handled by `update_scene(i)`.
- `window.LS_FIGURE = { scenes: [...], layers: [...], parts: [...] }`, each entry
  `{ key, label?, shows }` where `shows` says in plain words what it is. `scenes` is in
      slider
  order; `layers` keys equal their checkbox keys, and every layer also names its scene:
  `{ key: 'showStrip', scene: 'sweep', shows: '...' }`. Every layer you draw must be one the
      lesson
  will want -- the lesson is REFUSED if a layer is never shown, so do not draw spares.
- `update_emphasize(list)`: a space-separated list of part keys (or ''). The named parts
    stay at full
  strength and everything else on stage fades to about 0.16 opacity. Nothing is removed.
- Every handler is a top-level global function. The page must work from `file://` with NO
    network:
  no CDN, no fonts, no fetches -- hand-written SVG (preferred) or canvas for fills, but all
      TEXT as SVG
  `<text>` or HTML so it can be measured.

## The loop -- do not skip it
Run the oracle after every meaningful change:

    npx tsx /Users/shaden/Documents/lessonStudio/author/page_oracle.ts figure.html --shots
       shots --scenes 5

It freezes the clock, poses every scene with all layers on and all off, checks every control
    and
layer and part, and saves screenshots to `shots/`. Then **Read every screenshot** and judge
    it
against the bar above as a demanding designer would: is each scene a distinct idea? Is
    anything
cramped, overlapping, tiny, empty, off-balance, or wrong? Would a student understand it
    without the
narration? Fix, re-run, look again. You are done only when the oracle prints no problems AND
    you have
```

```
looked at every final screenshot and would be proud to ship it.

Work autonomously; do not ask questions. Write nothing outside this directory. Finish with a
    short
summary of each scene's picture, its layers, and its parts.
```

Plan summary (`plan.md`).

```
# The lesson this page draws for: The Simple Pendulum: Forces and Period

Objective: Identify the forces acting on a simple pendulum and describe how the restoring
    force depends on the displacement angle. Determine the period of a simple pendulum and
    explain how it is dictated by the length of the string and the acceleration due to
    gravity.

## The request's scenes, in order

1. Have you ever wondered why grandfather clocks keep such precise time regardless of how
    heavy their swinging weights are? The secret lies in the physics of a simple pendulum,
    where the time it takes to complete one swing depends only on its length and gravity.
2. Draw a fixed pivot point at the top of the screen and suspend a string of length L from
    it. Attach a spherical pendulum bob of mass m to the bottom of the string, and draw a
    dashed vertical line down from the pivot to represent the equilibrium position. Displace
    the bob to the right by an angle theta, drawing the arc length s, the downward weight
    vector mg, the tension vector along the string, and the restoring force vector -mg
    sin(theta) pointing back toward equilibrium.
3. Drag the pendulum bob to change the initial angle and watch the restoring force vector
    dynamically scale in magnitude. As the angle theta increases from 0 to 15 degrees, the
    component of gravity acting along the arc, -mg sin(theta), grows larger, providing a
    stronger restoring force toward the equilibrium position.
4. Adjust a slider to modify the string length and observe how the period of the pendulum's
    oscillation changes. Increasing the string length L from 0.5 meters to 2.0 meters causes
    the period T to increase, as the period is equal to 2 pi times the square root of L over
    g.
5. The motion of a simple pendulum is driven by a restoring force from gravity that scales
    with the sine of the displacement angle. For small angles, this results in simple
    harmonic motion where the period is completely determined by the length of the pendulum
    and the local acceleration due to gravity.

## The planned beats, by act (a beat is one step of the lesson; the narration is written
    later)

### Why Pendulums Keep Time
The learner can state that a pendulum's swing time depends only on length and gravity,
    motivating the rest of the lesson.

- `intro-question` (explain) -- Open on an ambient idle pendulum (grandfather-clock style)
    swinging; narrate the hook verbatim (unit 4): why does a heavy pendulum keep the same
    precise time regardless of weight? Advance on narration end so it plays like film.
- `intro-promise` (explain) -- Bridge line: promise that we'll uncover the forces on the bob
    and see exactly what sets the period (length and g). Narrated, advance on narration end;
    no new visual needed, keep prior frame.

### Anatomy of a Simple Pendulum
The learner can name and locate every force and geometric quantity on a displaced simple
    pendulum: pivot, string length L, bob mass m, angle theta, arc s, weight mg, tension,
    and restoring force -mg sin(theta).

- `build-pivot-string` (scene) -- Build the figure additively (unit 6, part 1): draw a fixed
    pivot at top, a string of length L hanging down, a spherical bob of mass m at its end,
    and a dashed vertical equilibrium line from the pivot. Narrated, advance on narration
    end.
- `build-forces` (scene) -- Continuing the SAME persistent figure (unit 6, part 2): displace
    the bob to the right by angle theta, add the arc length s, the downward weight vector
    mg, the tension vector along the string, and the restoring force vector -mg sin(theta)
    pointing back toward equilibrium. Narrated, advance on narration end.
- `forces-check` (mcq) -- Checkpoint on the still-visible force diagram: ask the learner to
    identify which force component actually restores the bob toward equilibrium (the
    tangential component of gravity, -mg sin theta) versus distractors (full weight,
    tension, a fictitious force). Keep the labeled figure on screen.
- `forces-remediate` (explain) -- Re-teach from a different angle: pose the SAME figure
    decomposed into the weight vector split into a component along the string (balanced by
    tension) and a component along the arc (the restoring force) -- make the tangential
    component visually obvious, then loop back to re-ask.

### Angle and the Restoring Force
The learner can explain, and has observed directly, that the restoring force -mg sin(theta)
    grows as the displacement angle theta grows.

- `angle-explorable` (explorable) -- THE demo the lesson turns on -- GATE it with a goal: a
```

```
    theta slider (0 to 15 degrees) the learner drags to displace the bob; the
    restoring-force vector on the persistent figure must visibly rescale live with
    sin(theta). Require the learner to sweep past roughly the top of the range before
    Continue unlocks. Also set ask:true here to invite questions while they're exploring.
    Narrated task instruction.
- `angle-check` (mcq) -- Checkpoint right after the drag demo, while the figure is still on
    screen: ask what happens to the restoring force as theta increases from 0 to 15 degrees
    (it grows, since it scales with sin theta), with a distractor claiming it stays constant
    or shrinks. This is the lesson's key mastery gate.
- `angle-remediate` (explain) -- Re-teach by posing the figure at a concrete side-by-side
    comparison: theta=5 degrees vs theta=15 degrees, with the restoring-force vector's
    length annotated numerically at each, making the growth undeniable before looping back
    to re-ask.

### Length and the Period
The learner can state and has observed that T = 2*pi*sqrt(L/g), so period grows with string
    length and shrinks with stronger gravity.

- `length-explorable` (explorable) -- A second, UNGATED demo (plain Continue button, no goal
    -- this one is for free exploration, set ask:true so learners can ask questions here):
    an L slider from 0.5 m to 2.0 m that actually re-swings the pendulum at the new length
    so the learner sees and can time the period changing, alongside the formula T =
    2*pi*sqrt(L/g). Narrated task.
- `length-check` (mcq) -- Checkpoint: ask which two quantities the period actually depends
    on (length L and gravity g -- NOT the mass or the amplitude), with distractors naming
    mass or the swing angle.
- `length-remediate` (explain) -- Re-teach by posing the figure at two lengths (0.5 m and
    2.0 m) side by side with their periods labeled numerically from the formula, making the
    L-dependence concrete, then loop back to re-ask. Also make clear mass never entered the
    formula.

### Putting It Together
The learner can summarize the full picture: restoring force scales with sin(theta), and for
    small angles this yields SHM whose period is fixed purely by L and g.

- `recap-explain` (explain) -- Narrated recap over the final labeled figure (unit 12): the
    restoring force from gravity scales with sin(theta); for small angles this gives simple
    harmonic motion whose period is set entirely by L and g. Advance on narration end.
- `final-check` (mcq) -- Final consolidation checkpoint: ask which quantity does NOT affect
    the period of a simple pendulum (correct: the mass / the bob's weight; distractors:
    length, gravity), reinforcing the m-independence takeaway.
- `final-remediate` (explain) -- Re-teach mass-independence by pointing at the pivot-to-bob
    figure and noting mass canceled out of T = 2*pi*sqrt(L/g) entirely -- show the formula
    has no m in it, then loop back to re-ask.
- `mastery-branch` (branch) -- Invisible fork on whether the learner truly mastered the
    restoring-force relationship earlier in the lesson: route accordingly.
- `harder-challenge` (explain) -- For learners who already showed mastery: pose a genuinely
    harder extension -- e.g. how the period would change on the Moon where g is smaller, or
    how energy trades between kinetic and potential across a swing -- then end the lesson.
- `recap-final` (explain) -- For everyone else: a gentle, plainly-stated final recap of the
    two big ideas (restoring force scales with sin theta; period depends only on L and g)
    with the finished figure on screen, then end the lesson.
```

C.11.3 FIGURE (WITHOUT THE PAGE AGENT)

System prompt.

```
You are building the APPARATUS for an interactive lesson: one figure, drawn as data, that
    every
beat of the lesson will pose. It is not an illustration. It is the thing the lesson is
    about, and
the learner will drag its controls.

WHAT YOU PRODUCE is a figure kit: a stage, a list of scene nodes, the controls it reacts to,
    the
LAYERS it can show one at a time, the pose it starts in, and a line about each part.
    Wherever a
number depends on a control, you write a BINDING instead of the number. That is what makes
    it
react: the learner's slider value and your `{"$ref":"theta"}` meet in the same object, and
    the
figure redraws.

## LAYERS -- THE PART MOST OF THIS JOB IS
  A lesson introduces one thing at a time: the apparatus, then the force on it, then the one
  component of that force that does the work, and only at the recap does the whole annotated
  diagram go up. So the figure is not one picture, it is a picture with switches.

  Group the nodes into 3-6 layers, one concept each, and give every layer a TOGGLE control.
```

```
      Then
  gate the layer's nodes on it, in their `opacity`:
      {"id":"weightArrow", "kind":"arrow", ..., "opacity":
          {"$if":[{"$eq":[{"$ref":"showWeight"},1]},1,0]}}
  Every node a layer lists must vanish when its toggle is 0 and be visible when it is 1.
      This is
  checked by switching each layer off and sampling the figure -- a layer wired to nothing is
  refused, because a figure that always draws everything cannot teach anything in order.

  The BASE layer (the apparatus itself -- the thing that is always there) can be ungated, or
      give
  it a toggle too if the lesson might want an empty stage. Everything annotative -- vectors,
  angle marks, component decompositions, axes, derived labels -- belongs to a layer.

## IF YOU WERE GIVEN A REFERENCE BUILD, IT IS NOT A DESIGN
  A reference figure is an INSPECTOR: it shows every vector, every label and every
      decomposition
  at once, because somebody exploring the physics wants all of it in front of them. That is
      the
  opposite of a lesson, and porting it faithfully would produce a control panel with
      narration
  read over it. Take from it what somebody actually decided -- the geometry, the control
      ranges,
  which quantities matter, what the demo steps are -- and then CUT IT INTO LAYERS, drop
      anything
  the scene plan never asks for, and re-draw it in this vocabulary and this palette.

## DRAWING (a `scene` beat's storyboard -- pure JSON, no code)
  storyboard  {duration: <ms>, initial: SceneNode[], tweens: Tween[], stage?: {w,h},
      camera?: CameraKey[]}
  tween       {target: <node id>, property: <animProp>, from?, to, start: <ms>, duration:
      <ms>, easing?}
  camera      {at: <ms>, x, y, w, h, easing?} -- keyframed viewBox window; omit for the
      whole stage
  every node  id, x, y, opacity, scale, rotation, fill, gradient, glow
  node kinds
    rect    w, h                                  axis-aligned rectangle anchored at (x,y)
    circle  r, stroke?, strokeWidth?              circle centred on (x,y)
    line    x2, y2, stroke?                       segment from (x,y) to (x2,y2)
    arrow   x2, y2, stroke?                       line with a head at (x2,y2)
    label   text, size?, anchor?, baseline?, weight? plain text; `text` is a RichText tree
        or a string. NOT math -- TeX is not rendered on the stage
    path    d, stroke?, strokeWidth?, draw?, len?  SVG path; animate `draw` 0->1 with `len`
        set to stroke it on
    ring    rOuter, rInner, innerDy?, stroke?     annulus -- an aperture, an orbit, a donut
    group   children                              nests nodes; animating the group moves
        them together
  animate     x y opacity scale rotation fill draw
  easings     smooth smootherstep rushInto rushFrom slowInto thereAndBack linear easeIn
      easeOut easeInOut cubicInOut expoOut backOut elasticOut bounceOut
  coordinates are stage units, growing right and DOWN; set stage:{w,h} and draw inside it.

## BINDINGS (an `explorable`'s figure, reacting to its own controls)
  Give the `explorable` a viz of {name:"declarative", props:{storyboard: ...}} and write a
  BINDING wherever a number goes. Its control values are the storyboard's inputs, so the
  figure re-draws as the learner drags -- a new interactive demo out of pure JSON.
    {"$ref":"hole"}                    the live value of the control keyed `hole` (a toggle
        reads 1/0)
    {"$mul":[{"$ref":"hole"},0.9]}     arithmetic: $add $sub $mul $div $min $max $clamp
        $round $sin $cos $tan $sqrt $abs $pow $mod
    {"$if":[{"$gt":[{"$ref":"hole"},20]},"#ef4444","#38bdf8"]}   pick between two values
  $sub and $div fold left; $clamp is [value,lo,hi]; $round is [value] or [value,step];
  $sin $cos $tan $sqrt $abs take [value] and work in RADIANS; $pow is [base,exp] and $mod is
  [value,modulus]; $if compares with $lt $lte $gt $gte $eq. That is the whole language --
      there is
  nothing else, and a binding to a control you did not declare is refused with the rest of
      the turn.

## COLOUR (name a colour; never invent a hex)
  blue teal green yellow gold red purple pink white lightGray gray darkGray indigo violet
      lavender emerald amber onMark
  Write the NAME's hex -- `"fill": "#58c4dd"` for blue -- and the theme re-maps it on paper
      stock
  and in light mode. Any other hex is kept exactly as written in every theme, which is the
      escape
  hatch and almost never what you want. The hexes are:
  blue=#58c4dd teal=#5cd0b3 green=#83c167 yellow=#ffff00 gold=#f0ac5f red=#fc6255
      purple=#9a72ac pink=#d147bd white=#ffffff lightGray=#bbbbbb gray=#888888
      darkGray=#444444 indigo=#818cf8 violet=#6366f1 lavender=#c4b5fd emerald=#34d399
      amber=#f59e0b onMark=#0b0e1a
```

```
## RULES THAT COST A WHOLE RUN WHEN BROKEN
  * DRAW IT COMPLETE. The nodes you return are what an `explorable` shows at t=0. A node
      with
    `opacity: 0` waiting for a tween is invisible there, forever. Hide things with a BINDING
    instead -- `{"$if":[{"$eq":[{"$ref":"showForces"},1]},1,0]}` on `opacity` -- and give
        the
    lesson a toggle for it. Then the filmed beats can animate that same opacity and the
    interactive ones can switch it.
  * STAY ON THE STAGE AT EVERY EXTREME. Each slider is sampled at BOTH ends and the figure
      is
    checked for nodes that have left the stage and for numbers that came out NaN. A vector
        whose
    length is a force will be longest at the extreme -- scale it so it still fits there, and
        clamp
    it with `$clamp` if the physics would run it off the edge.
  * ANGLES ARE RADIANS. `$sin`/`$cos` take radians; a slider in degrees needs
    `{"$mul":[{"$ref":"theta"},0.017453292519943295]}`. Write the multiplication out; there
        is
    no pi constant.
  * y GROWS DOWNWARD, as in SVG. A pendulum hanging below its pivot is at a LARGER y.
  * THERE IS NO ARC PRIMITIVE. A swept arc that has to follow a control is 4-8 short `line`
    segments whose endpoints are bound. A `path` may only be used for something that never
        moves,
    because its `d` is a string and a binding cannot live inside one.
  * LABELS ARE PLAIN TEXT. No TeX on the stage. '-mg sin theta' is fine; '$-mg\\sin\\theta$'
      is not.
  * KEEP IT UNDER ABOUT 25 NODES. Every one of them has to stay legible at 720p.
  * NO TWO LABELS IN THE SAME PLACE, at any pose the learner can reach. You are placing text
      by
    coordinate with no font metrics in front of you, so leave room -- a label is roughly
    0.55 x its size wide per character. Overlapping pairs are reported.
  * AN ARROW WHOSE LENGTH IS A QUANTITY NEEDS A FLOOR. The head is drawn for a full-length
    arrow, so a vector that shrinks to zero renders as a blob at its own tail. Either clamp
        the
    length (`$max` against about 18) or bind its `opacity` to 0 where the quantity vanishes.
    Checked across the WHOLE slider, not at its ends -- the worst case is usually in the
        middle.
  * A LABEL GOES BESIDE ITS MARK, NEVER ON IT. `m` printed over the bob labels the right
      thing
    and cannot be read: the mark is solid colour and the glyph is 15px.
  * FILL THE STAGE YOU DECLARE. Use most of it -- a drawing that occupies a third of its own
    canvas is reported, and reads as a thumbnail dropped into a white box however good it
        is.

## HOUSE STYLE
  This figure sits beside prose in a dark studio theme (and the same lesson is read on
      paper),
  so it is drawn the way the engine's own examples are drawn:
  * Structure recedes and data speaks: hairlines and construction lines in
      `gray`/`darkGray`,
    the quantities the lesson is about in a named colour each.
  * ONE COLOUR PER QUANTITY, used everywhere that quantity appears. Declare them in
      `symbols`
    with a dark and a light hue; the prose tags the same symbol and the two match, so a
        learner
    reading $\theta$ in a sentence knows which angle on the stage it is. Keep each hue at
        4.5:1
    against its own ground -- this is checked.
  * No drop shadows, no white halos behind text, no gradients except where a gradient IS the
    point. No decorative chrome: the figure is the argument, not a poster of it.
  * Label sizes in a narrow band (13-20), `anchor` set deliberately, and nothing overlapping
    anything at any pose the learner can reach.

Name every node with a short, meaningful id (`bob`, `stringLine`, `weightArrow`): act
    writers
animate the figure by naming these ids, and a node called `node7` is a node nobody will use.
```

User prompt.

```
## THE REQUEST
Create a narrated educational video that teaches the textbook figure
    'CNX_UPhysics_15_05_SimplePend' (university physics volume 1). Target length: 60-120
    seconds of narrated animation.
  Pacing: the request asks for 60-120 seconds of narrated animation.

## LEARNING OBJECTIVE
Identify the forces acting on a simple pendulum and describe how the restoring force depends
    on the displacement angle. Determine the period of a simple pendulum and explain how it
    is dictated by the length of the string and the acceleration due to gravity.
```

```
## SCENE PLAN (the spine; transcribe it, do not replace it)
  1. Hook -- Have you ever wondered why grandfather clocks keep such precise time regardless
       of how heavy their swinging weights are? The secret lies in the physics of a simple
       pendulum, where the time it takes to complete one swing depends only on its length and
       gravity.
  2. Construction -- Draw a fixed pivot point at the top of the screen and suspend a string
       of length L from it. Attach a spherical pendulum bob of mass m to the bottom of the
       string, and draw a dashed vertical line down from the pivot to represent the
       equilibrium position. Displace the bob to the right by an angle theta, drawing the arc
       length s, the downward weight vector mg, the tension vector along the string, and the
       restoring force vector -mg sin(theta) pointing back toward equilibrium.
  3. Demonstration -- Drag the pendulum bob to change the initial angle and watch the
       restoring force vector dynamically scale in magnitude. As the angle theta increases
       from 0 to 15 degrees, the component of gravity acting along the arc, -mg sin(theta),
       grows larger, providing a stronger restoring force toward the equilibrium position.
  4. Demonstration -- Adjust a slider to modify the string length and observe how the period
       of the pendulum's oscillation changes. Increasing the string length L from 0.5 meters
       to 2.0 meters causes the period T to increase, as the period is equal to 2 pi times
       the square root of L over g.
  5. Consolidation -- The motion of a simple pendulum is driven by a restoring force from
       gravity that scales with the sine of the displacement angle. For small angles, this
       results in simple harmonic motion where the period is completely determined by the
       length of the pendulum and the local acceleration due to gravity.

## CONSTRAINTS (rules on HOW, not content to teach)
  - Render in 2D unless the figure is inherently 3D. Keep every axis, label and symbol
       legible at 720p; no gratuitous camera motion.
  - Name the physical quantities and their units; keep the geometry quantitatively honest.
  - Each demonstration must actually play out on screen: sweep the parameter through its
       range and show the figure responding.
  - Narration must be synchronised to the visual it describes.

## WHAT THE LESSON WILL ASK THIS FIGURE TO SHOW
These are the beats that will pose it, with their assignments. Every one of them has to be
possible with the layers and controls you return -- and nothing else has to be.
  Why Pendulums Keep Time -- The learner can state that a pendulum's swing time depends only
       on length and gravity, motivating the rest of the lesson.
    intro-question (explain): Open on an ambient idle pendulum (grandfather-clock style)
         swinging; narrate the hook verbatim (unit 4): why does a heavy pendulum keep the
         same precise time regardless of weight? Advance on narration end so it plays like
         film.
  Anatomy of a Simple Pendulum -- The learner can name and locate every force and geometric
       quantity on a displaced simple pendulum: pivot, string length L, bob mass m, angle
       theta, arc s, weight mg, tension, and restoring force -mg sin(theta).
    build-pivot-string (scene): Build the figure additively (unit 6, part 1): draw a fixed
         pivot at top, a string of length L hanging down, a spherical bob of mass m at its
         end, and a dashed vertical equilibrium line from the pivot. Narrated, advance on
         narration end.
    build-forces (scene): Continuing the SAME persistent figure (unit 6, part 2): displace
         the bob to the right by angle theta, add the arc length s, the downward weight
         vector mg, the tension vector along the string, and the restoring force vector -mg
         sin(theta) pointing back toward equilibrium. Narrated, advance on narration end.
    forces-check (mcq): Checkpoint on the still-visible force diagram: ask the learner to
         identify which force component actually restores the bob toward equilibrium (the
         tangential component of gravity, -mg sin theta) versus distractors (full weight,
         tension, a fictitious force). Keep the labeled figure on screen.
    forces-remediate (explain): Re-teach from a different angle: pose the SAME figure
         decomposed into the weight vector split into a component along the string (balanced
         by tension) and a component along the arc (the restoring force) -- make the
         tangential component visually obvious, then loop back to re-ask.
  Angle and the Restoring Force -- The learner can explain, and has observed directly, that
       the restoring force -mg sin(theta) grows as the displacement angle theta grows.
    angle-explorable (explorable): THE demo the lesson turns on -- GATE it with a goal: a
         theta slider (0 to 15 degrees) the learner drags to displace the bob; the
         restoring-force vector on the persistent figure must visibly rescale live with
         sin(theta). Require the learner to sweep past roughly the top of the range before
         Continue unlocks. Also set ask:true here to invite questions while they're
         exploring. Narrated task instruction.
    angle-check (mcq): Checkpoint right after the drag demo, while the figure is still on
         screen: ask what happens to the restoring force as theta increases from 0 to 15
         degrees (it grows, since it scales with sin theta), with a distractor claiming it
         stays constant or shrinks. This is the lesson's key mastery gate.
    angle-remediate (explain): Re-teach by posing the figure at a concrete side-by-side
         comparison: theta=5 degrees vs theta=15 degrees, with the restoring-force vector's
         length annotated numerically at each, making the growth undeniable before looping
         back to re-ask.
  Length and the Period -- The learner can state and has observed that T = 2*pi*sqrt(L/g),
       so period grows with string length and shrinks with stronger gravity.
    length-explorable (explorable): A second, UNGATED demo (plain Continue button, no goal
         -- this one is for free exploration, set ask:true so learners can ask questions
         here): an L slider from 0.5 m to 2.0 m that actually re-swings the pendulum at the
```

```
        new length so the learner sees and can time the period changing, alongside the
        formula T = 2*pi*sqrt(L/g). Narrated task.
    length-check (mcq): Checkpoint: ask which two quantities the period actually depends on
        (length L and gravity g -- NOT the mass or the amplitude), with distractors naming
        mass or the swing angle.
    length-remediate (explain): Re-teach by posing the figure at two lengths (0.5 m and 2.0
        m) side by side with their periods labeled numerically from the formula, making the
        L-dependence concrete, then loop back to re-ask. Also make clear mass never entered
        the formula.
  Putting It Together -- The learner can summarize the full picture: restoring force scales
      with sin(theta), and for small angles this yields SHM whose period is fixed purely by
      L and g.
    recap-explain (explain): Narrated recap over the final labeled figure (unit 12): the
        restoring force from gravity scales with sin(theta); for small angles this gives
        simple harmonic motion whose period is set entirely by L and g. Advance on narration
        end.
    final-check (mcq): Final consolidation checkpoint: ask which quantity does NOT affect
        the period of a simple pendulum (correct: the mass / the bob's weight; distractors:
        length, gravity), reinforcing the m-independence takeaway.
    final-remediate (explain): Re-teach mass-independence by pointing at the pivot-to-bob
        figure and noting mass canceled out of T = 2*pi*sqrt(L/g) entirely -- show the
        formula has no m in it, then loop back to re-ask.
    harder-challenge (explain): For learners who already showed mastery: pose a genuinely
        harder extension -- e.g. how the period would change on the Moon where g is smaller,
        or how energy trades between kinetic and potential across a swing -- then end the
        lesson.
    recap-final (explain): For everyone else: a gentle, plainly-stated final recap of the
        two big ideas (restoring force scales with sin theta; period depends only on L and
        g) with the finished figure on screen, then end the lesson.

Build the figure. Keep the reference build's control keys, ranges and starting values where
    it
has them -- those ranges are decisions somebody already made about the physics. Cut
    everything
the beats above never need, and cut the diagram into layers so they can be shown in order.
```

Output schema.

```
{
  "type": "object",
  "properties": {
    "stage": {
      "type": "object",
      "properties": {
        "w": {
          "type": "number"
        },
        "h": {
          "type": "number"
        }
      },
      "required": [
        "w",
        "h"
      ]
    },
    "initial": {
      "type": "array",
      "description": "the scene nodes, drawn complete; any number may be a binding object",
      "items": {
        "type": "object"
      }
    },
    "controls": {
      "type": "array",
      "description": "every control the figure reacts to, layer toggles included",
      "items": {
        "type": "object",
        "properties": {
          "key": {
            "type": "string"
          },
          "label": {
            "type": "string"
          },
          "kind": {
            "type": "string",
            "description": "slider | toggle"
          },
          "min": {
```

```
            "type": "number"
          },
          "max": {
            "type": "number"
          },
          "step": {
            "type": "number"
          },
          "unit": {
            "type": "string"
          }
        },
        "required": [
          "key",
          "label",
          "kind"
        ]
      }
    },
    "defaults": {
      "type": "object",
      "description": "every control key -> its starting value (a toggle is 1 or 0)"
    },
    "layers": {
      "type": "array",
      "description": "3-6 layers, one concept each, every one switched by its own toggle
          control",
      "items": {
        "type": "object",
        "properties": {
          "key": {
            "type": "string",
            "description": "the toggle control that switches it"
          },
          "label": {
            "type": "string"
          },
          "shows": {
            "type": "string",
            "description": "the one concept this layer is about"
          },
          "nodes": {
            "type": "array",
            "items": {
              "type": "string"
            },
            "description": "node ids this layer governs -- each gated on `key` in its
                opacity"
          }
        },
        "required": [
          "key",
          "label",
          "shows",
          "nodes"
        ]
      }
    },
    "symbols": {
      "type": "object",
      "description": "symbol name -> {tex, means, dark, light}: the quantity colour key
          shared by figure and prose"
    },
    "notes": {
      "type": "array",
      "items": {
        "type": "string"
      },
      "description": "a line per part of the figure, for the writers"
    }
  },
  "required": [
    "stage",
    "initial",
    "controls",
    "defaults",
    "layers",
    "notes"
  ]
}
```

C.11.4 ACT WRITER

System prompt.

```
You are writing the CONTENT of one act of an interactive lesson. The lesson's shape is
    already
decided and is not yours to change: you are given beat ids, their types, and their
    assignments,
and you return the `params` for each one. Nothing else.

WHAT YOU PRODUCE is `{actId, params: {<beat id>: {...}}, notes: [...]}`.
  * You cannot set an edge. onWrong, onTimeout, then, else, exits, whenRef are the plan's
      and are merged over
    anything you send. If an assignment seems to need a different edge, say so in `notes`.
  * You cannot add, rename or reorder a beat. A beat you return nothing for becomes a
      visible
    placeholder in the lesson, so return every one you were given.
  * `notes` is for a human: a unit you could not honour, an assignment you think is wrong.

## BEAT TYPES (the `type` + `params` of an `addBeat` spec)
  explain
    Prose (markdown + $math$) with an optional visual. The plainest beat; `say` is sugar for
        one.
    * text             markdown + $math$; the words the learner reads
      narration        spoken variant, synthesized on demand
      viz              {name, props?, persistent?} -- a registered visual to pose
      advance          "narration" (move on when the spoken clip ends) or ms to dwell -- a
          run of beats carrying it plays like a film and stops at the first one that does
          not
      exits            [{label, to}] -- offer SEVERAL ways out instead of one Continue; `to:
          null` ends the lesson
      e.g. {"text":"The rays cross **at** the hole, so the image arrives inverted."}
  mcq
    A multiple-choice checkpoint. Marks the beat a checkpoint, so it scores and gates
        progress.
    * prompt           the question; markdown + $math$
    * choices          [{text, correct?, misconception?}] -- exactly one should be `correct:
        true`
      correctFeedback  prose shown after a right answer (default 'Correct!')
      wrongFeedback    prose shown after a wrong answer
      onWrong          beat id to detour to on a wrong answer
      skill            a correct answer raises mastery of this skill name
      narration        spoken variant, synthesized on demand
      e.g. {"prompt":"Doubling $u$ while holding $v$ fixed does what to the
          image?","choices":[{"text":"Halves it","correct":true},{"text":"Doubles
          it","misconception":"m-proportional-to-u"}],"correctFeedback":"Right -- $m=v/u$,
          so $u$ is in the denominator."}
  branch
    An invisible fork: routes to `then` when the named guard passes, `else` otherwise.
        `whenRef` must name a guard the LESSON already registered -- you cannot define one,
        since runtime-authored beats are pure JSON. Prefer `explain` with `exits` to offer
        the learner a choice; use `branch` only to reuse a guard the lesson exposes.
    * whenRef          name of an already-registered guard
    * then             beat id taken when the guard passes
    * else             beat id taken when it does not
      e.g. {"whenRef":"masteredMagnification","then":"hard-q","else":"recap"}
  scene
    A NEW declarative figure, drawn and animated from primitives. Pure JSON, so you can
        author one for a question the lesson never anticipated -- see the DRAWING vocabulary
        for node kinds, animatable properties and easings.
    * storyboard       {duration, initial: SceneNode[], tweens: Tween[], stage?: {w,h},
        camera?: CameraKey[]}
      narration        spoken script; plays alongside the motion, not synchronized to it
      e.g. {"storyboard":{"duration":1200,"stage":{"w":400,"h":200},"initial":[{"id":"ray","
          kind":"line","x":20,"y":40,"x2":380,"y2":160,"stroke":"#f59e0b","opacity":0},{"id"
          :"cap","kind":"label","x":20,"y":175,"text":"one ray, one direction","size":18}],"
          tweens":[{"target":"ray","property":"opacity","to":1,"start":0,"duration":600,"eas
          ing":"smooth"}]}}
  freeResponse
    A typed-answer checkpoint, graded against `accept` after normalization (case, spacing,
        articles, plurals; numbers by value; the phrase inside a short answer). Only for
        answers with a short canonical form -- use `mcq` otherwise.
    * prompt           the question; markdown + $math$
    * accept           [string] -- every spelling you will accept; matched after
        normalization
      hint             shown as the wrong-answer feedback when `wrongFeedback` is absent
      correctFeedback  prose shown after a right answer
      wrongFeedback    prose shown after a wrong answer
      onWrong          beat id to detour to on a wrong answer
      skill            a correct answer raises mastery of this skill name
      misconception    a wrong answer bumps this misconception's strength
```

```
      e.g. {"prompt":"With $u=10$ and $v=5$, what is the magnification
          $m$?","accept":["0.5","1/2",".5"],"hint":"$m = v/u$."}
  explorable
    An INTERACTIVE demo: controls the learner drags, wired to a visual. Two ways to name the
        visual -- an existing registered one (see the VISUALS list for names and accepted
        props), or `{name:"declarative"}` with a storyboard whose numbers are BINDINGS to
        the control keys, which is how you build a brand-new interactive figure without
        registering code.
    * controls         [{key, label, kind: slider|toggle|button|choice|text, min?, max?,
        step?, unit?, options?, placeholder?, suggestions?, rows?}] -- `text` is a typed
        field (a formula r = f(theta)); its value is the string, and `suggestions` [{label,
        value}] are one-click alternatives. For CODE give `rows` (e.g. 8): it becomes a
        multi-line editor
    * viz              {name, props?, persistent?} -- control values are merged over
        `props`, so a control named `u` sets prop `u`
      defaults         {key: number|boolean|string} -- starting values (else sliders start
          at min, toggles false, text empty)
      note             prose shown beside the demo
      goal             {key, equals?|min?|max?, tolerance?} -- hide Continue until the
          learner reaches it
      task             what to do; the instruction, shown until a `goal` (if any) is met
      success          shown once the `goal` is met, in place of `task`
      ask              true to show a free-text question box on this demo
      narration        spoken variant, synthesized on demand
      e.g. {"controls":[{"key":"hole","label":"Hole width","kind":"slider","min":1,"max":40,
          "step":1,"unit":"px"},{"key":"__next","label":"Continue","kind":"button"}],"defaul
          ts":{"hole":6},"viz":{"name":"declarative","props":{"storyboard":{"duration":0,"st
          age":{"w":400,"h":220},"initial":[{"id":"gap","kind":"rect","x":200,"y":{"$sub":[1
          10,{"$div":[{"$ref":"hole"},2}]}],"w":4,"h":{"$ref":"hole"},"fill":"#fbbf24"},{"id
          ":"blur","kind":"circle","x":330,"y":110,"r":{"$mul":[{"$ref":"hole"},0.9]},"fill"
          :"#38bdf8","opacity":0.5}],"tweens":[]}}},"task":"Widen the hole and watch the
          spot on the wall."}
  (* = required)

## THE FIGURE (an existing page, mounted as-is -- you POSE it, you never draw it)
  It draws itself and runs its own animation. You cannot draw on it and cannot animate it --
  what you can do is reveal part of it and pose it.

  SCENES -- it draws a different picture per scene. EVERY figure beat poses `scene` to one
      of
  these; a beat that does not is refused. Pick the picture that fits what the beat says:
    scene 0  clocks           two grandfather clocks side by side, a light bob and a heavy
        one on strings of the same length, swinging in step with the same T = 2.00 s
    scene 1  build            the textbook free-body diagram built up: pivot, string L, bob
        m, dashed equilibrium, angle theta, arc s, and the forces on the bob
    scene 2  angle            the bob draggable through theta = 0..15 deg while the
        restoring force -mg sin theta rescales, beside a plot of F against theta
    scene 3  length           the pendulum actually swinging at the length L, its angle
        traced against time, beside the curve T = 2pisqrt(L/g)
    scene 4  recap            the finished force diagram, the chain from -mg sin theta to T
        = 2pisqrt(L/g), and the two unequal bobs keeping the same period
  Layers and parts belong to the scene that draws them; showing one another scene lacks does
      nothing.

  LAYERS -- it shows one idea at a time. Name the ones a beat needs; every other goes off:
    in scene 1 (build):
      showTension        the tension arrow F_T along the string, pointing from the bob to
          the pivot
      showWeight         the weight arrow w = mg hanging straight down from the bob
      showComponents     the weight split into -mg cos theta along the string and the
          restoring -mg sin theta along the arc
    in scene 2 (angle):
      showForceCurve     the curve F = mg sin theta on the axes, with a marker at the angle
          now on stage
    in scene 3 (length):
      showPeriodCurve    the curve T = 2pisqrt(L/g) on the axes, with a marker at the length
          now on stage
    in scene 4 (recap):
      showTwins          the light and heavy bobs from the opening clocks, swinging in step
          to show mass drops out
  EVERY LAYER WAS DRAWN FOR THIS LESSON. A beat posing a scene should bring its layers in --
      a
  layer no beat ever shows is a picture the learner never sees, and it is refused.

  Controls (a beat may offer any subset as its own `controls`; the rest keep their pose):
    scene            0..4 step 1 -- "Scene"  (default 0)
    theta            0..15 step 0.5 -- "Angle theta (degrees)"  (default 12)
    length           0.5..2 step 0.05 -- "String length L (m)"  (default 1)
    gravity          1.6..9.8 step 0.1 -- "Gravity g (m/s^2)"  (default 9.8)
    showTension      toggle (1/0) -- "Tension"  (default 1)
    showWeight       toggle (1/0) -- "Weight mg"  (default 1)
```

```
    showComponents   toggle (1/0) -- "Components of mg"  (default 1)
    showForceCurve   toggle (1/0) -- "Force curve"  (default 1)
    showPeriodCurve  toggle (1/0) -- "Period curve"  (default 1)
    showTwins        toggle (1/0) -- "Two bobs, one period"  (default 1)

  PARTS -- names this page can LIGHT while the voice is on them. Everything else on screen
  fades back; nothing is removed. Use them so that when you say a thing, the learner sees
      it:
    pivot          the fixed pivot at the ceiling
    string         the string of length L
    bob            the pendulum bob of mass m
    equilibrium    the dashed vertical equilibrium line
    theta          the displacement angle theta
    arc            the arc length s from equilibrium to the bob
    tension        the tension arrow F_T
    weight         the weight arrow mg and its value in newtons
    restoring      the restoring force -mg sin theta and its value in newtons
    cosComp        the component -mg cos theta along the string
    forceCurve     the plot of F = mg sin theta against theta
    trace          the angle traced against time, with the period marked across one cycle
    periodCurve    the plot of T = 2pisqrt(L/g) against L
    period         the period read-out T = 2pisqrt(L/g)
    clockface      the clock dials and their stepping second hands
    twins          the light and heavy bobs staying in step

  PUT IT ON A BEAT with `figure`: {"show": ["showWeight"], "pose": {"thetaAngle": 25}}

  `figure.show` is what is up when the beat OPENS. If a beat needs three things, do not open
  with all three -- open with one and bring the others in as you say them, with `cues`:

    "narration": "Gravity pulls the bob straight down. Split that weight in two: the part
        along
                  the string is balanced by the tension, and the part along the arc is what
                  brings it back.",
    "figure": {"show": ["showWeight"], "pose": {"thetaAngle": 25}},
    "cues": [
      {"after": "straight down", "emphasize": ["weight"]},
      {"after": "Split that weight in two", "show": ["showComponents"]},
      {"after": "balanced by the tension", "show": ["showTension"], "emphasize": ["radial",
          "tension"]},
      {"after": "brings it back", "emphasize": ["restoring"]}
    ]

  HOW A CUE IS READ: `after` is a phrase copied EXACTLY from this beat's own `narration`.
      The
  clip comes back with a timestamp per word, so the cue fires the moment that phrase is
      spoken.
  A phrase that is not in the narration never fires and is refused. `show` brings a layer up
  and leaves it up -- so never `show` something `figure.show` already has. `emphasize`
      lights
  parts for as long as the phrase lasts and then lets go, so it is pointing, not revealing.
  NEVER write `viz` or `storyboard` on a beat. There is no drawing vocabulary in play here
      and
  a beat that draws its own picture is refused -- the whole lesson shows this one page.
  Dragging its own panel is reported to the engine as a control write, so the tutor still
  sees what the learner did -- but nothing verifies what it DRAWS. Do not describe details
  of the picture you have not been told about.

## VOICE
  * Write to one learner, in the second person, in plain sentences. No 'in this lesson we
      will'.
  * Maths is inline TeX between single dollars ($v/u$) or display between doubles. It is
      rendered
    by KaTeX in the prose column ONLY -- a figure `label` is plain text and TeX in one shows
        as
    backslashes on the stage.
  * `narration` is the SPOKEN line for the beat: what a voice says while this is on screen.
      One or
    two sentences, no markdown, no TeX, no symbols read as punctuation -- write 'v over u',
        not
    '$v/u$'. It is not a copy of `text`; it is what you would say out loud over the picture.
  * Name the physical quantities and their units where the request asks for them.

## THE THINGS THAT SILENTLY RUIN A BEAT
  * AN `explorable` NEEDS A WAY OUT. A `goal` does not create one -- it only un-hides a
      control
    with the exact key `__next`. Every explorable gets
        `{"key":"__next","label":"...","kind":"button"}`
    in its `controls`, or the learner reaches it and can never leave.
  * GATE AT MOST ONE DEMO, and only if your assignment says to. A `goal` HIDES that Continue
    until the learner reaches a value; on the beat the lesson turns on that is the point,
```

```
        and on
      every other demo it is a toll booth in front of something they were happy to explore. If
          your
      assignment does not ask for a gate, write the `task` as an invitation and leave `goal`
          out --
      the instruction still shows, and the learner leaves when they are ready.
    * A GOAL IS A DIRECTION, NOT A COORDINATE. Write `{key:"theta", min:15}` -- "get it up to
        15".
      `{equals: 15}` on a slider with ninety-one positions means hunting for one of them, and
          it
      passes every mechanical check because 15 is on the lattice. If you truly need one value,
          give
      it a `tolerance`. And check the number against the control you wrote: `min:15` against a
      slider that stops at 12 is a beat nobody can finish.
    * A CHECKPOINT'S `skill` IS A CONTRACT. Later beats fork on `stock.mastered:<skill>`. If
        your
      assignment names a skill, set exactly that string.
    * EVERY WRONG CHOICE CARRIES A `misconception`. That is what the engine records and what a
      remediation branch reads. A wrong answer with no misconception teaches the lesson
          nothing.
    * THE QUESTION MUST NOT ANSWER ITSELF. If the stem names the thing the right choice picks
        out
      ('which carrier crosses the PROTON exchange membrane?'), the learner reads the answer
          off the
      question. Describe it by what it does or where it is instead. Choice order does not
          matter:
      the engine shuffles the choices when it shows them.
    * `task` IS SHOWN UNTIL THE GOAL IS MET AND `success` REPLACES IT. Write both for a gated
        demo.
    * PUT `ask: true` on the explorable where a learner is most likely to be stuck. It is how
        they
      reach the tutor.

## WHAT HAPPENS TO WHAT YOU SEND
  It is compiled by the real engine and run through a battery that has no opinions and no
      mercy:
  every beat must reach an ending, every `goal` must be attainable on its own control's
      lattice,
  every binding must name a control that exists, every storyboard is sampled at five times
      and at
  every slider extreme for NaN and for nodes off the stage, every guard name must be real,
      and the
  whole thing must survive being replayed from its own event log. Anything it refuses comes
      back
  to you as a list, and you fix it and resend. Aim at passing the first time.
```

User prompt (Angle demo act).

```
## THE REQUEST
Create a narrated educational video that teaches the textbook figure
    'CNX_UPhysics_15_05_SimplePend' (university physics volume 1). Target length: 60-120
    seconds of narrated animation.
  Pacing: the request asks for 60-120 seconds of narrated animation.

## LEARNING OBJECTIVE
Identify the forces acting on a simple pendulum and describe how the restoring force depends
    on the displacement angle. Determine the period of a simple pendulum and explain how it
    is dictated by the length of the string and the acceleration due to gravity.

## SCENE PLAN (the spine; transcribe it, do not replace it)
  1. Hook -- Have you ever wondered why grandfather clocks keep such precise time regardless
      of how heavy their swinging weights are? The secret lies in the physics of a simple
      pendulum, where the time it takes to complete one swing depends only on its length and
      gravity.
  2. Construction -- Draw a fixed pivot point at the top of the screen and suspend a string
      of length L from it. Attach a spherical pendulum bob of mass m to the bottom of the
      string, and draw a dashed vertical line down from the pivot to represent the
      equilibrium position. Displace the bob to the right by an angle theta, drawing the arc
      length s, the downward weight vector mg, the tension vector along the string, and the
      restoring force vector -mg sin(theta) pointing back toward equilibrium.
  3. Demonstration -- Drag the pendulum bob to change the initial angle and watch the
      restoring force vector dynamically scale in magnitude. As the angle theta increases
      from 0 to 15 degrees, the component of gravity acting along the arc, -mg sin(theta),
      grows larger, providing a stronger restoring force toward the equilibrium position.
  4. Demonstration -- Adjust a slider to modify the string length and observe how the period
      of the pendulum's oscillation changes. Increasing the string length L from 0.5 meters
      to 2.0 meters causes the period T to increase, as the period is equal to 2 pi times
      the square root of L over g.
  5. Consolidation -- The motion of a simple pendulum is driven by a restoring force from
      gravity that scales with the sine of the displacement angle. For small angles, this
```

```
      results in simple harmonic motion where the period is completely determined by the
      length of the pendulum and the local acceleration due to gravity.

## CONSTRAINTS (rules on HOW, not content to teach)
  - Render in 2D unless the figure is inherently 3D. Keep every axis, label and symbol
      legible at 720p; no gratuitous camera motion.
  - Name the physical quantities and their units; keep the geometry quantitatively honest.
  - Each demonstration must actually play out on screen: sweep the parameter through its
      range and show the figure responding.
  - Narration must be synchronised to the visual it describes.

## YOUR ACT
act "angle-demo" -- Angle and the Restoring Force
  objective: The learner can explain, and has observed directly, that the restoring force
      -mg sin(theta) grows as the displacement angle theta grows.
  beats:
    angle-explorable  [explorable]  figure: declarative
      THE demo the lesson turns on -- GATE it with a goal: a theta slider (0 to 15 degrees)
          the learner drags to displace the bob; the restoring-force vector on the
          persistent figure must visibly rescale live with sin(theta). Require the learner
          to sweep past roughly the top of the range before Continue unlocks. Also set
          ask:true here to invite questions while they're exploring. Narrated task
          instruction.
    angle-check  [mcq]  figure: declarative
      Checkpoint right after the drag demo, while the figure is still on screen: ask what
          happens to the restoring force as theta increases from 0 to 15 degrees (it grows,
          since it scales with sin theta), with a distractor claiming it stays constant or
          shrinks. This is the lesson's key mastery gate.
      (grants the skill "restoring-force-mastery" -- the plan sets it; a later branch forks
          on it, so it is not yours to change)
      (continues to length-explorable; a wrong answer detours to angle-remediate -- the plan
          owns these; do not set them)
    angle-remediate  [explain]  figure: declarative
      Re-teach by posing the figure at a concrete side-by-side comparison: theta=5 degrees
          vs theta=15 degrees, with the restoring-force vector's length annotated
          numerically at each, making the growth undeniable before looping back to re-ask.
      (continues to angle-check -- the plan owns these; do not set them)

## THE DRAFT UNITS THIS ACT IS ACCOUNTABLE FOR (rewrite these; do not summarise them)
[unit 7]
## Scene 3 -- Demonstration

### Scene 3 -- Demonstration
[unit 8]
Drag the pendulum bob to change the initial angle and watch the restoring force vector
    dynamically scale in magnitude. As the angle theta increases from 0 to 15 degrees, the
    component of gravity acting along the arc, -mg sin(theta), grows larger, providing a
    stronger restoring force toward the equilibrium position.

## SOURCE PAGE (reference -- quote it, do not paste it)
[OpenStax, university-physics-volume-1, section 15-4-pendulums. Access for free at
https://openstax.org/books/university-physics-volume-1 - CC BY 4.0]
15.4 Pendulums - University Physics Volume 1 | OpenStax Skip to Content Go to
accessibility page Keyboard shortcuts menu University Physics Volume 1 15.4 Pendulums
University Physics Volume 1 15.4 Pendulums Contents Highlights Print Search Search Close
Search 15.4
Pendulums
Learning Objectives By the end of this section, you will be able to:
State the forces that act on a simple pendulum
Determine the angular frequency, frequency, and period of a simple pendulum in terms of
the length of the pendulum and the acceleration due to gravity
Define the period for a physical pendulum
Define the period for a torsional pendulum
Pendulums are in common usage. Grandfather clocks use a pendulum to keep time and a
pendulum can be used to measure the acceleration due to gravity. For small
displacements, a pendulum is a simple harmonic oscillator.
The Simple Pendulum
A simple pendulum is defined to have a point mass, also known as the pendulum bob ,
which is suspended from a string of length L with negligible mass ( Figure 15.20 ).
Here, the only forces acting on the bob are the force of gravity (i.e., the weight of
the bob) and tension from the string. The mass of the string is assumed to be negligible
as compared to the mass of the bob.
Figure
15.20
A simple pendulum has a small-diameter bob and a string that has a very small mass but
is strong enough not to stretch appreciably. The linear displacement from equilibrium is
s , the length of the arc. Also shown are the forces on the bob, which result in a net
force of - m g sin theta - m g sin theta toward the equilibrium position--that is, a
    restoring
force.
Consider the torque on the pendulum. The force providing the restoring torque is the
```

component of the weight of the pendulum bob that acts along the arc length. The torque is the length of the string L times the component of the net force that is perpendicular to the radius of the arc. The minus sign indicates the torque acts in the opposite direction of the angular displacement:
tau = - L (m g sin theta) ; I alpha = - L (m g sin theta) ; I d 2 theta d t 2 = - L (m g sin theta) ; m
L 2 d 2 theta d t 2 = - L (m g sin theta) ; d 2 theta d t 2 = - g L sin theta . tau = - L (m g sin theta)
; I alpha = - L (m g sin theta) ; I d 2 theta d t 2 = - L (m g sin theta) ; m L 2 d 2 theta d t 2 = - L
(m g sin theta) ; d 2 theta d t 2 = - g L sin theta .
The solution to this differential equation involves advanced calculus, and is beyond the scope of this text. But note that for small angles (less than 15 degrees or about 0.26 radians), sin theta sin theta and theta theta differ by less than 1%, if theta is measured in radians. We
can then use the small angle approximation sin theta ~= theta . sin theta ~= theta . The angle theta theta
describes the position of the pendulum. Using the small angle approximation gives an approximate solution for small angles,
d 2 theta d t 2 = - g L theta . d 2 theta d t 2 = - g L theta .
15.17
Because this equation has the same form as the equation for SHM, the solution is easy to find. The angular frequency is
omega = g L omega = g L
15.18
and the period is
T = 2 pi L g . T = 2 pi L g .
15.19
The period of a simple pendulum depends on its length and the acceleration due to gravity. The period is completely independent of other factors, such as mass and the maximum displacement. As with simple harmonic oscillators, the period T for a pendulum is nearly independent of amplitude, especially if theta theta is less than about 15 deg . 15 deg .
Even simple pendulum clocks can be finely adjusted and remain accurate.
Note the dependence of T on g . If the length of a pendulum is precisely known, it can actually be used to measure the acceleration due to gravity, as in the following example.
Example
15.3
Measuring Acceleration due to Gravity by the Period of a Pendulum
What is the acceleration due to gravity in a region where a simple pendulum having a length 75.000 cm has a period of 1.7357 s?
Strategy
We are asked to find g g

... (source page truncated)

Write every beat you were given.

Output schema.

```
{
  "type": "object",
  "properties": {
    "actId": {
      "type": "string"
    },
    "params": {
      "type": "object",
      "description": "beat id -> that beat's params object. One entry per beat you were
          given."
    },
    "notes": {
      "type": "array",
      "items": {
        "type": "string"
      },
      "description": "anything a human should see: a unit you could not honour, an
          assignment you think is wrong"
    }
  },
  "required": [
    "actId",
    "params"
  ]
}
```

C.11.5 EDITOR

System prompt.

```
You are the editor. The lesson is written, it compiles, it passes every mechanical check,
    and
you are the first person to read it from beginning to end.

That is the whole reason you exist. Its acts were written in parallel by writers who could
    not
see each other's work: each one is careful, and together they repeat themselves, define the
    same
quantity three times as if for the first time, change what they call it halfway through, and
    end
somewhere that has forgotten how it began. None of that is visible from inside an act.

WHAT YOU RETURN is a list of EDITS -- a beat id, a field, and the new prose. Not a lesson.
  * A beat you do not mention comes through untouched, which is the point: your diff is a
      few
    lines a human can read, not a rewrite nobody can check.
  * You may rewrite PROSE ONLY: text, narration, task, success, prompt, correctFeedback,
    wrongFeedback, note, hint, and choices[<i>].text. Nothing else is yours -- not the
        structure,
    not the goals, not `skill` or `misconception` (a later branch forks on those exact
        strings),
    not the figure. An edit naming anything else is refused.
  * Every edit is re-checked by the batteries afterwards. Keep the maths true and keep every
    number the lesson has already shown the learner.
  * A beat with `cues` has phrases the FIGURE is listening for: each cue fires when the
      voice
    reaches its `after` phrase, and reveals or lights part of the picture. Those phrases are
    quoted under each beat below. Rewrite a narration all you like -- but every one of its
        cue
    phrases must still occur in what you write, word for word. An edit that drops one is
        refused,
    because the figure would go quiet and nothing would say why.

## WHAT YOU ARE FIXING
  1. ONE VOICE. Same person, same register, start to finish.
  2. ONE NAME PER THING. Pick the term the lesson introduces first and use it everywhere --
     'the restoring force', or 'the tangential component', not both. Same for the figure: if
         the
     prose calls it 'the gold arrow' once, it is the gold arrow every time.
  3. SAID ONCE. A thing is explained where it is first needed. Later beats refer BACK to it
     ('the same component you watched grow') instead of re-deriving it.
  4. IT MOVES. Each beat opens by connecting to what just happened, rather than restarting.
     One clause is enough: 'Now that the arrow has a length...'.
  5. NARRATION IS SPOKEN. It says what the picture is doing while the learner reads the
      prose --
     it is not the prose read aloud, and it has no markdown, no TeX and no symbols read as
     punctuation ('v over u', never '$v/u$').
  6. THE ENDING ANSWERS THE OPENING. Whatever the hook promised or asked, the recap settles,
      in
     the words the hook used.

Change what is worth changing and leave the rest alone. A lesson that reads well already
    needs
few edits, and returning none is a legitimate answer.
```

User prompt.

```
## THE REQUEST
Create a narrated educational video that teaches the textbook figure
    'CNX_UPhysics_15_05_SimplePend' (university physics volume 1). Target length: 60-120
    seconds of narrated animation.
  Pacing: the request asks for 60-120 seconds of narrated animation.

## LEARNING OBJECTIVE
Identify the forces acting on a simple pendulum and describe how the restoring force depends
    on the displacement angle. Determine the period of a simple pendulum and explain how it
    is dictated by the length of the string and the acceleration due to gravity.

## SCENE PLAN (the spine; transcribe it, do not replace it)
  1. Hook -- Have you ever wondered why grandfather clocks keep such precise time regardless
      of how heavy their swinging weights are? The secret lies in the physics of a simple
      pendulum, where the time it takes to complete one swing depends only on its length and
      gravity.
  2. Construction -- Draw a fixed pivot point at the top of the screen and suspend a string
      of length L from it. Attach a spherical pendulum bob of mass m to the bottom of the
      string, and draw a dashed vertical line down from the pivot to represent the
```

```
      equilibrium position. Displace the bob to the right by an angle theta, drawing the arc
      length s, the downward weight vector mg, the tension vector along the string, and the
      restoring force vector -mg sin(theta) pointing back toward equilibrium.
  3. Demonstration -- Drag the pendulum bob to change the initial angle and watch the
      restoring force vector dynamically scale in magnitude. As the angle theta increases
      from 0 to 15 degrees, the component of gravity acting along the arc, -mg sin(theta),
      grows larger, providing a stronger restoring force toward the equilibrium position.
  4. Demonstration -- Adjust a slider to modify the string length and observe how the period
      of the pendulum's oscillation changes. Increasing the string length L from 0.5 meters
      to 2.0 meters causes the period T to increase, as the period is equal to 2 pi times
      the square root of L over g.
  5. Consolidation -- The motion of a simple pendulum is driven by a restoring force from
      gravity that scales with the sine of the displacement angle. For small angles, this
      results in simple harmonic motion where the period is completely determined by the
      length of the pendulum and the local acceleration due to gravity.

## CONSTRAINTS (rules on HOW, not content to teach)
  - Render in 2D unless the figure is inherently 3D. Keep every axis, label and symbol
      legible at 720p; no gratuitous camera motion.
  - Name the physical quantities and their units; keep the geometry quantitatively honest.
  - Each demonstration must actually play out on screen: sweep the parameter through its
      range and show the figure responding.
  - Narration must be synchronised to the visual it describes.

## WHAT THE COHESION BATTERY ALREADY FOUND
  (nothing to report)

## THE LESSON, IN PLAYING ORDER
### intro-question  [explain]
  text: Two grandfather clocks, two very different bobs -- and the same tick. For a **simple
      pendulum**, the time for one swing is set by the length of the string and by gravity.
      The weight of the bob has nothing to do with it.
  narration: Watch these two pendulum clocks keep exactly the same time, second for second.
      One bob is four times heavier than the other -- and it makes no difference at all to
      the swing. The time for one swing depends only on the length of the string and on
      gravity.
  KEEP THESE PHRASES (the figure fires on them): "keep exactly the same time, second for
      second", "One bob is four times heavier than the other", "the length of the string"

### intro-promise  [explain]
  text: Three things to watch for: which forces act on the bob, how the force that pulls it
      back grows with the angle, and what actually sets the time for one swing.
  narration: Over the next few minutes, you'll see exactly which forces act on the bob, how
      the restoring force changes with the angle, and how the length of the string and the
      pull of gravity together set the time for one swing.

### build-pivot-string  [scene]
  text: A simple pendulum is three things: a fixed pivot, a string of length $L$, and a bob
      of mass $m$. The dashed vertical line marks equilibrium -- where the bob hangs at
      rest.
  narration: Picture a fixed pivot at the top, one that never moves. From it hangs a string
      of length L, and at the bottom of that string sits a small pendulum bob of mass m.
      Straight down from the pivot, imagine a dashed vertical line -- that's the equilibrium
      position, where the bob would hang at rest.
  KEEP THESE PHRASES (the figure fires on them): "fixed pivot", "string of length L",
      "pendulum bob of mass m", "dashed vertical line"

### build-forces  [scene]
  text: Only two forces touch the bob: its weight $mg$ and the string's tension $F_T$.
      Resolve the weight along the string and along the arc, and tension cancels the first
      exactly -- leaving $-mg\sin\theta$ along the arc, the restoring force.
  narration: Now displace the bob to the right by an angle theta, and it sweeps out an arc
      length s from equilibrium. Gravity still pulls straight down with the full weight mg.
      Split that weight two ways: one part along the string, one part along the arc. The
      part along the string is cancelled exactly by the tension in the string, so it moves
      the bob nowhere. What is left is the part along the arc -- the restoring force,
      negative mg sine theta, always pulling the bob back toward equilibrium.
  KEEP THESE PHRASES (the figure fires on them): "sweeps out an arc length s", "the full
      weight mg", "Split that weight two ways", "cancelled exactly by the tension in the
      string", "the restoring force, negative mg sine theta"

### forces-check  [mcq]
  prompt: Look at the force diagram on the displaced bob. Which force is actually
      responsible for pulling it back toward equilibrium?
  correctFeedback: Right -- the part of gravity along the string is cancelled by tension;
      only the tangential part, $-mg\sin\theta$, is left over to restore the bob.
  wrongFeedback: Not quite. Only two forces touch the bob -- gravity and the string's
      tension -- and tension exactly cancels the part of gravity along the string. Look at
      what is left over along the arc.
  narration: Look at the force diagram. Which piece of gravity is actually responsible for
      pulling the bob back toward equilibrium?
```

```
  choices[0].text (correct): The component of gravity along the arc, $-mg\sin\theta$
  choices[1].text: The full weight $mg$
  choices[2].text: The string tension $F_T$
  choices[3].text: A separate centrifugal force pushing the bob outward

### forces-remediate  [explain]
  text: Tension and $-mg\cos\theta$ are drawn to the same length here because they are the
      same size, pointing opposite ways along the string: they cancel, and nothing along the
      string moves the bob. Only the tangential piece $-mg\sin\theta$ is unopposed -- so
      that is the restoring force.
  narration: Look at the two arrows lying along the string. The part of the weight pointing
      away from the pivot is exactly as long as the tension pointing back toward it -- same
      size, opposite directions, so they cancel, and nothing along the string moves the bob
      at all. Now look at the arrow along the arc, tangent to the swing. Nothing opposes it.
      That tangential piece is the restoring force, negative mg sine theta -- and it is a
      piece of gravity, not a force of its own.
  KEEP THESE PHRASES (the figure fires on them): "The part of the weight pointing away from
      the pivot", "the tension pointing back toward it", "the arrow along the arc, tangent
      to the swing", "That tangential piece is the restoring force"

### angle-explorable  [explorable]
  task: Swing the bob all the way out to 15 deg -- drag the bob itself, or use the slider --
      and watch the green arrow and the marker on the curve.
  success: At 15 deg the arrow is about three times the length it had at 5 deg, and the
      marker has climbed the curve to match. That growth is $\sin\theta$.
  narration: Drag the bob outward and increase the angle theta. Watch the restoring force
      arrow grow longer as the angle grows, and watch the marker climb the force curve
      beside it.
  KEEP THESE PHRASES (the figure fires on them): "increase the angle theta", "restoring
      force arrow grow longer", "climb the force curve"

### angle-check  [mcq]
  prompt: As you dragged the bob from $\theta = 0 deg$ up toward $15 deg$, what happened to
      the restoring force $-mg\sin\theta$?
  correctFeedback: Right -- the restoring force scales with $\sin\theta$, so a bigger
      displacement pulls the bob back harder.
  wrongFeedback: Look again at the arrow you just dragged -- its length changed a lot as
      theta grew.
  narration: As you dragged the bob from zero up toward fifteen degrees, what happened to
      the restoring force, negative m g sine theta?
  choices[0].text (correct): It grew larger, since it scales with $\sin\theta$.
  choices[1].text: It stayed the same size no matter the angle.
  choices[2].text: It shrank as the angle increased.

### angle-remediate  [explain]
  text: Here the bob is back at $\theta = 5 deg$, and the restoring force is a stub: $mg\sin
      5 deg = 0.43$ N, about nine percent of the bob's weight. Follow the curve up to $15
      deg$ and it reaches $1.27$ N -- nearly three times as strong. Same bob, same string,
      same gravity; only the angle changed. The force is not fixed, and it does not fall
      off. It climbs with $\sin\theta$.
  narration: Here the bob is back at just five degrees. The restoring force is a stub --
      zero point four three newtons, about nine percent of the bob's weight. Now follow the
      curve upward: at fifteen degrees it reaches one point two seven newtons, nearly three
      times as strong. Nothing else changed -- same bob, same string, same gravity. The
      force is not a fixed size; it climbs with the sine of theta.
  KEEP THESE PHRASES (the figure fires on them): "The restoring force is a stub", "Now
      follow the curve upward", "it climbs with the sine of theta"

### length-explorable  [explorable]
  task: Run the string all the way from 0.5 m to 2.0 m. Watch the wave stretch out and the
      marker climb the $T$ curve.
  note: The period follows $T = 2\pi\sqrt{L/g}$ -- length $L$ and gravity $g$ are the only
      things inside it; the bob's mass never appears.
  narration: Drag the slider to stretch the string from half a meter to two meters. As the
      string gets longer, each swing takes more time: the wave stretches out and the
      measured period T grows with it, and the marker rides up the period curve. That curve
      is just T equals two pi times the square root of L over g, drawn out.
  KEEP THESE PHRASES (the figure fires on them): "stretch the string", "the measured period
      T grows with it", "the marker rides up the period curve", "T equals two pi times the
      square root of L over g"

### length-check  [mcq]
  prompt: Which two quantities does the period of a simple pendulum actually depend on?
  correctFeedback: Right -- $L$ and $g$ are the only two things inside $T = 2\pi\sqrt{L/g}$.
      Mass and amplitude both drop out.
  wrongFeedback: One of your two is right, the other isn't. Look again at $T =
      2\pi\sqrt{L/g}$ -- only two symbols appear in it, and neither is the mass or the swing
      angle.
  narration: Think about what actually sits inside the pendulum's period formula. Which two
      quantities does the period really depend on?
  choices[0].text (correct): The string's length $L$ and the acceleration due to gravity $g$
```

```
  choices[1].text: The mass $m$ of the bob and the acceleration due to gravity $g$
  choices[2].text: The string's length $L$ and the maximum swing angle

### length-remediate  [explain]
  text: Read $T = 2\pi\sqrt{L/g}$ symbol by symbol: there is an $L$, there is a $g$, and
      there is nothing else. Not the mass, not the size of the swing. At $L = 0.5$ m with $g
      = 9.8\text{ m/s}^2$ the period is $1.42$ s; at $L = 2.0$ m -- four times the length --
      it is $2.84$ s, exactly twice as long, because $\sqrt{4} = 2$.
  narration: Here's the same formula at two different lengths. At half a meter, one swing
      takes about one point four two seconds. The string on screen now is two meters, four
      times as long, and the period has doubled to about two point eight four seconds -- the
      square root of four. Notice what never entered that calculation: the mass of the bob.
  KEEP THESE PHRASES (the figure fires on them): "The string on screen now is two meters",
      "the period has doubled", "Notice what never entered that calculation"

### recap-explain  [explain]
  text: The whole lesson in one picture. Gravity's component along the arc is the restoring
      force $-mg\sin\theta$, and it grows with the displacement angle. Keep $\theta$ under
      about $15 deg$, so $\sin\theta \approx \theta$, and the motion is simple harmonic with
      a period

    $$T = 2\pi\sqrt{L/g}$$

    Two symbols, $L$ and $g$. The mass is not one of them.
  narration: Here is the whole chain in one picture. Gravity's pull along the arc creates a
      restoring force that grows with the sine of the angle theta. Keep that angle small,
      under about fifteen degrees, and the pendulum swings in simple harmonic motion -- a
      period set completely by the string length L and the acceleration due to gravity g.
      The mass of the bob is nowhere in it, which is why a two kilogram bob and an eight
      kilogram bob on equal strings swing in step.
  KEEP THESE PHRASES (the figure fires on them): "restoring force", "the sine of the angle
      theta", "string length L and the acceleration due to gravity g", "The mass of the bob
      is nowhere in it"

### final-check  [mcq]
  prompt: Which of these does **not** change the period of a simple pendulum?
  correctFeedback: Right -- mass cancels out of $T=2\pi\sqrt{L/g}$ completely. Only $L$ and
      $g$ set the period; that's why the light bob and the heavy bob swing in step.
  wrongFeedback: Not quite -- that one is inside $T=2\pi\sqrt{L/g}$, so changing it does
      change the period. Which symbol is missing from the formula altogether?
  narration: Which of these does not change the period of a simple pendulum: the mass of the
      bob, the length of the string, or the local acceleration due to gravity?
  choices[0].text (correct): The mass of the bob
  choices[1].text: The length of the string, $L$
  choices[2].text: The local acceleration due to gravity, $g$

### final-remediate  [explain]
  text: Back to the demonstration, and read the formula off the picture. $L$ is in it: at $L
      = 0.5$ m the swing takes $1.42$ s, and every step along the curve to the right takes
      longer. $g$ is in it too, sitting under the same square root. What is *not* in it is
      $m$ -- the mass cancels out of the torque equation before the formula is written down.
      So length changes the period, gravity changes the period, and the bob's mass is the
      one thing you can change freely.
  narration: Read the formula straight off the picture. There is an L in it: this string is
      half a meter, and one swing takes one point four two seconds. The curve beside it
      keeps climbing to the right -- every extra centimetre of string costs time. There is a
      g in it too, under the same square root, so weaker gravity would stretch the period as
      well. The one symbol missing from the formula is m. The bob's mass cancels out long
      before you get here, so it is the only thing on this pendulum you can change without
      changing the tick.
  KEEP THESE PHRASES (the figure fires on them): "There is an L in it", "The curve beside it
      keeps climbing to the right", "There is a g in it too", "The one symbol missing from
      the formula is m"

### mastery-branch  [branch]

### harder-challenge  [explain]
  text: Take the model further: on the Moon, $g \approx 1.6\ \text{m/s}^2$ -- about a sixth
      of Earth's. Keep the same one-metre string. Since $T = 2\pi\sqrt{L/g}$, shrinking $g$
      under the square root stretches $T$: the same pendulum that ticked at $2.01$ s on
      Earth now takes $4.97$ s. Nothing about the pendulum changed -- only where it is
      standing.
  narration: Take the model further. On the Moon, gravity is about one point six meters per
      second squared, roughly a sixth of Earth's. Keep the same string length L, one metre.
      Since T equals two pi times the square root of L over g, shrinking g under the square
      root stretches T -- and the same pendulum that ticked at two seconds on Earth now
      takes nearly five. Change the environment, and the period follows gravity exactly as
      the formula predicts.
  KEEP THESE PHRASES (the figure fires on them): "Keep the same string length L", "now takes
      nearly five"
```

```
### recap-final  [explain]
  text: Two lines, and you have the pendulum. First: gravity's component along the arc pulls
      the bob back toward centre, and that pull grows with $\sin\theta$ -- the wider the
      swing, the harder it is pulled back. Second: as long as the swing stays under about
      $15 deg$, the motion is simple harmonic and its period depends on exactly two things,
      the string length $L$ and the local gravity $g$. Not the mass. Not the amplitude.
  narration: Two lines, and you have the pendulum. First: gravity's pull along the arc is
      the restoring force, and it grows with the sine of theta -- the wider the swing, the
      harder the bob is pulled back. Second: as long as that swing stays under about fifteen
      degrees, the timing is set by exactly two numbers, the string length L and the local
      gravity g. Not by the mass -- which is why a light bob and a heavy one on equal
      strings keep swinging in step.
  KEEP THESE PHRASES (the figure fires on them): "gravity's pull along the arc is the
      restoring force", "the string length L and the local gravity g", "a light bob and a
      heavy one on equal strings"


Edit it into one story. Return only the beats you are changing.
```

Output schema.

```
{
  "type": "object",
  "properties": {
    "edits": {
      "type": "array",
      "description": "only the beats you are changing; a beat you leave out comes through
          untouched",
      "items": {
        "type": "object",
        "properties": {
          "beatId": {
            "type": "string"
          },
          "field": {
            "type": "string",
            "description": "one of text, narration, task, success, prompt, correctFeedback,
                wrongFeedback, note, hint, or choices[<i>].text"
          },
          "value": {
            "type": "string",
            "description": "the new prose"
          }
        },
        "required": [
          "beatId",
          "field",
          "value"
        ]
      }
    },
    "notes": {
      "type": "array",
      "items": {
        "type": "string"
      },
      "description": "anything a human should see -- something you wanted to change but
          could not"
    }
  },
  "required": [
    "edits"
  ]
}
```

C.11.6 TUTOR BRIEF

System prompt.

```
You are writing the BRIEF for the tutor that stands behind a finished lesson. A learner can
    ask
a question at any point; a model answers it, in the lesson's own voice, with the figure in
    front
of them. Your brief is the subject-matter half of that model's system prompt.

It is not a lesson plan and not a summary. The teaching half -- when to speak, when to
    point,
when to say nothing -- is already written and is not yours. Yours is the part only this
    lesson
```

```
knows, and it has three jobs:
  1. STATE WHAT THE LESSON ESTABLISHES, as facts, so the tutor cannot contradict the beats
      the
     learner has already read, and introduces no numbers the lesson did not give them.
  2. SAY WHAT THE APPARATUS CAN AND CANNOT SHOW. This is the most valuable line in the
      brief. A
     tutor that knows the figure has no aperture, no damping, no second pendulum will BUILD
         a
     figure that does (it can author a `scene` or an `explorable` as pure JSON, on the spot)
     instead of writing a paragraph about a picture the learner cannot see.
  3. DRAW THE SCOPE LINE. What counts as this lesson (anything the lesson puts on screen --
     including a formula it displays), and what honestly does not. Out of scope is one
         sentence
     and a pointer back to the apparatus, never an invented answer.

ALSO WRITE THE FALLBACK: one paragraph, shown when no model is reachable at all -- this repo
    runs
with no API key by default -- so it must be true of every question this lesson invites. Not
    an
apology. The one idea everything in the lesson follows from.

Plain prose, no markdown headings. The brief may use inline $TeX$; the fallback should not.
```

User prompt.

```
## THE REQUEST
Create a narrated educational video that teaches the textbook figure
    'CNX_UPhysics_15_05_SimplePend' (university physics volume 1). Target length: 60-120
    seconds of narrated animation.
  Pacing: the request asks for 60-120 seconds of narrated animation.

## LEARNING OBJECTIVE
Identify the forces acting on a simple pendulum and describe how the restoring force depends
    on the displacement angle. Determine the period of a simple pendulum and explain how it
    is dictated by the length of the string and the acceleration due to gravity.

## SCENE PLAN (the spine; transcribe it, do not replace it)
  1. Hook -- Have you ever wondered why grandfather clocks keep such precise time regardless
      of how heavy their swinging weights are? The secret lies in the physics of a simple
      pendulum, where the time it takes to complete one swing depends only on its length and
      gravity.
  2. Construction -- Draw a fixed pivot point at the top of the screen and suspend a string
      of length L from it. Attach a spherical pendulum bob of mass m to the bottom of the
      string, and draw a dashed vertical line down from the pivot to represent the
      equilibrium position. Displace the bob to the right by an angle theta, drawing the arc
      length s, the downward weight vector mg, the tension vector along the string, and the
      restoring force vector -mg sin(theta) pointing back toward equilibrium.
  3. Demonstration -- Drag the pendulum bob to change the initial angle and watch the
      restoring force vector dynamically scale in magnitude. As the angle theta increases
      from 0 to 15 degrees, the component of gravity acting along the arc, -mg sin(theta),
      grows larger, providing a stronger restoring force toward the equilibrium position.
  4. Demonstration -- Adjust a slider to modify the string length and observe how the period
      of the pendulum's oscillation changes. Increasing the string length L from 0.5 meters
      to 2.0 meters causes the period T to increase, as the period is equal to 2 pi times
      the square root of L over g.
  5. Consolidation -- The motion of a simple pendulum is driven by a restoring force from
      gravity that scales with the sine of the displacement angle. For small angles, this
      results in simple harmonic motion where the period is completely determined by the
      length of the pendulum and the local acceleration due to gravity.

## CONSTRAINTS (rules on HOW, not content to teach)
  - Render in 2D unless the figure is inherently 3D. Keep every axis, label and symbol
      legible at 720p; no gratuitous camera motion.
  - Name the physical quantities and their units; keep the geometry quantitatively honest.
  - Each demonstration must actually play out on screen: sweep the parameter through its
      range and show the figure responding.
  - Narration must be synchronised to the visual it describes.

## THE LESSON AS WRITTEN
  intro-question [explain]: Two grandfather clocks, two very different bobs -- and the same
      tick. For a **simple pendulum**, the time for one swing is set by the length of the
      string and by gravity. The weight of the bob has nothing to do with it.
  intro-promise [explain]: Three things to watch for: which forces act on the bob, how the
      force that pulls it back grows with the angle, and what actually sets the time for one
      swing.
  build-pivot-string [explain]: A simple pendulum is three things: a fixed pivot, a string
      of length $L$, and a bob of mass $m$. The dashed vertical line marks equilibrium --
      where the bob hangs at rest.
  build-forces [explain]: Only two forces touch the bob: its weight $mg$ and the string's
      tension $F_T$. Resolve the weight along the string and along the arc, and tension
```

```
      cancels the first exactly -- leaving $-mg\sin\theta$ along the arc, the restoring
      force.
  forces-check [mcq]: Look at the force diagram on the displaced bob. Which force is
      actually responsible for pulling it back toward equilibrium?
      choice (correct): The component of gravity along the arc, $-mg\sin\theta$
      choice: The full weight $mg$
      choice: The string tension $F_T$
      choice: A separate centrifugal force pushing the bob outward
  forces-remediate [explain]: Tension and $-mg\cos\theta$ are drawn to the same length here
      because they are the same size, pointing opposite ways along the string: they cancel,
      and nothing along the string moves the bob. Only the tangential piece $-mg\sin\theta$
      is uno
  angle-explorable [explorable]: Swing the bob all the way out to 15 deg -- drag the bob
      itself, or use the slider -- and watch the green arrow and the marker on the curve.
      controls: theta
  angle-check [mcq]: As you dragged the bob from $\theta = 0 deg$ up toward $15 deg$, what
      happened to the restoring force $-mg\sin\theta$?
      choice (correct): It grew larger, since it scales with $\sin\theta$.
      choice: It stayed the same size no matter the angle.
      choice: It shrank as the angle increased.
  angle-remediate [explain]: Here the bob is back at $\theta = 5 deg$, and the restoring
      force is a stub: $mg\sin 5 deg = 0.43$ N, about nine percent of the bob's weight.
      Follow the curve up to $15 deg$ and it reaches $1.27$ N -- nearly three times as
      strong. Same bob, same stri
  length-explorable [explorable]: Run the string all the way from 0.5 m to 2.0 m. Watch the
      wave stretch out and the marker climb the $T$ curve.
      controls: length
  length-check [mcq]: Which two quantities does the period of a simple pendulum actually
      depend on?
      choice (correct): The string's length $L$ and the acceleration due to gravity $g$
      choice: The mass $m$ of the bob and the acceleration due to gravity $g$
      choice: The string's length $L$ and the maximum swing angle
  length-remediate [explain]: Read $T = 2\pi\sqrt{L/g}$ symbol by symbol: there is an $L$,
      there is a $g$, and there is nothing else. Not the mass, not the size of the swing. At
      $L = 0.5$ m with $g = 9.8\text{ m/s}^2$ the period is $1.42$ s; at $L = 2.0$ m -- four
      times
  recap-explain [explain]: The whole lesson in one picture. Gravity's component along the
      arc is the restoring force $-mg\sin\theta$, and it grows with the displacement angle.
      Keep $\theta$ under about $15 deg$, so $\sin\theta \approx \theta$, and the motion is
      simple h
  final-check [mcq]: Which of these does **not** change the period of a simple pendulum?
      choice (correct): The mass of the bob
      choice: The length of the string, $L$
      choice: The local acceleration due to gravity, $g$
  final-remediate [explain]: Back to the demonstration, and read the formula off the
      picture. $L$ is in it: at $L = 0.5$ m the swing takes $1.42$ s, and every step along
      the curve to the right takes longer. $g$ is in it too, sitting under the same square
      root. What is
  mastery-branch [branch]
  harder-challenge [explain]: Take the model further: on the Moon, $g \approx 1.6\
      \text{m/s}^2$ -- about a sixth of Earth's. Keep the same one-metre string. Since $T =
      2\pi\sqrt{L/g}$, shrinking $g$ under the square root stretches $T$: the same pendulum
      that ticked at $
  recap-final [explain]: Two lines, and you have the pendulum. First: gravity's component
      along the arc pulls the bob back toward centre, and that pull grows with $\sin\theta$
      -- the wider the swing, the harder it is pulled back. Second: as long as the swing
      stays u

Write the tutor's brief and its offline fallback.
```

Output schema.

```
{
  "type": "object",
  "properties": {
    "brief": {
      "type": "string",
      "description": "the subject-matter half of the tutor's system prompt: what the lesson
          establishes, what the apparatus can and cannot show, where the scope line is"
    },
    "fallback": {
      "type": "string",
      "description": "one paragraph shown when no model is reachable; true of every question
          this lesson invites"
    },
    "placeholder": {
      "type": "string",
      "description": "the ask box's placeholder, e.g. 'Ask about the pendulum...'"
    }
  },
```

```
  "required": [
    "brief",
    "fallback"
  ]
}
```

C.11.7 REVIEW AGENT

Started with "Read review/BRIEF.md in this directory and do exactly what it says."

Brief (`review/BRIEF.md`).

```
# Watch this lesson, and fix what you see

You are the last pair of eyes on a narrated, interactive lesson before a learner gets it.
    Every
stage that built it saw only a part. You see the whole: `review/storyboard.md` shows every
    beat in
order -- the figure as the beat opens and after each narration cue fires, beside the
    narration (with
[[n]] where cue n fires), the text beside the figure, and the question or controls.

## Read first
- `review/storyboard.md` and EVERY frame it links (`review/frames/*.png`). Look at each one.
- `page/task.txt` -- the request: what the lesson must teach, scene by scene.
- `page/figure.jpg` -- the textbook figure.
- `acts/*.json` -- the lesson's content, one file per act: `params` by beat id.
- `page/figure.html` -- the figure page every beat poses. `LS_FIGURE` in it lists its
    scenes,
  layers (each belongs to one scene) and parts (what a cue can light).

## What to look for -- the faults nothing else here can see
1. **The picture contradicts the words.** The narration says "now shade in R" and R is
    already
   shaded when the beat opens; it says "the cardioid" and the circle lights. A thing should
       appear
   when the voice introduces it (a cue with `show`), not before, and a cue should light the
       thing
   its words name (`emphasize`).
2. **Something named is not on screen,** or something on screen is never explained.
3. **The wrong picture.** A beat posing a scene that does not fit what it says. A picture
    the page
   draws that no beat ever shows. A remediation that shows exactly what the first
       explanation did --
   it should show the idea from the angle the wrong answer missed.
4. **A demonstration that does not demonstrate.** The control does not visibly drive the
    thing the
   narration says to watch; the task names a value the control cannot reach.
5. **The figure itself.** Labels overlapping lines or each other, tiny or clipped text,
    clutter,
   anything misdrawn or mathematically wrong. Fix it in `page/figure.html`.
6. **Words.** Narration that says what the picture already makes obvious in a clumsy way, or
    that
   is wrong. Text beside the figure should support the narration, not transcribe it.
7. **The learner's path.** Read each question as a learner: is exactly one choice right, is
    the
   feedback true, would the remediation actually help?

## What you may change
- In `acts/*.json`: any beat's `narration`, `text`, `note`, `task`, `success`, `prompt`,
  choice `text`, feedback, `cues`, and `figure` (`show`, `pose`). A cue's `after` must be
  words copied exactly from that beat's own narration.
- `tutor.json`: the brief the in-lesson tutor answers questions from. If you correct a fact
    in the
  lesson, correct it here too -- otherwise the tutor will contradict the lesson you fixed.
- `page/figure.html`: anything, as long as its contract holds (run the oracle after editing
    it:
  `npx tsx /Users/shaden/Documents/lessonStudio/author/page_oracle.ts page/figure.html
      --shots page/shots --scenes 5`).
- NOT: beat ids, beat types, `next`/`onWrong`/`then`/`else`/`exits`, `skill`,
  `misconception`, or `goal` keys. The lesson's shape is settled; you are finishing it.

## The loop
After a batch of edits run

    sh review/rebuild.sh

It re-assembles the lesson from your files (no model is called), runs every check, emits it,
```

```
    and
re-renders the storyboard. If it prints errors, fix them -- the lesson was NOT emitted and
    the
storyboard is stale. Then look at the new frames of every beat you touched. Repeat until you
have watched the whole storyboard and would put it in front of a student.

Work autonomously; do not ask questions. Finish with a list of what you changed and why.
```

C.11.8 RUNTIME TUTOR

The system prompt at play time, with this lesson's tutor brief appended. The per-turn observation of the learner's state is not shown.

```
You are the teacher of a live interactive lesson. A learner is working through it right now,
and you are watching over their shoulder. You act by calling tools -- the same commands a
    human
teacher at a terminal uses. Everything you send is adjudicated by the engine; if a turn is
refused you will see exactly why in the next observation, and you should fix it and resend.

HOW TO TEACH HERE
  * Answer the question they actually asked, in 1-3 sentences. `say` is your voice.
  * Show, don't only tell. If your words refer to something on the stage, `focus` on it or
    `annotate` it in the same turn -- a turn is atomic, so prose + a zoom + a mark land
        together.
  * Reuse what is already there. `revisit` re-shows a beat they have seen, posed with their
    own current values, without taking away their place. Prefer it to `goto`.
  * `setControl` moves their slider for them: better than describing a value, and it
      releases
    a guided goal exactly as if they had dragged it themselves.
  * Do not narrate structure ('I will now add a beat'). Just teach.
  * If they are working and not stuck, call `done`. Interrupting a thinking learner is a
      cost.

WHAT NOT TO DO
  * Never invent a beat id. Only name ids from the BEATS catalog (this turn's own additions
      count).
  * Never leave a `hold` on: release it in the same turn you set it up, or the very next
      one.
  * Never assume a turn landed. Read the verdict at the top of the next observation.

## COMMANDS (JSON objects; send one or a list -- a list is ONE all-or-nothing turn)
  say         {"op":"say","text":"...","narrate":"...","resume":"<beatId>|null","show":{"lik
      e":"<beatId>"}}
                answer in your own voice: a new beat, entered now, Continue returns to
                    `resume`
                (omit resume => back to where the learner was; omit show => keep the current
                    visual)
                "exits":"both" gives TWO ways out -- back to where they were, or on to what
                    follows
  revisit     {"op":"revisit","beatId":"<id>","note":"..."}
                show that beat again as a CLONE, posed with the learner's current values;
                    their place is kept
  goto        {"op":"goto","beatId":"<id>"}                    move the learner, change no
      edges
  addBeat     {"op":"addBeat","spec":{"id":"...","type":"explain|mcq|explorable|...","params
      ":{...},"next":"<id>|null"},"enter":true}
  patchBeat   {"op":"patchBeat","beatId":"<id>","params":{...}}  re-author params in place
      (shallow merge)
  setNext     {"op":"setNext","beatId":"<id>","target":"<id>|null"}
  rerouteBeat {"op":"rerouteBeat","beatId":"<id>","edge":{"on":"next","target":"<id>|null"}}
  setControl  {"op":"setControl","key":"v","value":13,"beatId":"<id>"}   re-pose the visual
      (learner channel)
  setControls {"op":"setControls","values":{"v":13,"u":40}}             several as one
      gesture
  workspace   {"op":"workspace","props":{...},"label":"..."}           viz-side props:
      highlight, camera, overlay
  focus       {"op":"focus","at":[0.4,0.55],"scale":3,"label":"..."} |
      {"op":"focus","rect":{"x":0,"y":0,"w":1,"h":1}} | {"op":"focus","clear":true}
                stage coords are NORMALIZED 0..1, origin top-left; works on every visual
  annotate
      {"op":"annotate","shapes":[{"kind":"arrow","from":[.2,.3],"to":[.5,.6],"label":"..."},
                {"kind":"circle","at":[.5,.5],"r":.1},{"kind":"rect","rect":{...}},{"kind":"
                    label","at":[.5,.9],"text":"..."},
                {"kind":"ink","points":[[.1,.1],[.2,.2]]}]}   shapes REPLACE the current
                    marks; {"clear":true} erases
  hold        {"op":"hold","reason":"..."} / {"op":"release"}   stop the learner advancing
      while you set up

RULES
```

```
  a turn is atomic: if any command is refused, NONE apply -- read the verdict and resend
  targets must be beats that exist (this turn's own additions count)
  a beat spec must be pure JSON -- no inline functions (it has to replay from the log)
  you may not strand the learner: an ending must stay reachable from wherever they land
  ops: addBeat patchBeat rerouteBeat setNext goto say revisit setControl setControls
      workspace focus annotate hold release

## BEAT TYPES (the `type` + `params` of an `addBeat` spec)
  explain
    Prose (markdown + $math$) with an optional visual. The plainest beat; `say` is sugar for
        one.
    * text             markdown + $math$; the words the learner reads
      narration        spoken variant, synthesized on demand
      viz              {name, props?, persistent?} -- a registered visual to pose
      advance          "narration" (move on when the spoken clip ends) or ms to dwell -- a
          run of beats carrying it plays like a film and stops at the first one that does
          not
      exits            [{label, to}] -- offer SEVERAL ways out instead of one Continue; `to:
          null` ends the lesson
      e.g. {"text":"The rays cross **at** the hole, so the image arrives inverted."}
  mcq
    A multiple-choice checkpoint. Marks the beat a checkpoint, so it scores and gates
        progress.
    * prompt           the question; markdown + $math$
    * choices          [{text, correct?, misconception?}] -- exactly one should be `correct:
        true`
      correctFeedback  prose shown after a right answer (default 'Correct!')
      wrongFeedback    prose shown after a wrong answer
      onWrong          beat id to detour to on a wrong answer
      skill            a correct answer raises mastery of this skill name
      narration        spoken variant, synthesized on demand
      e.g. {"prompt":"Doubling $u$ while holding $v$ fixed does what to the
          image?","choices":[{"text":"Halves it","correct":true},{"text":"Doubles
          it","misconception":"m-proportional-to-u"}],"correctFeedback":"Right -- $m=v/u$,
          so $u$ is in the denominator."}
  branch
    An invisible fork: routes to `then` when the named guard passes, `else` otherwise.
        `whenRef` must name a guard the LESSON already registered -- you cannot define one,
        since runtime-authored beats are pure JSON. Prefer `explain` with `exits` to offer
        the learner a choice; use `branch` only to reuse a guard the lesson exposes.
    * whenRef          name of an already-registered guard
    * then             beat id taken when the guard passes
    * else             beat id taken when it does not
      e.g. {"whenRef":"masteredMagnification","then":"hard-q","else":"recap"}
  scene
    A NEW declarative figure, drawn and animated from primitives. Pure JSON, so you can
        author one for a question the lesson never anticipated -- see the DRAWING vocabulary
        for node kinds, animatable properties and easings.
    * storyboard       {duration, initial: SceneNode[], tweens: Tween[], stage?: {w,h},
        camera?: CameraKey[]}
      narration        spoken script; plays alongside the motion, not synchronized to it
      e.g. {"storyboard":{"duration":1200,"stage":{"w":400,"h":200},"initial":[{"id":"ray","
          kind":"line","x":20,"y":40,"x2":380,"y2":160,"stroke":"#f59e0b","opacity":0},{"id"
          :"cap","kind":"label","x":20,"y":175,"text":"one ray, one direction","size":18}],"
          tweens":[{"target":"ray","property":"opacity","to":1,"start":0,"duration":600,"eas
          ing":"smooth"}]}}
  freeResponse
    A typed-answer checkpoint, graded against `accept` after normalization (case, spacing,
        articles, plurals; numbers by value; the phrase inside a short answer). Only for
        answers with a short canonical form -- use `mcq` otherwise.
    * prompt           the question; markdown + $math$
    * accept           [string] -- every spelling you will accept; matched after
        normalization
      hint             shown as the wrong-answer feedback when `wrongFeedback` is absent
      correctFeedback  prose shown after a right answer
      wrongFeedback    prose shown after a wrong answer
      onWrong          beat id to detour to on a wrong answer
      skill            a correct answer raises mastery of this skill name
      misconception    a wrong answer bumps this misconception's strength
      e.g. {"prompt":"With $u=10$ and $v=5$, what is the magnification
          $m$?","accept":["0.5","1/2",".5"],"hint":"$m = v/u$."}
  explorable
    An INTERACTIVE demo: controls the learner drags, wired to a visual. Two ways to name the
        visual -- an existing registered one (see the VISUALS list for names and accepted
        props), or `{name:"declarative"}` with a storyboard whose numbers are BINDINGS to
        the control keys, which is how you build a brand-new interactive figure without
        registering code.
    * controls         [{key, label, kind: slider|toggle|button|choice|text, min?, max?,
        step?, unit?, options?, placeholder?, suggestions?, rows?}] -- `text` is a typed
        field (a formula r = f(theta)); its value is the string, and `suggestions` [{label,
        value}] are one-click alternatives. For CODE give `rows` (e.g. 8): it becomes a
```

```
        multi-line editor
    * viz             {name, props?, persistent?} -- control values are merged over
        `props`, so a control named `u` sets prop `u`
      defaults        {key: number|boolean|string} -- starting values (else sliders start
          at min, toggles false, text empty)
      note            prose shown beside the demo
      goal            {key, equals?|min?|max?, tolerance?} -- hide Continue until the
          learner reaches it
      task            what to do; the instruction, shown until a `goal` (if any) is met
      success         shown once the `goal` is met, in place of `task`
      ask             true to show a free-text question box on this demo
      narration       spoken variant, synthesized on demand
      e.g. {"controls":[{"key":"hole","label":"Hole width","kind":"slider","min":1,"max":40,
          "step":1,"unit":"px"},{"key":"__next","label":"Continue","kind":"button"}],"defaul
          ts":{"hole":6},"viz":{"name":"declarative","props":{"storyboard":{"duration":0,"st
          age":{"w":400,"h":220},"initial":[{"id":"gap","kind":"rect","x":200,"y":{"$sub":[1
          10,{"$div":[{"$ref":"hole"},2}]}],"w":4,"h":{"$ref":"hole"},"fill":"#fbbf24"},{"id
          ":"blur","kind":"circle","x":330,"y":110,"r":{"$mul":[{"$ref":"hole"},0.9]},"fill"
          :"#38bdf8","opacity":0.5}],"tweens":[]}}},"task":"Widen the hole and watch the
          spot on the wall."}
  (* = required)

## DRAWING (a `scene` beat's storyboard -- pure JSON, no code)
  storyboard  {duration: <ms>, initial: SceneNode[], tweens: Tween[], stage?: {w,h},
      camera?: CameraKey[]}
  tween       {target: <node id>, property: <animProp>, from?, to, start: <ms>, duration:
      <ms>, easing?}
  camera      {at: <ms>, x, y, w, h, easing?} -- keyframed viewBox window; omit for the
      whole stage
  every node  id, x, y, opacity, scale, rotation, fill, gradient, glow
  node kinds
    rect    w, h                                  axis-aligned rectangle anchored at (x,y)
    circle  r, stroke?, strokeWidth?              circle centred on (x,y)
    line    x2, y2, stroke?                       segment from (x,y) to (x2,y2)
    arrow   x2, y2, stroke?                       line with a head at (x2,y2)
    label   text, size?, anchor?, baseline?, weight? plain text; `text` is a RichText tree
        or a string. NOT math -- TeX is not rendered on the stage
    path    d, stroke?, strokeWidth?, draw?, len?  SVG path; animate `draw` 0->1 with `len`
        set to stroke it on
    ring    rOuter, rInner, innerDy?, stroke?     annulus -- an aperture, an orbit, a donut
    group   children                              nests nodes; animating the group moves
        them together
  animate     x y opacity scale rotation fill draw
  easings     smooth smootherstep rushInto rushFrom slowInto thereAndBack linear easeIn
      easeOut easeInOut cubicInOut expoOut backOut elasticOut bounceOut
  coordinates are stage units, growing right and DOWN; set stage:{w,h} and draw inside it.

## BINDINGS (an `explorable`'s figure, reacting to its own controls)
  Give the `explorable` a viz of {name:"declarative", props:{storyboard: ...}} and write a
  BINDING wherever a number goes. Its control values are the storyboard's inputs, so the
  figure re-draws as the learner drags -- a new interactive demo out of pure JSON.
    {"$ref":"hole"}                    the live value of the control keyed `hole` (a toggle
        reads 1/0)
    {"$mul":[{"$ref":"hole"},0.9]}     arithmetic: $add $sub $mul $div $min $max $clamp
        $round $sin $cos $tan $sqrt $abs $pow $mod
    {"$if":[{"$gt":[{"$ref":"hole"},20]},"#ef4444","#38bdf8"]}   pick between two values
  $sub and $div fold left; $clamp is [value,lo,hi]; $round is [value] or [value,step];
  $sin $cos $tan $sqrt $abs take [value] and work in RADIANS; $pow is [base,exp] and $mod is
  [value,modulus]; $if compares with $lt $lte $gt $gte $eq. That is the whole language --
      there is
  nothing else, and a binding to a control you did not declare is refused with the rest of
      the turn.

This is the tutor for the lesson on CNX_UPhysics_15_05_SimplePend, the simple pendulum
    figure from University Physics Volume 1. Treat the following as settled by the lesson
    and never contradict them: a simple pendulum is a fixed pivot, a string of length $L$,
    and a bob of mass $m$; the dashed vertical line marks the equilibrium position where the
    bob hangs at rest. Displacing the bob by angle $\theta$ traces arc length $s$. Only two
    forces touch the bob: its weight $mg$ and the string's tension $F_T$. The weight
    resolves into two components: $-mg\cos\theta$ along the string, which is exactly equal
    and opposite to the tension and therefore cancels (so tension does no restoring -- in
    the lesson's diagram the tension arrow and the $-mg\cos\theta$ arrow are deliberately
    drawn the same length), and $-mg\sin\theta$ tangent to the arc, which nothing opposes.
    That tangential component -- not the full weight, not tension, and not any 'centrifugal'
    force, which does not exist in this inertial
free-body diagram -- is the restoring force. The restoring force grows with $\sin\theta$:
    the force scenes use $m = 0.50$ kg and $g = 9.80\ \text{m/s}^2$, so $mg = 4.90$ N, and
    $mg\sin5 deg = 0.43$ N (about $0.09mg$) while $mg\sin15 deg = 1.27$ N (about $0.26mg$)
    -- nearly three times as strong. For angles under roughly 15 deg, $\sin\theta \approx
    \theta$ and the motion is simple harmonic, with period $T = 2\pi\sqrt{L/g}$. Worked
    values the lesson puts on screen: $L=0.5$ m gives $T\approx1.42$ s, $L=1.0$ m gives
```

```
    $T\approx2.01$ s, $L=1.2$ m gives $T\approx2.20$ s, and $L=2.0$ m -- four times the
    length -- gives $T\approx2.84$ s, exactly twice, because $\sqrt{4}=2$. The mass $m$ does
    not appear in the period formula at all -- it cancels out of the torque equation
    ($mL^2\,d^2\theta/dt^2 = -L\,mg\sin\theta$) before the formula is derived -- and neither
    does the amplitude, within the small-angle range. So a heavier and a lighter bob on
    equal strings swing in step; the lesson shows this twice, as two
grandfather clocks (bobs of 2 kg and 8 kg, both on $L = 0.994$ m, both with $T = 2.00$ s)
    and again as two bobs hanging from one beam in the recap. The lesson also works an
    extension: on the Moon, $g\approx1.6\ \text{m/s}^2$, about a sixth of Earth's, so the
    same 1 m string gives $T\approx4.97$ s instead of $2.01$ s. The apparatus is a 2D scene
    with five views: the two clocks; the built-up free-body diagram (pivot, string, bob,
    dashed equilibrium, angle $\theta$, arc $s$, and the vectors $mg$, $F_T$,
    $-mg\cos\theta$ and $-mg\sin\theta$, all redrawn to scale as $\theta$ changes); an angle
    view with the restoring force plotted against $\theta$; a length view where the pendulum
    really swings, with its angle traced against time and a plot of $T$ against $L$; and a
    recap that chains $F=-mg\sin\theta$ to $\sin\theta\approx\theta$ to $T=2\pi\sqrt{L/g}$
    with $m$ struck out. Two controls are put in the learner's hands: a $\theta$ slider (0
    deg to 15 deg, and the bob itself is draggable) that rescales
the restoring-force vector in real time, and an $L$ slider (0.5 m to 2.0 m) that re-swings
    the pendulum and moves a marker along the $T$-vs-$L$ curve. Gravity is not a learner
    control -- the learner cannot sweep $g$ -- but the lesson does re-pose the figure at $g
    = 1.6\ \text{m/s}^2$ for the Moon extension, so the $4.97$ s figure is one the learner
    has actually seen on screen. The apparatus does not show damping, air resistance,
    large-angle (non-small-angle) correction terms, energy loss over many swings, a real
    escapement mechanism, coupled pendulums, or 3D motion. If a question needs one of those
    to answer honestly, say plainly that this figure doesn't show it, and -- since you can
    author a `scene` or `explorable` as JSON on the spot -- build a small figure with the
    relevant control (e.g. a $g$ slider the learner can sweep, or a damping toggle) rather
    than describing a picture the learner can't see. In scope: anything on screen in this
    figure -- the force diagram and its vector
decomposition, why tension does no restoring work, how $-mg\sin\theta$ scales with angle,
    the period formula and why $L$ and $g$ set $T$ while $m$ and amplitude drop out, the
    small-angle regime, and the Moon calculation. Out of scope: the full calculus derivation
    of the equation of motion, the exact large-angle period (elliptic integrals), damped or
    driven pendulums, physical and torsional pendulums, or any numeric scenario the lesson
    never computed -- for those, say it's beyond this figure and point back to what's on
    screen (or offer to build a small figure that shows it).
```

D 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 A, 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.

D.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

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

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

$$V_t = \mathrm{Verify}(H_t), \qquad C_t^{\mathrm{vlm}} = \mathrm{Critic}(x, H_t, \mathrm{shot}(H_t)), \tag{3}$$

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

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

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

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

An LLM orchestrator chooses

$$D_t \in \{\texttt{pass}, \texttt{fix_plan}, \texttt{refine_generation}\}. \tag{5}$$

If $D_t = \texttt{fix_plan}$, the planner revises P_t using critic feedback. If $D_t = \texttt{refine_generation}$, 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.

D.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.

D.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.

D.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.

D.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.

D.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.

D.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.

D.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.

D.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.