Ludwig: Autonomous Multilingual Synthesis of Correct, Engaging Explainer Videos

Alexander Krentsel
UC Berkeley
Berkeley, CA, USA

Abstract

Building the right mental model is central to teaching and understanding systems. Well-animated explainer videos help build these mental models effectively, but are expensive and time-consuming to produce, requiring pedagogical design, scripting, animation, narration, and review work. We argue that recent capability shifts enable multi-modal models to effectively produce such content by autonomously authoring animation code from a style guide, auditing rendered frames for defects, and applying multilingual neural text-to-speech.

We prototype an agent-driven system, Ludwig, structured around a highly capable producer agent that plans, narrates, and animates, and an independent vision-model reviewer that closes the loop by auditing the rendered output. Run for two months as a public channel, Ludwig has produced 150 videos across five languages with over 45,000 views at $2 to $10 and about 25 minutes of wall-clock time each, orders of magnitude below a human-authored equivalent. We discuss the implications of such a system for systems education. All videos are publicly available on the live channel at www.youtube.com/@ludwigexplains.

1 Introduction

A core goal of computer science pedagogy is to give students a robust mental model of how computer systems work that generalizes to new scenarios. This holds across all classes, but is especially true in systems classes, where we try to provide intuition about the behavior of large-scale systems, like global-scale protocols, congestion control, and routing on the internet. Students learn in a range of ways, and engaging, well-animated video is an effective medium for building the foundational mental models. Channels such as 3Blue1Brown, Veritasium, and Khan Academy have shown that genuinely engaging visual instruction works at scale [7, 11].

Well-animated visualizations are expensive and time consuming to produce, and as a result exist only for a small set of topics, almost exclusively in English. A polished four-minute animated explainer requires days of human work, including pedagogical design, scripting, animation code (for example, in Manim [12]), voice recording, editing, and review. The time investment required prevents videos from being produced for most topics, languages, and learner populations.

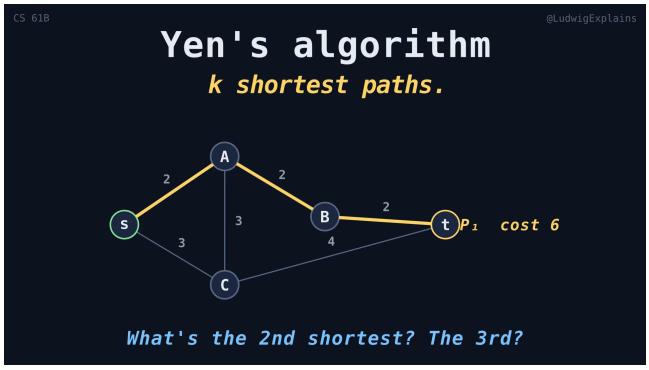

Figure 1: Opening frame of a 4.5-min explainer on Yen's k-shortest-paths algorithm, generated end-to-end by our system, Ludwig. We walk through this video in §4.

Three recent capability shifts change the economics, each of which was missing two years ago. First, frontier large-language models (LLMs) can now autonomously author animation code in a specific library such as Manim when given a style guide. Second, vision models can effectively review sampled frames from a rendered video and catch layout, overflow, and readability defects. Third, multilingual neural text-to-speech, at tens of cents per thousand characters, makes localized narration trivially cheap. Together these make fully autonomous, end-to-end production of engaging educational video both technically and economically feasible at student-relevant quality. They also make multilingual delivery a matter of configuration rather than a separate production effort.

The challenge today, put colloquially, is in ensuring synthesized content goes beyond "AI slop[1]". Convincing video, text, and image generation has come a long way in the past two years, and is now well-sufficient for entertainment purposes [3, 6, 10]. However, in educational content, the nuances in semantic meaning (e.g. what analogy to represent a packet) and the correctness of syntactic presentation (e.g. correctly animating a packet without overlapping text) are materially far more important. Gaps in either can greatly negatively

[1]Slop is defined as "digital content of low quality that is produced usually in quantity by means of artificial intelligence", by Merriam-Webster, who also selected it as the 2025 word of the year [8].

affect learners: syntactic errors make content more difficult to consume, and semantic errors actively mislead learners.

To address these challenges, we propose Ludwig, an agentic system for educational content synthesis with a focus on quality and correctness. Our design standardizes on the Manim library [12] for explicit, deterministic content rendering, and breaks the educational-content synthesis process into stages (described in §3) with frequent review-gates to detect inconsistencies and presentation bugs, maintaining a core split in responsibilities between independent *producer* and *reviewer* roles. To address the important class of semantic bugs, our system splits research and curriculum design into a separate step before any video creation takes place, and is designed with educator-in-the-loop usability as a first-class concern, running in a Discord or Slack space in which Ludwig asks for input from the educator, shares draft content for review, and adjusts content to address feedback. Furthermore, the system learns from educator-provided feedback to better create future content.

We have implemented and run Ludwig with full control over its own YouTube channel[2] for a period of two months. In this time, Ludwig has produced roughly 150 videos spanning computer science, physics, mathematics, and biology in five languages totaling over 45,000 cumulative views, produced at a measured cost of $2 to $10 and about 25 minutes of wall-clock time per explainer video, end-to-end. Our deployment explored both fully-autonomous and educator-in-the-loop variants of content synthesis.

In the rest of this paper, we describe the design philosophy of Ludwig (§2), present the architecture of our prototype system (§3) and a case study of generating a single explainer video (§4), report early metrics on cost and performance (§5), and finally close with discussion on the many remaining open questions (§6). We see this work as an early exploration of how we can use AI to make quality education accessible in a wider range of topics and languages, in order to help educate the next generation of systems thinkers.

2 Core Design Philosophy

Our work's goal, at its core, is to make **quality** educational content more **accessible**. We take quality to mean two kinds of correctness: (1) *semantic* correctness, that the explanation and its analogies are pedagogically sound, and (2) *presentation* correctness, that the rendered frames are legible and defect-free. We take accessibility to mean two things: (1) accessibility in *topic*, enabled by bringing down creation cost and effort to be feasible for topics outside of mainstream most-popular topics, and (2) accessibility in *language*, enabled by automating the re-localization process. The design choices below address each.

[2]The channel can be found at https://www.youtube.com/@ludwigexplains.

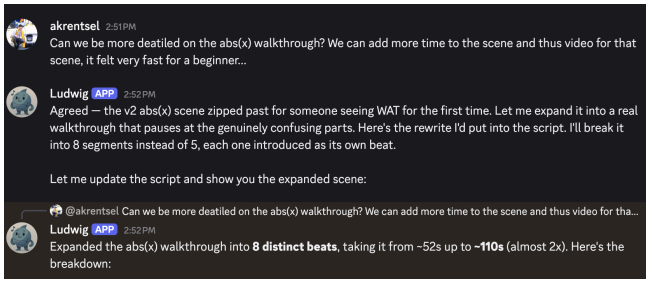

Figure 2: Instructor feedback interface. The instructor requests more detail and slower pacing on a confusing step, which the producer plans, handles, and presents.

Research and curriculum design before production. Before any animation code is written, the producer researches the topic and designs the lesson, pinning down the misconception to correct, the key insight, and the pedagogical arc, and recording correctness notes in a plan. Doing this first ensures Ludwig can devote reasoning attention to semantic correctness directly.

Separate producing from reviewing. A core design philosophy of Ludwig is to separate authoring responsibilities from reviewing responsibilities. This separation ensures there is no conflict of interest in the agent's reasoning. Empirically in the design of Ludwig, we found this to be very important for ensuring quality – when both creation and review was given jointly to the same agent, the resulting created videos contained significantly more mistakes in both syntax and semantics. The same blind spots that produced a defect tend to skip over it on a second read, so an author reviewing its own output is a weak detector.

Topic-independent architectures to generalize. Though we design the system with computer science curriculum in mind, the system architecture intentionally bakes in no assumptions about the concrete topics it will cover. The internal abstractions – scene counts, color palette, pacing, pedagogical framing, etc. – all apply for any topic, and are decided for the given topic. This lets the same pipeline ship a wide range of content without per-topic engineering: a Rayleigh-scattering explainer, an entropy primer, and a key-value-store walkthrough are all produced by a system designed without any of those topics in mind.

First-class support for educator-in-the-loop. Good educational content is heavily tailored to the intended audience's context; selecting what framing to use, how much detail to go into, what past or future topics to connect to, etc. The nuances of the target audience and syllabus context are impossible for the system to assume correctly, therefore Ludwig *must* involve the educator in-the-loop. This allows the educator to provide guidance in natural language without

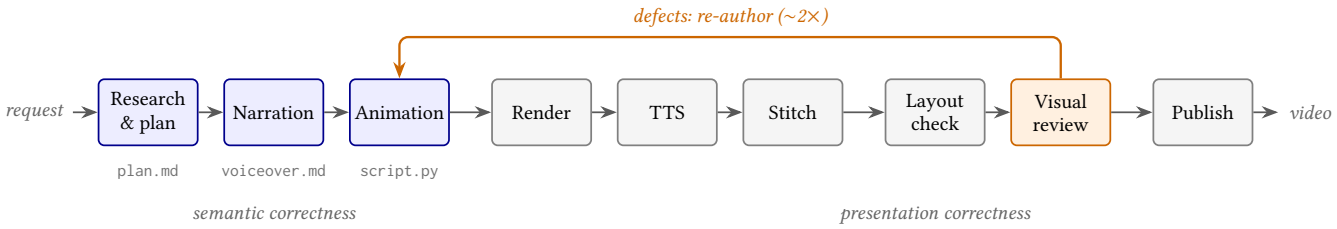

Figure 3: End-to-end pipeline from an educator request to a published video. The producer agent (blue) researches and plans, narrates, and animates; deterministic scripts (gray) render, synthesize narration, stitch, and publish; two review gates guard presentation correctness, a deterministic layout check (gray) and an independent vision reviewer (orange) that returns defects to the animation stage for up to two re-authoring iterations.

touching code while the system updates its internal plan accordingly. We choose Discord or Slack as a natural interface between Ludwig and the educator, where Ludwig requests input, posts drafts for review, and revises in response.

Bringing the educator in-the-loop applies at two scopes. At the single-video scope, it revises a piece of content: Fig. 2 shows an instructor asking for a slower walkthrough of a confusing step; the producer responds by rewriting and expanding the affected scene. At the system scope, feedback is carried forward to future videos: early in our deployment we noted that a short comprehension-check question strengthens an explainer, and once incorporated, such a scene was added to subsequent videos.

Localization from the ground up. Common post-hoc re-localization efforts rely on voice-over dubs to make content accessible beyond the original language. However, this leaves on-screen elements untranslated, and dubbed speech may need to be sped up or slowed down to match scene lengths. Ludwig instead treats re-localization as a first-class concern, re-synthesizing the voiceover, on-screen content, and scene timing in the target language. To enable this, Ludwig keeps all intermediate artifacts for a video, such as the voice-over script and animation code, and translates them into the target language, re-running downstream rendering and stitching.

3 System Architecture

Fig. 3 shows the pipeline end to end. From an initial educator-initiated request, the producer first researches the topic and writes a plan (`plan.md`): it identifies the key ideas and insights for the topic, searches for relevant resources, then fixes the pedagogical arc, scene breakdown, and correctness notes. This research-and-design step precedes any code. The producer then drafts the narration (`voiceover.md`) for each scene and authors the animation as Manim code (`script.py`). Deterministic scripts render each scene to video, synthesize narration audio with neural TTS, and stitch the scenes and audio into a single file. Two review gates guard presentation correctness: a deterministic layout check scans sampled frames for content clipped at the frame edges, and the vision reviewer audits sampled frames against a broader rubric and returns either a pass or a list of defects. Defects are routed back to animation for re-authoring. For ease-of-use, Ludwig also includes a publishing module that holds YouTube credentials and uploads the video and metadata directly.

Prototype. We prototyped Ludwig on top of OpenClaw [9], supported by a handful of skills [2] which aid the system's subagents in using tools for rendering, translation, review, etc. Translation uses the DeepL and Google Translate APIs, and narration uses ElevenLabs and Google Cloud Text-to-Speech for multilingual synthesis [4, 5]. The internal architecture and per-video system logs are available in a repository. Because the animation code the producer writes is reusable, later videos on related topics can build on earlier scene code rather than starting from scratch.

Deployment. Ludwig has run for roughly two months as a public channel, both fully autonomously and with an educator in the loop, producing about 150 educational videos across five languages with over 45,000 cumulative views. Fig. 5 shows the opening frame of one "print vs. return" explainer produced natively in English, Russian, Spanish, and Amharic. The four variants are not subtitled copies: the title, body text, and on-screen captions are regenerated per language, and the layout is re-derived so that the differing text lengths still fit the frame.

4 Case Study: Yen's k Shortest Paths

To make the system's output concrete, we walk through one synthesized video end-to-end. We chose an explainer on Yen's algorithm for the k shortest paths[3], an upper-division algorithms topic appearing in network routing solutions that compute not just the shortest route but a ranked list of alternates in case of link failures. We ran Ludwig in full-autonomous mode from a one-line request ("Produce a long-form Ludwig Explains video (1920×1080, 16:9, 60fps) for the

[3]The video can be found at https://youtu.be/bQCewgMFaYQ.

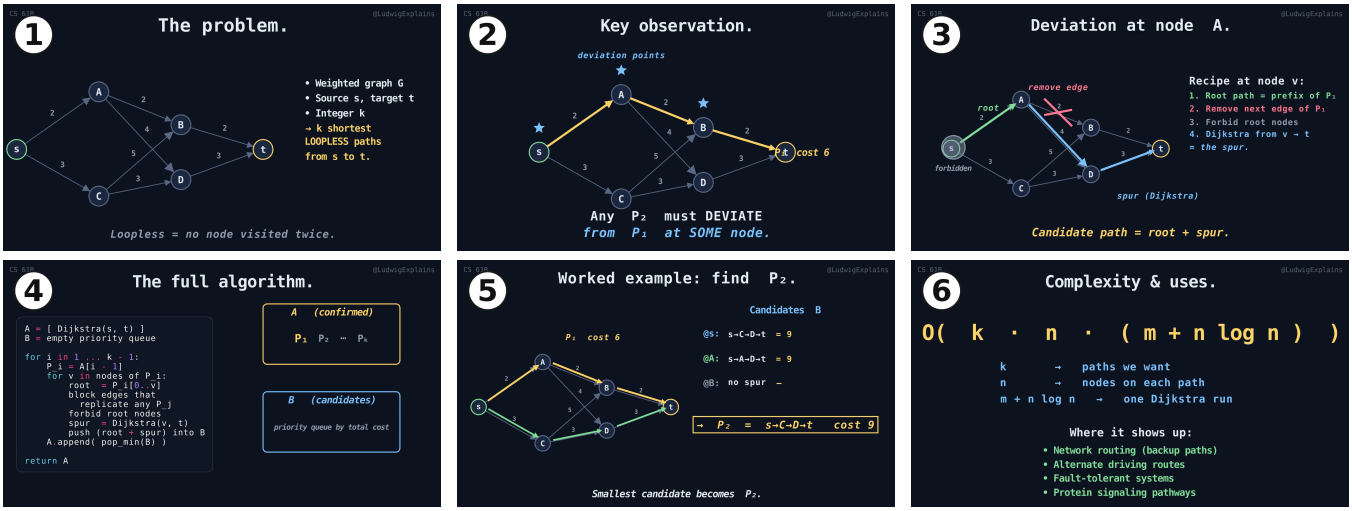

Figure 4: Six of the nine scenes from the autonomously generated Yen's k-shortest-paths explainer (left to right, top to bottom): (1) the formal problem on a small weighted graph; (2) the key observation that each next path must deviate from a known path at some node; (3) the deviation mechanic, with the removed edge (red) and forbidden root nodes (gray) for the spur Dijkstra; (4) the full algorithm, with pseudocode and the confirmed/candidate path sets; (5) a worked trace computing P_2 with a candidate-cost table; and (6) complexity and real-world uses, led by network routing. A single small graph and a consistent color scheme (gold active path, red removed edge, gray forbidden zone) carry across scenes.

CS 61B series on Yen's algorithm for k shortest paths.") allowing it to autonomously resolve any pedagogical questions as they surface, at a measured cost of $8.24 in 31 minutes of wall-clock time.

The narrative the producer planned. Before writing any animation code, the producer iterated on an engaging pedagogical arc for the explainer video, broken into nine distinct scenes. The result opens with a motivating hook (a network route fails and we need plan B), states the formal problem, isolates the single key insight (any next-shortest path must *deviate* from an already-found path at some node), develops the mechanic that exploits it (a "spur" Dijkstra run with a removed edge and a forbidden root), assembles these into the full algorithm with pseudocode, grounds the abstraction in a fully worked example on a small graph, and closes with complexity and real-world uses. Fig. 4 shows six of the nine scenes.

Why these scenes work. The choices reflect Ludwig's deliberate pedagogy decisions rather than a flat recitation of the algorithm. The hook frames the problem in concrete networking terms before any formalism appears. A single small weighted graph is reused across scenes so the viewer never has to re-orient[4], and a consistent color scheme carries meaning: gold for the active path, red for a removed edge, gray for the forbidden zone. The worked example computes the second-shortest path step by step with a running cost table, converting the abstract deviation rule into something a student can follow by hand.

Correctness in practice. From runtime logs, we observe that the system successfully detected and fixed both semantic correctness bugs and syntactic presentation bugs. In the planning stage, Ludwig noticed and pinned down a subtlety where it originally went astray: when generating a deviation, one must forbid the leading edge of *every* previously found path that shares the same root prefix, not just the most recent one, since otherwise the search rediscovers an already-found path. On the presentation side, an early render of the algorithm scene collapsed the pseudocode's indentation; this was caught in review and corrected by re-rendering the scene with a proper code block before publishing. Together, self-reflection in the research step guards the semantics, and the visual review gate guards the presentation.

5 Cost and Performance

We run Ludwig to produce both short-form (≤60 s) and long-form (~4 min) content, and we measure wall-clock time, API cost, and TTS character counts per video. Table 1 reports medians across 13 shorts and 11 long-form videos.

[4]Internal system logs reveal this to be an explicit pedagogical choice made by Ludwig early in the planning stage, before writing concrete scenes: "Use a small graph (5–6 nodes)...so every step is legible."

Figure 5: Opening title slide from a single "print vs. return" explainer localized to English, Russian, Spanish, and Amharic (left to right; each language links to its video). Title, body text, and captions are regenerated per language and the layout is re-derived to fit the differing text lengths, rather than overlaid as subtitles.

Table 1: Measured per-video cost and performance, reported as medians.

Metric	Short (≤60 s)	Long-form (4 min)
Videos measured	13	11
Wall-clock time	22 min	26 min
LLM cost	$2.00	$8.49
TTS characters	575	2,750
TTS cost	$0.17	$0.82
Visual-review # iter	0–2	0–2

A full-length video costs about $9.31 and takes about 26 minutes of wall-clock time, end to end. Cost is dominated by the LLM stages that author the script and animation, not by narration. Crucially, those authoring stages are *reused* across languages: a localized variant only re-translates the narration and on-screen text, re-synthesizes the narration, and re-renders the affected scenes, all cheap steps that avoid re-invoking the LLM. Producing a native variant in an additional language therefore adds little to the total cost of a video. The visual-review loop converges quickly, typically in zero to two iterations.

These figures are roughly two to three orders of magnitude below the time and cost of a human-authored equivalent at comparable quality, where a single polished explainer is days of skilled work. At a few dollars and tens of minutes per video, we believe broad multilingual curriculum generation is now economically viable rather than aspirational.

6 Discussion

This short paper serves as an early report on our experience designing a system to make high-quality explainer videos accessible for more topics and languages. While the preliminary results are promising, much further design and evaluation is required. We discuss our learnings, next steps, and open questions.

Collaborative agentic system design. Though we primarily report the motivation and final system design for Ludwig in this paper, we also learned much in the design process. Unlike traditional systems where design and implementation is distinct from system use, much of Ludwig's system design was done *in collaboration with Ludwig itself*; that is, after creating the initial minimal set of instructions for video creation and wiring up Ludwig to a Discord environment, subsequent design and implementation changes were made in consultation with Ludwig as it was producing videos, through requests made by the human reviewing produced videos over a period of weeks. This represents a fundamental shift in how systems are to be developed – in collaborative consultation with the system itself, rather than independently by a technical human expert – and enables less-technical instructors to customize the system for their own deployment.

A planned deployment with a real language barrier. To both evaluate the system effectiveness and demonstrate its multi-language benefit, we plan to pilot Ludwig in an intensive introductory programming program for students in Ethiopia [1], where the language barrier is a concrete obstacle: strong instructional material exists in English, but many students are more comfortable in Amharic, Oromo, or Somali. We will synthesize optional, targeted videos for topics that past cohorts have struggled with, including print vs. return, for-loops vs. while-loops, function definition vs. calling, and variable scope, in English and three local languages. We note that the cost figures in §5 are what make this practical: producing four-language variants of a dozen targeted videos totals under $200. If allowed by program organizers and the IRB, we hope to report student reviews on the usefulness and clarity of the videos, as well as any observed exam score changes based on primary language spoken compared to previous year offerings.

Benefit to the networking pedagogy community. While Ludwig is designed to be topic-agnostic – we tested topics as different as philosophy, music, and political science to probe its failure modes – our core motivating interest for Ludwig is systems. The author is a member of the networked systems community. In our testing, we applied Ludwig to a dozen networking research papers and found its explanation

to be of high-quality, per feedback from the papers' own authors. The capability we most want to sharpen is producing visual explanations that build intuition for the large, complex systems taught in upper-division courses, such as internet architecture, congestion control, and operating-system internals, where an animation can convey intuition that static slides and lectures struggle to.

Concept understanding vs. skill application. An open question is whether these videos close the concept understanding gap without leaving a skill-application gap. A well-animated explainer can make a concept click, but understanding a concept is not the same as being able to apply it under exam or programming conditions. The natural next step is to pair each video with targeted practice drawn from the course materials students are actually assessed on, and to ask whether the producer can also generate or sequence that practice. As an early attempt, in working with Ludwig in the educator-in-the-loop Discord, we discussed this with the system and it successfully began incorporating a self-understanding quick quiz question into its series of computer science videos. We leave a full, controlled study of this question to future work.

Augmenting, not replacing educators. Ludwig's goal is to automate much of the tedium in video creation in order to support educators in producing high-quality content that is widely accessible, rather than to replace the educators. Indeed, the educator must guide Ludwig in appropriately adjusting its content and presentation for the context of the target audience. We see this generated content as supplementing rather than replacing human-written curriculum.

Correctness, and the limits of automated review. As we discussed in §1, a core challenge of educational content is that correctness and precision are of paramount concern. Despite careful design, Ludwig's semantic correctness rests on the upfront research and planning steps, and a subtly misleading analogy, or a claim that is plausible but wrong, can still pass automated review and reach a learner. This is the failure mode we worry about most, because a presentation defect merely annoys whereas a semantic one actively misleads. For now, the educator-in-the-loop is our final and strongest trust mechanism. We hope to expand on techniques to support the educator-in-the-loop's review in the future.

These are early efforts in AI-supported pedagogy. We are excited to see how such systems continue to improve, both as core LLM capabilities continue to improve, and as the surrounding community learns to better harness their existing capabilities.

Acknowledgments

We thank our pedagogy-oriented colleagues at UC Berkeley for valuable discussions that shaped this work, including Lisa Yan, Josh Hug, Sylvia Ratnasamy, Jelani Nelson, and Tess Despres. We thank Thomas Kidane, Akotet Tesema, and Margarita Geleta for their help in reviewing localization translation quality.

References

[1] AddisCoder. 2026. AddisCoder: A Free Intensive Summer Program in Computer Science and Algorithms for Ethiopian High School Students. https://www.addiscoder.com/. Accessed 2026-06-08.

[2] Agent Skills. 2026. Agent Skills: An Open Format for Extending AI Agent Capabilities. https://agentskills.io/. Accessed 2026-06-08.

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, et al. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[4] DeepL. 2025. DeepL API. https://www.deepl.com/. Accessed 2026-06-01.

[5] ElevenLabs. 2025. ElevenLabs Text to Speech. https://elevenlabs.io/. Accessed 2026-06-01.

[6] Dan Kondratyuk, Lijun Yu, Xiuye Gu, José Lezama, Jonathan Huang, Grant Schindler, Rachel Hornung, Vighnesh Birodkar, Jimmy Yan, Ming-Chang Chiu, Krishna Somandepalli, Hassan Akbari, Yair Alon, Yong Cheng, Josh Dillon, Agrim Gupta, Meera Hahn, Anja Hauth, David Hendon, Alonso Martinez, David Minnen, Mikhail Sirotenko, Kihyuk Sohn, Xuan Yang, Hartwig Adam, Ming-Hsuan Yang, Irfan Essa, Huisheng Wang, David A. Ross, Bryan Seybold, and Lu Jiang. 2024. VideoPoet: A Large Language Model for Zero-Shot Video Generation. arXiv:2312.14125 [cs.CV] https://arxiv.org/abs/2312.14125

[7] Richard E. Mayer. 2009. *Multimedia Learning* (2nd ed.). Cambridge University Press.

[8] Merriam-Webster. 2025. Word of the Year 2025: Slop. https://www.merriam-webster.com/wordplay/word-of-the-year. Accessed 2026-06-08.

[9] OpenClaw Contributors. 2026. OpenClaw: A Self-Hosted, Open-Source AI Agent Framework. https://github.com/openclaw/openclaw. Accessed 2026-06-08.

[10] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-Resolution Image Synthesis with Latent Diffusion Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[11] Grant Sanderson. 2025. 3Blue1Brown. https://www.3blue1brown.com/. Accessed 2026-06-01.

[12] The Manim Community Developers. 2025. Manim – Mathematical Animation Framework. https://www.manim.community/. Accessed 2026-06-01.