TIMELINEBENCH: EVALUATING VIDEO-EDITING AGENTS ON EDITABLE PROJECT STATES

Anonymous authors
Paper under double-blind review

ABSTRACT

In professional video editing, an agent's output is not a rendered video but an editable project that human editors continue to work on. Thus, rigorous evaluation of video-editing agents requires both scalable construction of realistic editing tasks and systematic verification of such project states. We introduce **TimelineBench**, a benchmark of 781 video-editing tasks that connects task construction and outcome evaluation through a shared structured specification. TimelineBench extracts real workflows from official editing tutorials and organizes them into 48 atomic editing operations and 121 reusable task templates. Each atomic operation specifies both the edit to perform and evaluation rules for the requested changes and required preservation, allowing natural-language instructions and their evaluation to come from the same task specification. TimelineBench evaluates agents directly on the resulting project state, verifying request completion and state preservation, without requiring a reference GUI trajectory or a single gold project. We build OpenCut-TL to run the full benchmark and evaluate seven computer-use agents, whose task success rates range from 3.5% to 87.2%, with preservation failures remaining a major source of error. By unifying realistic task construction with systematic outcome evaluation, TimelineBench provides a scalable and reproducible foundation for evaluating video-editing agents.

1 INTRODUCTION

Professional video editing is not merely about producing a rendered video, but about creating an editable project that human editors can continue to refine. Evaluating a video-editing agent therefore requires judging not only the rendered video, but also whether the underlying project reflects the intended edits. Prior video-editing research has largely followed two directions. Generative video editing transforms visual content based on natural-language instructions and primarily evaluates the perceptual quality and instruction adherence of rendered videos (Wu et al., 2026; Gao et al., 2026; Chen et al., 2026). Meanwhile, a small but growing body of work benchmarks computer-use agents that operate the GUIs of real video-editing software (Hu et al., 2026). This direction reflects a broader shift in creative-tool research from flattened outputs toward editable artifacts such as PPTX and SVG (Tang et al., 2025; Liang et al., 2025; Xing et al., 2026).

Benchmarking video-editing agents requires both realistic editing tasks and reliable evaluation of the resulting project state. However, existing approaches face limitations on both fronts. For task construction, there is a trade-off between realism and scalability: expert-crafted tasks and demonstrations reflect realistic editing workflows but are costly to scale, whereas LLM-synthesized tasks scale easily but may yield implausible or incoherent editing requests. For evaluation, existing benchmarks commonly rely on rendered videos or editor screenshots. These visual outputs are only partial observations of the underlying project state and cannot systematically verify which objects were modified, which parameter values were applied, or whether unrelated parts of the project were preserved. In summary, a unified approach that supports both scalable construction of realistic video-editing tasks and systematic evaluation of their outcomes remains lacking.

To tackle these challenges, we introduce **TimelineBench**, a benchmark suite for video-editing agents containing 781 tasks derived from 121 templates (Figure 1). TimelineBench addresses both components through *tutorial-grounded task construction* and *predicate-based project-state evaluation*. Official editing tutorials provide workflows, ordered procedures that accomplish a concrete editing

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.

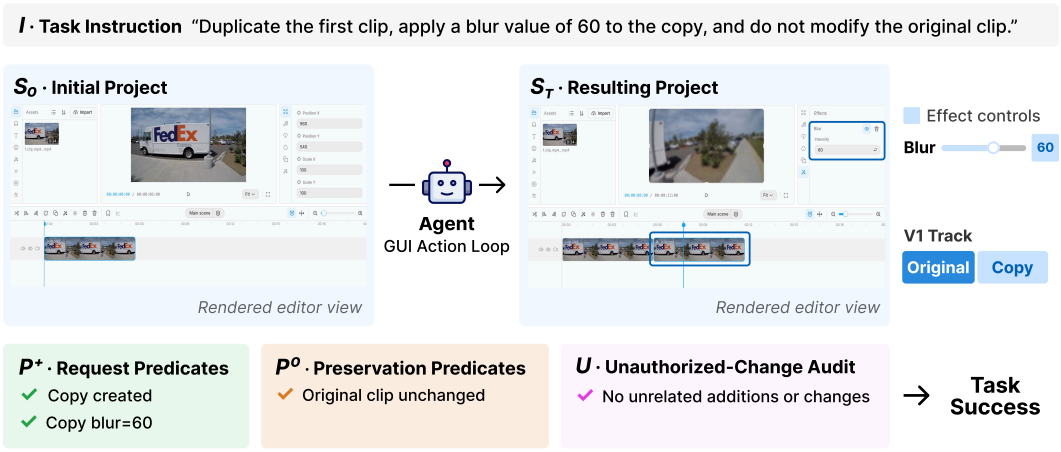

Figure 1: An episode of a running example of a TimelineBench task. A computer-use agent edits the initial project S_0 through GUI actions, producing the final project S_T. The screenshots are rendered editor views; the evaluator compares the structured project states S_0 and S_T using Boolean predicates. The request predicates P^+ check the requested edits, the preservation predicates P^0 check that the original clip is unchanged, and the unauthorized-change audit U checks for changes that no predicate authorizes. The episode succeeds only if all three checks pass.

goal, which serve as our source of realism. We decompose these workflows into atomic operations such as duplicating a clip or setting an effect parameter, and recompose them into reusable task templates. Binding each template to different media, targets, and parameters constructs diverse, realistic task instances without per-task expert effort. A human validation study on 200 tasks confirms that the generated tasks are meaningful (96.2%) and executable (92.8%). For evaluation, each atomic operation provides predicate templates for the requested changes and required preservation. Instantiating the instruction and these predicates from the same task specification enables systematic, rule-based evaluation of the resulting project state.

To run our benchmark at scale, we build OpenCut-TL, an instrumented extension of the open-source OpenCut Classic editor, and verify application fidelity in Adobe Premiere Pro. TimelineBench clearly separates seven computer-use agents by capability: task success spans from 87.2% for the strongest proprietary model to below 5% for both open-weight agents. Substantial headroom remains even for the strongest model: GPT-6 Sol fails on a notable share of animation and timeline-editing tasks, and its success drops from 83.9% to 62.5% when the same tasks are run in Premiere Pro. Altogether, TimelineBench provides a rigorous and scalable foundation for evaluating the video-editing capabilities of computer-use agents and tracking future progress.

2 Related Work

Video editing and generation. Most recent video-editing work modifies the rendered video directly. Diffusion-based and instruction-guided methods are typically evaluated on instruction following, visual quality, and temporal consistency (Sun et al., 2024; Bai et al., 2026), while CoVEBench, VEFX-Bench, and IVEBench similarly score edits mainly from output videos (Wu et al., 2026; Gao et al., 2026; Chen et al., 2026). LAVE (Wang et al., 2024) and EditDuet (Sandoval-Castañeda et al., 2025) instead operate on editing workflows, but are evaluated through user studies and output comparisons rather than as reusable benchmarks.

GUI agents in professional software. WebArena, AndroidWorld, and OSWorld provide evaluation on the web, mobile devices, and desktop (Zhou et al., 2024; Rawles et al., 2025; Xie et al., 2024). Recent benchmarks extend this setting to professional software. VideoGUI derives tasks from instructional videos and human reproduction (Lin et al., 2024), while ProSoftArena curates tasks across 13 professional applications (Ai et al., 2026). CutVerse, the closest to our setting,

collects expert demonstrations of post-production workflows across seven applications and derives evaluation milestones from them (Hu et al., 2026). Such human-grounded task construction provides realism but is costly to scale. TimelineBench instead compositionally instantiates tutorial-grounded editing operations, jointly constructing the initial project, instruction, and evaluation conditions.

Execution- and state-based evaluation. SWE-bench evaluates model patches with executable tests (Jimenez et al., 2024), while AppWorld generates tasks with initial states and programmatic evaluations that accept different valid solutions while detecting unintended changes (Trivedi et al., 2024). Similar state-based evaluation has been applied to editable artifacts: PPTArena uses structural and visual evidence for PowerPoint edits (Ofengenden et al., 2025), and Interactive Reward Agent verifies completion from files and application state (Shi et al., 2026). This approach is particularly important for video editing because rendered outputs do not uniquely determine the underlying project; splitting a clip, for example, may leave the rendered video unchanged. TimelineBench therefore evaluates predicates over initial and final project states, verifying requested edits and preservation without requiring an exact reference project.

3 Benchmark Formulation and Environment

3.1 Formulation

Task Definition. We formulate our task as a tuple $\mathcal{T} = (S_0, I, P^+, P^0)$. The initial project state S_0 is already prepared for the task, with the application and sequence open and the required media and timeline set up, reflecting workflows that continue from a human editor's timeline. The instruction I is a natural-language editing request. The request predicates P^+ and preservation predicates P^0 are Boolean functions of S_0 and the final project state S_T: those in P^+ check whether the requested changes have occurred, and those in P^0 check whether the state that must remain unchanged has been preserved. In the running example (Figure 1), P^+ checks whether a copy has been created and whether its blur value is 60, while P^0 checks whether the original `1.clip` remains unmodified. Predicates are hidden from the agent and are only revealed to the evaluator.

Atomic Operations. The edit requested by each TimelineBench task decomposes into an ordered sequence of one or more *atomic operations*, the smallest changes to the project state that can be requested on their own (e.g., `set_clip_effect` or `apply_video_transition`). Atomicity is defined by the granularity of changes to the project state, not by the granularity of GUI actions. A single operation may be completed through different action sequences. In Premiere Pro, for example, `split_clip` can be performed with the `Ctrl+K` shortcut or with the Razor tool. TimelineBench defines a set $\mathcal{O}$ of 48 atomic operations (Appendix A.1). Each operation $o \in \mathcal{O}$ specifies a precondition C_o on the initial project and predicate templates $\widehat{P}_o^+$ and $\widehat{P}_o^0$, i.e., predicates whose target and value are left as parameters. For example, as a precondition, `set_clip_effect`(x, e, v) requires that clip x exists. Its request template checks that effect e on x has value v, and its preservation template checks that the timeline and source intervals of x are unchanged. The operations are not part of the task tuple and are never shown to the agent; instead, they serve as building blocks from which I, P^+, and P^0 are instantiated (§4). Because each operation carries its own predicate templates, any valid composition of operations comes with evaluation conditions by construction, making task generation and evaluation coherent.

3.2 Episode Protocol

At each time step, the agent receives the instruction I, the current screenshot, its interaction history, and the remaining action budget, together with a fixed system prompt (see Appendix C.2). The history records the agent's previous responses, the GUI actions it executed, and their execution results. It observes neither the DOM nor the internal project state. The agent responds with low-level GUI actions such as `click`, `drag`, `type`, and `keypress`; the concrete action format follows each agent's native interface. A single response may contain several actions, which the environment executes in order before returning a new screenshot. The episode ends when the agent returns a completion response or exhausts its action budget, after which the resulting project state S_T is evaluated.

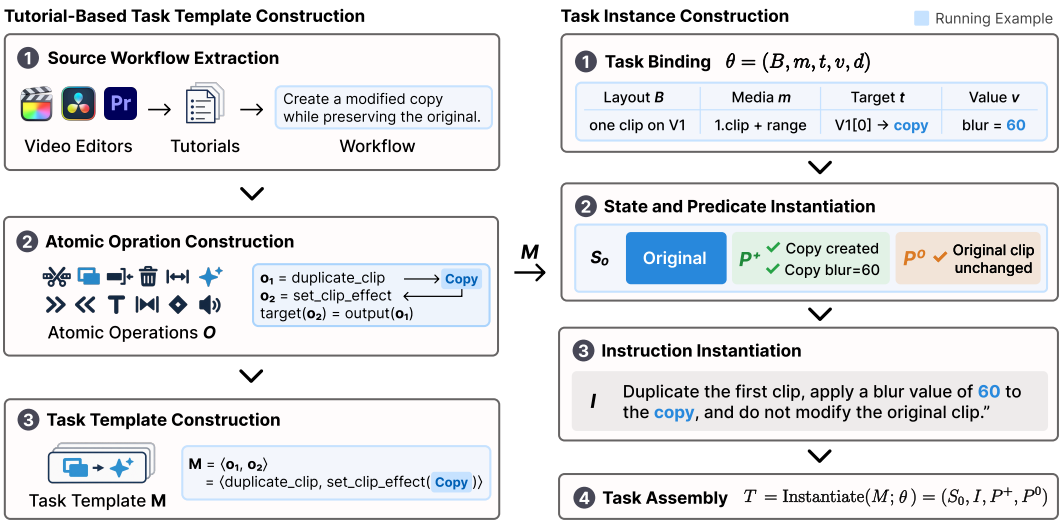

Figure 2: The TimelineBench task-generation pipeline. Tutorial workflows are abstracted into task templates (§4.1), which are bound to instantiate the initial project, instruction, and predicates (§4.2). Skyblue insets show the running example.

3.3 Controllable Execution Environment

We use two environments to run TimelineBench: (1) OpenCut-TL as the primary execution environment for large-scale parallel experiments and (2) Adobe Premiere Pro as the target environment for verifying application fidelity in realistic professional video editing.

OpenCut-TL. TimelineBench requires a scalable, web-based video-editing environment for parallel agent testing. We implemented OpenCut-TL, our benchmark environment built as an instrumented fork of the MIT-licensed OpenCut Classic editor (OpenCut Project). When an episode ends, it automatically exports the current project state in JSON format. Out of the 48 operations defined in TimelineBench, 19 were newly implemented in OpenCut-TL, while 29 were already provided through existing OpenCut features. See Appendix A.2 for the complete operation list and operation-to-interface mapping. To verify the accuracy and stability of the 19 newly implemented operations, we configured 178 native and web tests, including time and source-range calculations, preservation of unrelated objects, rejection of incorrect edits, undo/redo, and JSON save and restore. All 178 tests passed on the final run.

Adobe Premiere Pro. To verify application fidelity, we also conducted experiments on a 56-task subset in Adobe Premiere Pro, a widely used professional video-editing application. §6 provides the detailed settings. When an episode ends, an evaluator-side UXP extension exports the current project state in Final Cut Pro XML format.

4 TimelineBench Task Generation Pipeline

Even if the individual operations are feasible, a random combination is not guaranteed to form a plausible editing request. (e.g. *"Delete the clip and crop the bottom of the clip."*) TimelineBench tasks stem from *official editing tutorials*, which pair concrete editing goals with ordered procedures for achieving them. Each procedure thus supplies a realistic and coherent workflow, rather than an arbitrary combination of operations. We abstract these workflows into reusable task templates and instantiate many tasks from each, so that the benchmark scales through binding rather than per-task expert effort. Construction proceeds in two phases (Figure 2). We first build atomic operations and reusable task templates from editing tutorials (§4.1), and bind each template to compatible media, targets, and parameters to create concrete task instances (§4.2). After the generation is completed, we assess the quality of the tasks through human validation (§4.3).

4.1 TUTORIAL-BASED TASK TEMPLATE CONSTRUCTION

We use task templates as the foundation for generating multiple instances. A task template abstracts a single editing request as the required sequence of operations. A concrete task instance is created after binding the video, target clip, and parameter values.

Task template construction proceeds by (1) extracting source workflows from official editing tutorials, (2) organizing these workflows into common atomic operations, and (3) composing the operations into a task template.

Source Workflow Extraction. A workflow is an ordered sequence of editing operations, together with their target relationships, that fulfills a coherent editing request. For example, duplicating a clip and applying an effect to its copy forms a single workflow: *creating a modified copy while preserving the original.* We collected 45 official tutorial documents for three representative video-editing applications, Adobe Premiere Pro, Final Cut Pro, and DaVinci Resolve, and extracted 165 source workflows from them.

Atomic Operation Set Construction. We derive the operation set $\mathcal{O}$ from 165 source workflows, merging equivalent functions across the three editors into 48 atomic operations that cover timeline editing, layout and effects, playback speed, text, transitions, keyframes, and audio. See Appendix A.1 for the complete operation set $\mathcal{O}$. Operations that were unsupported by OpenCut Classic, or whose existing behavior differed from the required semantics, were newly implemented or modified in OpenCut-TL. §3.3 describes the OpenCut-TL implementation and verification methods.

Task Template Set Construction. A task template M is formulated as an ordered operation sequence $M = \langle o_1, o_2, \ldots, o_K \rangle$, where $o_j \in \mathcal{O}$. A total of 121 task templates were created: 95 were constructed from contiguous subworkflows of source workflows after verifying their executability, and 26 were manually crafted from the tutorials to supplement atomic-operation coverage. See Appendix A.3 for the complete template set.

4.2 TASK INSTANCE CONSTRUCTION

Task Binding. A task template M specifies only the order of operations, not the media, targets, or parameters they act on. To create a concrete task instance, we should assign these elements through what we define as task binding $\theta = (B, m, t, v, d)$. Thus, binding the same template in different ways therefore yields distinct tasks that share a plausible editing goal.

Task binding consists of a base timeline layout B, source media and source range m, targets t and parameter values v for the template's operations, and distractor media d that is present in the project but not needed for the task. The base layout is selected from eight reusable timeline structures that vary in the number of clips, the presence of a text object, and resolution (see Appendix A.4). A target may refer either to an object in the initial project or to an object produced by an earlier operation in the template. The distractor media are imported into the project but not needed for the task, so that the agent must identify the target rather than use whatever is available.

In the running example (Figure 1 and 2), the template that duplicates a clip and applies an effect to the copy is bound to a one-clip layout, the first clip on V1 as the duplication target, the resulting copy as the effect target with a blur value of 60, and a distractor video imported into the source panel but not placed on the timeline.

Not every binding suits every template. The requirements of M follow from the preconditions C_o from its atomic operations. For example, a template containing a transition operation and a text editing operation requires two adjacent clips and an existing text object. For each template M, we therefore enumerate the bindings that satisfy these requirements and select k of them, $\{\theta_i\}_{i=1}^{k}$, using a fixed seed.

State and Predicate Instantiation. For each selected binding θ_i, we construct the initial project state $S_{0,i}$ by placing the source media and range m_i into the base layout B_i and importing the distractor media d_i. We then instantiate the predicates by filling each operation's predicate templates

with the targets and parameter values in θ_i and taking the union over the operations of M:

$$P_i^+ = \bigcup_{j=1}^{K} \widehat{P}_{o_j}^+[\theta_i], \qquad P_i^0 = \bigcup_{j=1}^{K} \widehat{P}_{o_j}^0[\theta_i], \tag{1}$$

where $\widehat{P}_{o_j}[\theta_i]$ denotes the template of o_j with its parameters bound by θ_i. When a target is produced by an earlier operation, such as the copy created by a duplication, the predicate refers to that produced object, which the evaluator locates in the final state.

Instruction Instantiation. Once $S_{0,i}$ and the target and parameter assignments t_i and v_i are fixed, a deterministic compiler produces a complete concrete instruction. An LLM author then revises it into the final instruction I_i for concision without changing the bound task specification. A separate LLM critic audits the revision against the original instruction, operation program, and predicate sets P_i^+ and P_i^0 before acceptance. See Appendix B for the prompts, model, and acceptance procedure.

Task Assembly. Together, the procedures above form an executable benchmark task $\mathcal{T}_i = \mathrm{Instantiate}(M; \theta_i) = (S_{0,i}, I_i, P_i^+, P_i^0)$.

We automatically validate the task instance through a reference program derived from the task binding. This reference program is used only to verify task executability and evaluator consistency; it is neither exposed to the agent nor treated as a required GUI trajectory.

We instantiate 10 tasks from each of the 21 manually selected templates focused on timeline operations, and 5–10 tasks from each of the remaining 100 templates. After the automatic validation, the pipeline yields the final set of 781 TimelineBench tasks, spanning 48 atomic operations and 121 task templates. We evaluate the quality of the generated tasks through the human validation study described in §4.3.

4.3 Human Validation of Generated Tasks

We assess generated tasks through human annotation on meaningfulness, self-containedness/executability, and language quality. These criteria ask whether the request is plausible for a video-editing user, whether the required edit is determined by the instruction and initial state, and whether the instruction is clear and natural, respectively.

We evaluate 200 tasks with 10 annotators, with 60 tasks per annotator and three independent annotators per task. Annotators provide a binary Yes/No judgment for each criterion. Positive response rates are 96.2% for meaningfulness, 92.8% for self-containedness/executability, and 88.8% for language quality. Pairwise agreement is 92.3%, 88.0%, and 81.3% for the three criteria, respectively. Gwet's AC1 (Gwet, 2008), an inter-annotator agreement measure, is 0.917, 0.862, and 0.767, respectively.

5 Predicate-based project-state evaluation

The TimelineBench evaluator judges an episode by comparing the structured project states before and after editing, rather than the agent's GUI trajectory or visual outputs such as rendered videos and screenshots, which are only partial observations of the project. In particular, because the predicates are instantiated from the same template and binding as the instruction, evaluation conditions come from the task itself. This alignment makes evaluation systematic, scoring every episode by the same deterministic rules over the same project-state representation.

5.1 Evaluation Formalization

Project State Representation. When an episode begins and ends, we export the project as OpenCut-TL JSON or Premiere Pro XML and parse it to obtain the initial state S_0 and the final state S_T. The parser extracts sequence settings, tracks, clips, text, transitions, media references, timeline positions, source timing, effects, and parameters, and converts temporal quantities to a common frame-based representation. Both OpenCut-TL and Premiere Pro exports are converted into a common canonical representation and evaluated using the same predicate checks.

Object Identification. An *object* is an individually identifiable editable element. Video clips, audio clips, and text graphics are representative objects, while timing, effects, and parameters are expressed as attributes of these objects. We match objects across S_0 and S_T by their object IDs in OpenCut-TL. Since IDs are not reliably preserved in XML exports, we additionally use track, media-reference, and timing information in Premiere Pro. Split, duplicate, and insert operations create objects that exist only in S_T. For such targets, the evaluator identifies the new object produced by the corresponding operation, using its source media and expected interval, and then evaluates the predicates on that object.

5.2 Success Criterion

Predicate Checks. Each predicate in P^+ and P^0 is evaluated on (S_0, S_T) after its target is resolved. Numeric conditions, such as frame positions and effect parameters, are compared within explicit tolerances, whereas categorical values such as transition types and text content must match exactly. In the running example, the evaluator compares S_0 with S_T to identify the newly created copy. P^+ checks whether the blur value is 60, P^0 checks whether the original clip was preserved.

Unauthorized-Change Audit. Preservation predicates protect only the state that a task explicitly declares, so the audit U checks the rest of the project. It compares the full set of changes between S_0 and S_T, denoted $\Delta(S_0, S_T)$, with the changes authorized by the task's predicates, $A(P^+, P^0)$, and passes only if no unauthorized change remains:

$$U(S_0, S_T; P^+, P^0) = \mathbf{1}\big[\Delta(S_0, S_T) \setminus A(P^+, P^0) = \varnothing\big]. \tag{2}$$

In the running example, adding a text graphic fails U even if all predicates hold.

5.3 Metrics

Success Rate. An episode succeeds only if all three checks pass.

$$\mathrm{Success}(\mathcal{T}, S_T) = \Big(\bigwedge\nolimits_{p\in P^+} p(S_0, S_T)\Big) \wedge \Big(\bigwedge\nolimits_{q\in P^0} q(S_0, S_T)\Big) \wedge U(S_0, S_T; P^+, P^0). \tag{3}$$

This criterion leaves the editing route to the agent but is strict about the outcome: any sequence of GUI actions is accepted if the final project satisfies the request, preserves the protected state, and contains no unauthorized change.

Partial Progress. We compute request coverage and preservation coverage for every task, regardless of success. These metrics measure the fraction of requested conditions and preservation conditions satisfied in the final state.

$$\mathrm{Cov}^+ = \frac{1}{|P^+|}\sum_{p\in P^+} \mathbf{1}[p(S_0, S_T)], \qquad \mathrm{Cov}^0 = \begin{cases} \frac{1}{|P^0|}\sum_{q\in P^0} \mathbf{1}[q(S_0, S_T)], & |P^0| > 0, \\ 1, & |P^0| = 0. \end{cases} \tag{4}$$

6 Experiments

6.1 Experimental Setup

We evaluate seven models on all 781 TimelineBench tasks in OpenCut-TL. The five proprietary models are GPT-6 Sol, GPT-5.6 Sol, GPT-6 Luna, GPT-5.6 Luna, and GPT-5.4 mini (OpenAI, b). The two open-weight GUI agents are UI-TARS-1.5-7B (Qin et al., 2025) and OpenCUA-7B (Wang et al., 2025).

We run all models through the TimelineBench evaluation harness, which executes GUI actions and captures screenshots. GPT models use OpenAI's native computer tool (OpenAI, a), while UI-TARS and OpenCUA use their model-specific prompting and action formats. Agents receive the task instruction, screenshots, and interaction history, with a budget of 80 executed GUI actions per episode. Upon termination, we evaluate the final project state using success rate (SR), request coverage, and preservation coverage as defined in §5. Coverage is macro-averaged over tasks. Appendix C provides model-specific inference settings and history configurations.

Table 1: Performance of seven models on 781 TimelineBench tasks in OpenCut-TL. All values are percentages. Bold indicates the best value within each model group. Request and preservation coverage are macro-averaged over tasks.

Model	SR (%)	Request coverage (%)	Preservation coverage (%)
Proprietary models			
GPT-6 Sol	**87.20**	**91.11**	**99.82**
GPT-5.6 Sol	84.25	89.38	99.52
GPT-6 Luna	71.45	77.98	99.25
GPT-5.6 Luna	69.01	75.85	98.91
GPT-5.4 mini	47.63	59.74	96.94
Open-weight GUI agents			
UI-TARS-1.5-7B	**4.10**	12.24	**85.70**
OpenCUA-7B	3.46	**14.14**	85.50

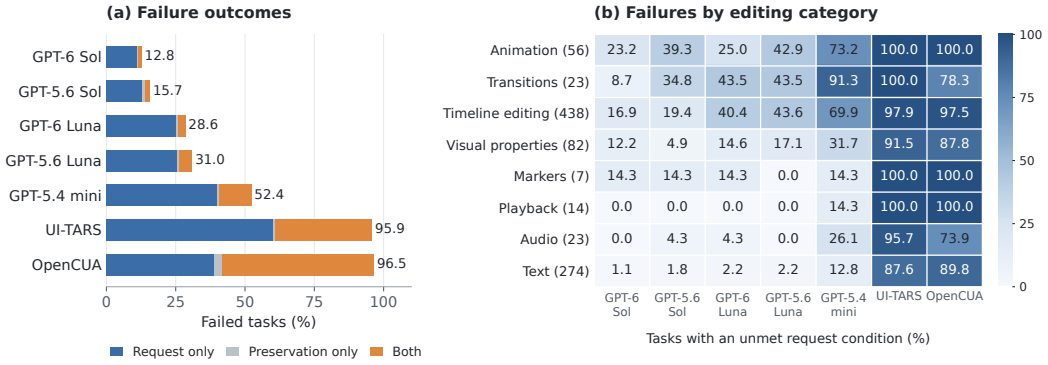

Figure 3: Failure analysis. (a) Failure outcomes as percentages of all 781 tasks. (b) Request-condition failure rates within each editing category, ordered by decreasing mean failure rate across the seven models. Each category includes tasks with at least one corresponding request condition, with task counts in parentheses. Tasks may belong to multiple categories.

6.2 Main Results

GPT-6 Sol achieves the highest success rate at 87.20%, followed by GPT-5.6 Sol at 84.25% (Table 1). Both Luna models outperform GPT-5.4 mini, while the two open-weight agents complete fewer than 5% of tasks. OpenCUA has higher request coverage but lower success than UI-TARS, showing that partial completion does not necessarily yield task success. Preservation coverage is high for proprietary models (96.94–99.82%) but lower for UI-TARS (85.70%) and OpenCUA (85.50%).

6.3 Failure Analysis

Analysis procedure. We analyze failures at two levels: episode outcomes and individual editing conditions. Using the evaluator's checks on the initial and final project states, we distinguish failures involving unmet requests, preservation violations, or both. For these outcome groups, preservation violations include changes outside the permitted edits. We group request conditions into editing categories, such as timeline editing, text, and animation. A task belongs to a category if it contains at least one corresponding condition. For each category, we report the percentage of tasks with at least one unmet condition in that category. Tasks may belong to multiple categories. Appendix C.6 defines each editing category and details the conditions it includes.

Failure outcomes. Request-only failures are the largest group for all proprietary models, accounting for 89 of GPT-6 Sol's 100 failed episodes (Figure 3(a)). For OpenCUA, 426 of its 754 failed episodes involve both unmet requests and preservation violations, including unauthorized changes.

Editing categories. Figure 3(b) shows that both Sol models leave text conditions unmet relatively rarely, but encounter more failures in timeline editing and animation. GPT-6 Sol's failure rate is 1.09% for text, 16.89% for timeline editing, and 23.21% for animation. GPT-5.6 Sol shows the same ordering at 1.82%, 19.41%, and 39.29%, respectively. Timeline editing includes changes to clip timing, arrangement, and source selection, while animation requires specified parameter changes over time. The two open-weight agents show failures across all eight categories, with timeline-editing failure rates above 97% and unmet animation conditions in all 56 tasks in that category.

Case studies. In one OpenCut-TL episode, UI-TARS is asked to set a clip's opacity to 60%. It repeats the same click action 48 times within an 80-action episode, yet the final opacity remains at 78%. All four preservation conditions are satisfied. In another, GPT-6 Sol applies the requested 10% left crop and attempts to adjust scale and position. It then declares completion, although the final state fails the framing condition. Supplementary analyses of action-budget exhaustion and task level appear in Appendix C.6 and Appendix C.5, respectively.

6.4 Cross-Editor Evaluation

We evaluate GPT-6 Sol and Luna on the same 56 tasks in OpenCut-TL and Adobe Premiere Pro (Table 2). Sol achieves 83.93% success in OpenCut-TL and 62.50% in Premiere, while Luna achieves 58.93% and 48.21%, respectively. Both models perform better in OpenCut-TL, with Sol outperforming Luna in both editors. The editors serve complementary purposes: OpenCut-TL supports controlled, large-scale evaluation through structured project-state checks, while Premiere extends the evaluation to a professional desktop editor. Appendix C.4 provides coverage results and execution details.

Table 2: Success rates (%) on the same 56 tasks in both editors.

Model	OpenCut-TL	Premiere
GPT-6 Sol	83.93	62.50
GPT-6 Luna	58.93	48.21

6.5 Human–Agent Comparison

We recruited 10 graduate students with little or no prior video-editing experience and assigned each of the 100 tasks to two participants. Participants succeeded in 129 of 200 assignments (64.50%). On the same task set, both Sol and both Luna models achieve higher success rates than the human participants, while GPT-5.4 mini and the two open-weight agents score lower (Table 3). GPT-6 Sol achieves 92.00%, exceeding the human success rate by 27.50 percentage points on this task set. Scoring details and supplementary human coverage are provided in Appendix C.3.

Table 3: Success rates on the same 100 tasks, with two participant assignments per task for humans.

Participant / model	SR (%)
Human	64.50
GPT-6 Sol	92.00
GPT-5.6 Sol	88.00
GPT-6 Luna	76.00
GPT-5.6 Luna	75.00
GPT-5.4 mini	57.00
UI-TARS-1.5-7B	9.00
OpenCUA-7B	6.00

7 Conclusion

We introduced TimelineBench, a benchmark for evaluating video-editing agents through the structured state of editable projects. TimelineBench contains 781 tasks instantiated from 121 tutorial-grounded templates spanning 48 atomic editing operations. Its predicate-based evaluator verifies requested changes, required preservation, and unauthorized changes without requiring agents to follow a reference GUI trajectory or reproduce a single gold project. Experiments with seven computer-use agents reveal substantial differences in video-editing capability, with task success ranging from 3.5% to 87.2%. The strongest model surpasses novice human editors on a 100-task subset (92.0% vs. 64.5%), whereas the evaluated open-weight agents remain below 5%. Large-scale execution in OpenCut-TL demonstrates the scalability of the benchmark, while evaluation on a matched subset in Adobe Premiere Pro examines its applicability to a professional editing environment. Together, these results show that structured project-state evaluation can support rigorous, systematic, and scalable assessment of video-editing agents while preserving the editability that distinguishes professional creative workflows from flattened-output generation.

AI USE STATEMENT

In this work, we used generative AI tools for assisting with the generation of a synthetic benchmark dataset, providing feedback on the research methodology and experimental design, implementing methods, translation, cleaning and reformatting dataset, and interpreting results. We have not used generative AI tools for developing the conceptual framework, and formulating mathematical claims, providing critical ingredients for proving mathematical claims, developing or writing proofs, proposing or refining hypotheses, and qualitative or thematic data analysis are not applicable to this work. Additionally, we used generative AI tools to create and edit software code, search for and summarize relevant literature, prepare scientific figures and research artifacts, and revise the manuscript. We have reviewed all AI-assisted work. Each task and its initial instruction were generated deterministically from a tutorial-grounded structured specification, while generative AI was used to revise the instruction for clarity and concision. The revisions were checked against the fixed task conditions by an independent critic and deterministic rule-based validation, and AI-assisted code and reported results were verified through automated tests and saved evaluator outputs. We take responsibility for the final content of this work, including text, claims, code, data, figures, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We plan to publicly release the TimelineBench code and benchmark artifacts upon acceptance, including the construction pipeline, the predicate-based evaluator, and OpenCut-TL artifacts. We detail benchmark construction, evaluation, and implementation in §4, §5, and §6, with further details in Appendices A, B, and C. These resources are intended to support reproduction of our experiments and future evaluations of video-editing agents.

REFERENCES

Jiaxin Ai, Yukang Feng, Fanrui Zhang, Jianwen Sun, Zizhen Li, Chuanhao Li, Yifan Chang, Wenxiao Wu, Ruoxi Wang, Mingliang Zhai, and Kaipeng Zhang. Prosoftarena: Benchmarking hierarchical capabilities of multi-modal agents in professional software environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 34586–34595, June 2026.

Yunpeng Bai, Yossi Gandelsman, Michaël Gharbi, and Qixing Huang. Instruction-based video editing by repurposing an image editing model, 2026. URL https://arxiv.org/abs/2608.14790.

Yinan Chen, Jiangning Zhang, Teng Hu, Yuxiang Zeng, Zhucun Xue, Qingdong He, Chengjie Wang, Yong Liu, Xiaobin Hu, and Shuicheng YAN. IVEBench: Modern benchmark suite for instruction-guided video editing assessment. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=n0wVbCxcob.

Xiangbo Gao, Sicong Jiang, Bangya Liu, Xinghao Chen, Minglai Yang, Siyuan Yang, Mingyang Wu, Jiongze Yu, Qi Zheng, Haozhi Wang, et al. Vefx-bench: A holistic benchmark for generic video editing and visual effects. *arXiv preprint arXiv:2604.16272*, 2026.

Kilem L. Gwet. Computing inter-rater reliability and its variance in the presence of high agreement. *British Journal of Mathematical and Statistical Psychology*, 61(1):29–48, May 2008. doi: 10.1348/000711006X126600.

Haobo Hu, Xiangwu Guo, Zhiheng Chen, Difei Gao, Haotian Liu, Libiao Jin, and Qi Mao. CutVerse: A compositional GUI agents benchmark for media post-production editing. *arXiv preprint arXiv:2605.19484*, 2026. URL https://arxiv.org/abs/2605.19484.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Xin Liang, Xiang Zhang, Yiwei Xu, Siqi Sun, and Chenyu You. Slidegen: Collaborative multimodal agents for scientific slide generation. *arXiv preprint arXiv:2512.04529*, 2025.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Qinchen Wu, Mingyi Yan, Zhengyuan Yang, Lijuan Wang, and Mike Zheng Shou. VideoGUI: A benchmark for GUI automation from instructional videos. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/804e757b7d7043c26701c3a313032101-Abstract-Datasets_and_Benchmarks_Track.html.

Michael Ofengenden, Yunze Man, Ziqi Pang, Liang-Yan Gui, and Yu-Xiong Wang. PPTArena: A benchmark for PowerPoint editing. *arXiv preprint arXiv:2512.03042*, 2025. URL https://arxiv.org/abs/2512.03042v3.

OpenAI. Computer use. https://developers.openai.com/api/docs/guides/tools-computer-use, a.

OpenAI. Openai models. https://developers.openai.com/api/docs/models, b.

OpenCut Project. OpenCut Classic. https://github.com/OpenCut-app/opencut-classic. Archived software repository, MIT License; accessed 2026-09-26.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William E Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Kenji Toyama, Robert James Berry, Divya Tyamagundlu, Timothy P Lillicrap, and Oriana Riva. Androidworld: A dynamic benchmarking environment for autonomous agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=il5yUQsrjC.

Marcelo Sandoval-Castañeda, Bryan Russell, Josef Sivic, Gregory Shakhnarovich, and Fabian Caba Heilbron. Editduet: A multi-agent system for video non-linear editing. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference (SIGGRAPH)*, 2025.

Chenrui Shi, Yuwei Wu, Yang Liu, Ruining Feng, Zirui Shang, Zhi Gao, Lifeng Fan, and Che Sun. Interactive reward agent: Gui task evaluation via environment-state verification, 2026. URL https://arxiv.org/abs/2607.25904.

Wenhao Sun, Rong-Cheng Tu, Jingyi Liao, and Dacheng Tao. Diffusion model-based video editing: A survey, 2024. URL https://arxiv.org/abs/2407.07111.

Wenxin Tang, Jingyu Xiao, Wenxuan Jiang, Xi Xiao, Yuhang Wang, Xuxin Tang, Qing Li, Yuehe Ma, Junliang Liu, Shisong Tang, and Michael R. Lyu. SlideCoder: Layout-aware RAG-enhanced hierarchical slide generation from design. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 9015–9039, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.458. URL https://aclanthology.org/2025.emnlp-main.458/.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *ACL*, 2024.

Bryan Wang, Yuliang Li, Zhaoyang Lv, Haijun Xia, Yan Xu, and Raj Sodhi. Lave: Llm-powered agent assistance and language augmentation for video editing. *arXiv preprint arXiv:2402.10294*, 2024.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, Zhennan Shen, Zhuokai Li, Ryan Li, Xiaochuan Li, Junda Chen, Boyuan Zheng, Peihang Li, Fangyu Lei, Ruisheng Cao, Yeqiao Fu, Dongchan Shin, Martin Shin, Jiarui Hu, Yuyan Wang, Jixuan Chen, Yuxiao Ye, Danyang Zhang, Yipu Wang, Heng Wang, Diyi Yang, Victor Zhong, Y. Charles, Zhilin Yang, and Tao Yu. OpenCUA: Open foundations for computer-use agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=6iRZvJiC9Q.

Jiangtao Wu, Jiaming Wang, Yiwen He, Yuanxing Zhang, Shihao Li, Dunyuan Liu, Xuedong Zhao, Jialu Chen, Zekun Moore Wang, and Jiaheng Liu. Covebench: Can video editing models handle complex instructions?, 2026. URL https://arxiv.org/abs/2606.08415.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments, 2024.

Ximing Xing, Ziteng Xue, Zhenxi Li, Weicong Liang, Linqing Wang, Zhantao Yang, Tiankai Hang, Zijin Yin, Qinglin Lu, Chunyu Wang, and Qian Yu. Hierarchical svg tokenization: Learning compact visual programs for scalable vector graphics modeling. *arXiv preprint arXiv:2604.05072*, 2026.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=oKn9c6ytLx.

A Benchmark Construction Details

A.1 Atomic Operation Vocabulary

An atomic operation is the smallest change to a project that can be requested on its own, regardless of the editor used. One operation may require several GUI actions. For example, splitting a clip with a keyboard shortcut or a razor tool counts as the same operation, `split_clip`. A standard trim and a ripple trim are separate operations because they affect the surrounding timeline differently. Table 4 lists all 48 operations.

Table 4: The 48 atomic operations in TimelineBench, grouped by the type of edit.

Family	Count	Atomic operations
Timeline editing	20	`split_clip`, `trim_clip_start`, `trim_clip_end`, `delete_clip`, `delete_gap`, `move_clip`, `duplicate_clip`, `insert_clip`, `rearrange_clip`, `ripple_delete_clip`, `slip_clip`, `roll_edit`, `slide_clip`, `ripple_trim_start`, `ripple_trim_end`, `connect_insert_clip`, `overwrite_clip`, `replace_clip`, `replace_clip_preserve_timing`, `move_clip_group`
Visual layout and effects	8	`set_scale`, `set_position`, `set_rotation`, `set_opacity`, `set_crop_side`, `set_clip_effect`, `set_dynamic_zoom`, `frame_clip`
Playback	2	`set_speed`, `set_reverse`
Text and graphics	11	`create_text_graphic`, `set_text_content`, `set_text_position`, `trim_text_start`, `trim_text_end`, `delete_text_graphic`, `set_font_family`, `set_font_size`, `set_font_weight`, `set_text_color`, `set_text_alignment`
Video transitions	3	`apply_video_transition`, `remove_video_transition`, `set_video_transition_duration`
Keyframes	1	`set_parameter_keyframe`
Audio editing	2	`set_audio_gain`, `set_audio_fades`
Timeline markers	1	`add_timeline_marker`
Total	48	

Some operations cover more than one interface action. For example, `set_parameter_keyframe` includes both creating a keyframe and updating an existing one. Tasks combine these operations in an ordered sequence, but each combination is not counted as a new atomic operation.

A.2 Operation-to-Interface Mapping

Tables 5 and 6 show the controls available for each operation. OpenCut Classic already supported 29 operations. We added or modified controls in OpenCut-TL for the remaining 19. Agents may use any valid interaction sequence, including routes other than those listed in the tables.

We tested the added and modified controls with 178 native and browser tests. These check timing and source ranges, preservation of unrelated objects, rejection of invalid edits, undo and redo, and saving and restoring project state. All 178 tests passed in the experiment configuration.

Two dataset operations reuse existing controls: `move_clip_group` uses the route for `move_clip`, and `replace_clip_preserve_timing` uses the route for `replace_clip`.

Table 5: Base OpenCut-TL operation-to-interface mapping (29 operations).

Operations	Available UI route
split_clip	Clip and playhead → Split
trim_clip_start, trim_clip_end, trim_text_start, trim_text_end	Drag the object's left or right edge
delete_clip, delete_text_graphic, delete_gap	Select object or gap → Delete
move_clip, duplicate_clip, insert_clip	Drag, Duplicate, or Asset → Insert
set_scale, set_position, set_rotation, set_crop_side, set_opacity	Clip → Properties → Transform, Crop, or Blending
set_speed, set_reverse	Clip → Properties → Speed
create_text_graphic, set_text_content, set_text_position, set_font_family, set_font_size, set_font_weight, set_text_color, set_text_alignment	Text panel or selected text → Properties
apply_video_transition, remove_video_transition, set_video_transition_duration	Timeline transition control or Properties → Transition

Table 6: TimelineBench workflow-extension mapping (19 operations).

Operations	Available UI route
ripple_trim_start, ripple_trim_end	Properties → Ripple Trim
ripple_delete_clip	Clip context menu → Ripple Delete
roll_edit, slip_clip, slide_clip	Properties → Native Trim → mode
rearrange_clip	Clip context menu → Rearrange
overwrite_clip, replace_clip	Asset menu → Overwrite or Replace
connect_insert_clip	Asset menu → Connect
frame_clip	Properties → Framing
set_clip_effect	Properties → Effects
set_parameter_keyframe	Property diamond and keyframe value field
set_dynamic_zoom	Properties → Dynamic Zoom
set_audio_gain, set_audio_fades	Properties → Audio or Audio Fade
add_timeline_marker	Playhead → marker control

A.3 Task Template Inventory

Of the 121 task templates, 95 come from executable, contiguous subworkflows in the source tutorials. We added 26 tutorial-grounded compositions to cover additional atomic operations. Each template stores an ordered sequence of operations from Appendix A.1.

The list below groups templates by their main requested edit. Templates spanning several editing families appear only once. We use workflow names where available and operation sequences otherwise. The group counts sum to 121.

Timeline structure and source selection (35).

close-rejected-interval-gap;
complete-chronological-story-insert; connect_insert_clip;
curate-selects; curate-selects-boundary; delete_clip +
delete_gap; duplicate_clip; insert_clip + insert_clip;
middle-shot-placement-and-passage; move-shot-into-free-space;
narrative-rough-cut-repair; open-interval-with-group-offset;
overwrite_clip; remove-obsolete-editorial-note;
remove-unwanted-occurrence-preserve-timing;
reorder-story-clips; replace_clip; restore-hard-cut;
revised-v2-ordered-excerpts; ripple-end-sequence-continuity;
ripple_trim_start; roll-shared-edit; slip-source-content;

slip-then-slide-middle-passage; split_clip + delete_clip;
split_clip + split_clip + delete_clip + delete_gap;
state-grounded-v1-ordered-story-sections;
story-relocation-entrance-continuity; trim_clip_start;
v6-dv01-t5; v6-dv02-t1; v6-dv02-t3; v6-tt-extend-clip-edge;
v6-tt-instant-replay; v6-tt-jump-cut-condensation.

Visual properties, playback, and transitions (20).

apply-adjustable-effect; apply_video_transition +
set_video_transition_duration; crop-distracting-edge;
edge-cleanup-fill; focused-mixed-v1-bounded-blur-treatment;
focused-mixed-v2-interior-passage-reverse;
grounded-additional-v1-interior-rejection;
grounded-additional-v1-purpose-04;
grounded-additional-v1-purpose-06;
position-shot-within-canvas; reverse-footage;
revised-v2-three-shot-full-image-fit; scoped-shot-scale;
selected-shot-opacity; shot-rotation-existing-anchor;
soft-full-canvas-backing; soften-two-shot-cut;
spatially-conform-footage; two-shot-dissolve-duration;
v6-tt-constant-speed-change.

Text and graphics (50).

create_text_graphic + set_font_family; create_text_graphic +
set_font_family + set_font_weight; create_text_graphic +
set_font_family + set_font_weight + set_font_size;
create_text_graphic + set_font_family + set_font_weight +
set_font_size + set_text_color + set_text_position +
trim_text_end; create_text_graphic + set_text_content;
create_text_graphic + set_text_content + set_font_family;
create_text_graphic + set_text_content + set_font_family +
set_font_weight; create_text_graphic + set_text_content +
set_font_family + set_font_weight + set_font_size +
set_text_alignment + set_text_color + set_text_position +
trim_text_start + trim_text_end; opening-identification;
opening-identification-alignment;
opening-identification-font-size;
opening-identification-position;
opening-identification-start-trim;
opening-identification-trim-end; opening-title-color;
opening-title-font-family; opening-title-normal-weight;
revise-existing-editorial-note; set_font_family +
set_font_weight; set_font_family + set_font_weight +
set_font_size; set_font_family + set_font_weight + set_font_size
+ set_text_alignment; set_font_family + set_font_weight +
set_font_size + set_text_color; set_font_size +
set_text_alignment; set_font_size + set_text_alignment +
set_text_color; set_font_size + set_text_alignment +
set_text_color + set_text_position; set_font_size +
set_text_color; set_font_size + set_text_color +
set_text_position; set_font_size + set_text_color +
set_text_position + trim_text_end; set_font_weight +
set_font_size; set_font_weight + set_font_size +
set_text_alignment; set_font_weight + set_font_size +
set_text_alignment + set_text_color; set_font_weight +
set_font_size + set_text_color; set_font_weight + set_font_size
+ set_text_color + set_text_position; set_text_alignment +
set_text_color; set_text_alignment + set_text_color +
set_text_position; set_text_alignment + set_text_color +
set_text_position + trim_text_start; set_text_color +
set_text_position; set_text_color + set_text_position +

trim_text_end; set_text_color + set_text_position + trim_text_start; set_text_color + set_text_position + trim_text_start + trim_text_end; set_text_content + set_font_family; set_text_content + set_font_family + set_font_weight; set_text_content + set_font_family + set_font_weight + set_font_size; set_text_position + trim_text_end; set_text_position + trim_text_start; set_text_position + trim_text_start + trim_text_end; shot-timed-horizontal-title; trim_text_start + trim_text_end; v6-tt-animated-secondary-credit; v6-tt-timed-closing-title.

Audio, animation, and markers (16).

adjust-clip-volume; animate-opening-logo; fade-audio-boundary; grounded-audio-db-v3-p05; incoming-passage-shared-cut; interior-passage-ripple-trims; keyframe-audio-property; movie-opening-identification; movie-opening-identification (variant 2); nudge-clip-exactly; purpose-mixed-db-v1-p08; selects-navigation-marker; set_parameter_keyframe + set_parameter_keyframe; v6-fc09-t2; v6-tt-build-continuous-ambience; v6-tt-duck-music-under-dialogue.

A.4 Base Timeline Layouts

TimelineBench constructs initial project states from eight reusable, declarative layout specifications rather than creating each project independently (Table 7). Each specification defines the canvas, frame rate, tracks, objects, and initial intervals. Task materialization binds concrete media and task identifiers while retaining the selected layout's structure and timing. All layouts use 24 fps. Cross-dissolve variants are obtained by adding a transition to the two-clip layout and are therefore not counted as additional base layouts.

Table 7: Reusable base timeline layouts. Clip and text intervals are in frames.

Layout	Canvas	Objects	Initial arrangement
One clip	1920×1080	1 video	One V1 clip in $[0, 598)$.
Wide-canvas one clip	1280×720	1 video	One V1 clip in $[0, 598)$; varies project resolution.
Two clips	1920×1080	2 video	Contiguous V1 clips in $[0, 598)$ and $[598, 983)$.
Three clips	1920×1080	3 video	Contiguous V1 clips in $[0, 400)$, $[400, 700)$, and $[700, 950)$.
Short gap	1920×1080	2 video	V1 clips in $[0, 360)$ and $[420, 805)$; a 60-frame gap.
Long gap	1920×1080	2 video	V1 clips in $[0, 240)$ and $[420, 720)$; a 180-frame gap.
One title	1920×1080	1 video, 1 text	One V1 clip and one title in $[72, 240)$.
Two titles	1920×1080	1 video, 2 text	One V1 clip and distinguishable titles in $[24, 120)$ and $[200, 360)$.

Operation-to-layout compatibility is determined before task instantiation. For example, text-editing templates require a layout containing a text object, whereas two-clip ordering templates require two adjacent clips. This filtering prevents structurally incompatible task bindings.

B Instruction Revision and Verification Details

We first fix the initial project, operation sequence, targets, parameter values, and request and preservation predicates. A deterministic compiler uses these fields to write the initial instruction. We then

ask a language-model author to shorten the wording and a separate critic to check it against the same specification. Only the instruction is revised.

B.1 Inputs to the Author and Critic

The inputs are listed in Table 8. The author works from the original instruction and task specification. The critic receives both, along with the proposed revision. Evaluation conditions are available to these two models during dataset construction. They are not shown to the agent that performs the task.

Table 8: Inputs to instruction revision and verification.

Field	Information supplied to the model
Task metadata	Exact task identity and task purpose, when available
Original instruction	Complete instruction produced before concise revision
Initial state	Abstract initial requirements and the materialized initial project
Operation program	Ordered operations, dependencies, targets, and parameters
Request conditions	Every requested final-state condition with a stable condition ID
Preservation conditions	Every protected-state condition with a stable condition ID
Evaluator context	Clarifications needed to interpret condition fields and temporal signs

B.2 Author Prompt

The author is asked to remove unnecessary wording while keeping the requested edits and constraints. We reproduce the prompt below, with the output-schema description abbreviated.

```
Rewrite each original English instruction as the shortest clear
natural request that remains faithful to the supplied concrete
specification and evaluator conditions.  Keep every requested change,
target identifier, independent quantity, source range, title string,
necessary neighboring consequence, and non-default property.
Preserve operation order only where the requested outcome depends on
it, and do not prescribe a unique GUI trajectory.

Treat ordinary preservation of unrelated state as implicit.  Remove
redundant lists of unchanged media, properties, audio settings, or
tracks, and do not replace them with a generic ``leave everything
else unchanged.''  Keep an explicit preservation clause only when
omitting it permits a materially different valid edit, including gap
versus ripple behavior, fixed outer bounds, source-range versus
timeline movement, aspect ratio, static versus animated controls,
keyframe interpolation, or another stated editorial constraint.

Do not invent semantic video content or repair a questionable
specification by adding requirements.  If a condition is surprising,
conflicting, implementation-dependent, or invalid, report it for
specification review.  Return the task ID, the complete revised
instruction, a brief account of removed boilerplate and retained
disambiguation, and any specification concerns.
```

B.3 Independent Critic Prompt

The critic checks each request and preservation condition against the revised instruction. Its response records how each condition is covered, using one of the four labels defined in the prompt below.

```
Independently audit each revised instruction against its original
task packet under the same concise-instruction policy.  Check every
target, value, range, default, audio state, and required neighboring
consequence.  Do not pass an instruction merely because it is
shorter, and flag specification defects rather than masking them.

For every supplied condition ID, assign exactly one coverage mode:
explicit, implied_by_request, ordinary_preservation, or
missing_or_invalid.  Quote an exact instruction substring as evidence
for explicit or implied coverage, and give a short justification for
every mode.  Ordinary preservation may normally cover only
preservation conditions, never an omitted primary request.

Return a verdict of pass, revise, or spec_review, concrete issues, the
complete condition-by-condition coverage record, and Boolean checks
that redundant boilerplate was removed and no unsupported requirement
was added.
```

B.4 Acceptance Gate and Model

We accept an instruction only if the author reports no specification concerns and the critic returns `pass` with no issues. The critic must also confirm that the revision removes redundant wording and adds no unsupported requirements. These judgments are followed by checks in code. Every condition ID must have exactly one coverage record and a justification, and the instruction must be nonempty. Explicit and implied coverage require a quote from the revised text. A request condition labeled as ordinary preservation causes rejection.

GPT-6 Astra serves as both author and critic. Each batch contains ten tasks, with at most two author–critic attempts per unresolved task in a run. Accepted revisions are kept without further generation. We save the requests and parsed responses, then check the output files before release. These checks cover the schema, task-set membership, condition coverage, and provenance hashes. We also compare the files with the originals: only the instruction and its provenance metadata may change. The operation program, initial state, targets, parameters, predicates, and evaluator must remain unchanged.

C Supplementary Experimental Details

C.1 Execution Settings

OpenCut-TL episodes run in a 1280×720 viewport with a limit of 80 actions. For Premiere, screenshots are resized to at most 1280 pixels on the longer side. The OpenCut-TL Guide is accessible through an on-screen toggle, rather than included in full in the initial prompt.

Table 9: Model inference settings for OpenCut experiments. Output limits are in tokens per response.

Model	Reasoning effort	Output limit	History
GPT-6 Sol	Medium	4,096	Stateful
GPT-5.6 Sol	Medium	4,096	Stateful
GPT-6 Luna	Medium	4,096	Stateful
GPT-5.6 Luna	Medium	4,096	Stateful
GPT-5.4 mini	Medium	4,096	Stateful
UI-TARS-1.5-7B	Not used	1,024	Current image
OpenCUA-7B	Not used	4,096	Three images

All GPT models use medium reasoning effort and retain conversation history through the provider. UI-TARS receives the current screenshot and text action history. OpenCUA retains three screenshots.

C.2 AGENT PROMPT

The GPT initial prompt is shown below. UI-TARS and OpenCUA use their native prompt formats with the same Guide and timebase guidance.

```
You are operating the OpenCut Classic web video editor in Chromium.

If you are unsure how to use a feature, please consult the Guide
toggle above.  Search the guide for explanations of most features.
Also note that the timeline runs at 24 fps:  40 frames corresponds to
00:00:01:16.

Use the computer tool and inspect each resulting screenshot.

Instruction:  {natural_instruction}

Remaining action budget:  80

Use only what is visible in the screenshots.  Do not guess hidden
evaluation criteria.

When the instruction is fully satisfied and visibly confirmed, stop
issuing actions and reply with a short message stating the task is
complete.
```

C.3 HUMAN COMPARISON

Human success rate is computed over all 200 assignments, with skipped assignments counted as failures. Participants succeeded in 129 assignments (64.50%). Supplementary coverage is computed from the 186 completed attempts. We average coverage across completed attempts within each task and then across all 100 tasks, each of which has at least one completed attempt. This gives request coverage of 79.36% and preservation coverage of 98.08%. All completed attempts are evaluated using the same task predicates and evaluator as the agents.

C.4 CROSS-EDITOR EVALUATION

OpenCut provides an on-screen Guide and is evaluated through project-state JSON, while Premiere is evaluated through exported XML. The final Premiere results include 35 successes and 21 failures for GPT-6 Sol, and 27 successes and 29 failures for GPT-6 Luna. Coverage is averaged over all 56 tasks, assigning preservation coverage of one to the nine tasks without preservation conditions.

Table 10: Performance on the same 56 tasks in OpenCut and Adobe Premiere Pro. All values are percentages and coverage is macro-averaged over tasks.

Model	Editor	SR	Request cov.	Preservation cov.
GPT-6 Sol	OpenCut	83.93	86.43	100.00
	Premiere	62.50	73.30	98.97
GPT-6 Luna	OpenCut	58.93	62.80	97.77
	Premiere	48.21	61.85	94.84

C.5 Task-Level Results

Task level denotes the number of atomic operations in a task. Success rates do not decrease monotonically with level because the groups differ in operation composition. Levels 5–10 contain only 4–10 tasks each.

Table 11: Success rates (%) by task level. Task level is the number of atomic operations in a task.

Level	Tasks	GPT-6 Sol	GPT-5.6 Sol	GPT-6 Luna	GPT-5.6 Luna	GPT-5.4 mini	UI-TARS	OpenCUA
1	303	92.74	89.44	84.49	79.21	57.76	8.25	7.92
2	166	80.72	84.94	51.81	54.22	42.77	4.22	1.81
3	207	83.57	78.26	63.77	61.84	40.10	0.00	0.00
4	71	94.37	88.73	90.14	83.10	59.15	0.00	0.00
5	4	100.00	100.00	75.00	100.00	25.00	0.00	0.00
6	5	100.00	40.00	100.00	100.00	0.00	0.00	0.00
7	5	100.00	100.00	40.00	100.00	0.00	0.00	0.00
8	10	40.00	40.00	40.00	40.00	0.00	0.00	0.00
9	5	60.00	20.00	20.00	0.00	0.00	0.00	0.00
10	5	100.00	100.00	100.00	80.00	0.00	0.00	0.00

Entries are success rates (%). Task counts are identical across models.

C.6 Failure Analysis Details

We analyze saved evaluator results for all seven models on 781 tasks, covering 5,467 OpenCut episodes.

Episode outcomes. Failed episodes in Figure 3(a) are assigned to three mutually exclusive groups. *Request only* indicates unmet request conditions without preservation violations. *Preservation only* indicates satisfied request conditions with preservation violations, including unrequested changes. *Both* indicates violations of both kinds.

Editing categories. The eight categories below group conditions by editing functionality. A task may contribute to multiple categories. Text timing belongs to timeline editing, while text content and style belong to text. Fixed visual settings belong to visual properties, while changes to those settings over time belong to animation.

Category	Editing requirements
Timeline editing	Set clip or text timing and source intervals. Insert, delete, move, split, duplicate, connect, or replace clips, and maintain the required order and continuity.
Visual properties	Set framing, position, scale, crop, opacity, and other fixed effect parameters.
Text	Create or remove text elements and set their content and style.
Animation	Vary clip parameters over time using keyframes, curves, waypoints, or dynamic zoom.
Transitions	Add or remove transitions and set their duration.
Audio	Set volume and fades, or preserve whether embedded audio is enabled.
Playback	Set playback speed or reverse direction.
Markers	Place timeline markers with the required attributes.

Action-budget exhaustion. UI-TARS and OpenCUA exhaust the 80-action budget on 63.51% and 75.93% of tasks, compared with 2.18% for GPT-6 Sol and 18.31% for GPT-5.4 mini (Table 12). These termination counts overlap the predicate-based failure categories.

Table 12: Episodes reaching the 80-action limit out of 781 tasks per model. Reaching the limit does not necessarily indicate task failure.

Model	Episodes	Rate (%)
GPT-6 Sol	17	2.18
GPT-5.6 Sol	50	6.40
GPT-6 Luna	101	12.93
GPT-5.6 Luna	102	13.06
GPT-5.4 mini	143	18.31
UI-TARS-1.5-7B	496	63.51
OpenCUA-7B	593	75.93