TEXT2CUT: CONTROLLING NON-LINEAR VIDEO EDITING WITH LANGUAGE

Anonymous authors
Paper under double-blind review

ABSTRACT

Non-linear video editing manipulates clips, tracks, transitions, and effects over structured timelines, yet existing editing systems often rely on software-specific representations, making it challenging to translate natural-language instructions into portable and executable edits. OpenTimelineIO (OTIO) provides a software-agnostic abstraction for timeline-level editing, but does not uniformly represent many effect-level transformations required in practical workflows. To this end, we introduce **Text2Cut**, a language-driven non-linear video editing framework that extends OTIO with a unified space of executable operations spanning both timeline-level manipulations and effect-level transformations. Rather than directly generating a complete timeline, Text2Cut predicts atomic editing operations that are executed and validated over structured timeline states, with the resulting state compiled into a portable OTIO-based representation. Furthermore, we train Text2Cut on a hybrid corpus of synthetic trajectories and real editing projects, and evaluate it on controlled compositional benchmarks and held-out real editing workflows. Experiments show that Text2Cut achieves highest final-state correctness while maintaining strong execution validity. Text2Cut Demos are available at `https://lucent-fudge-62586f.netlify.app/`.

1 INTRODUCTION

Non-linear video editing (NLE) organizes media assets over structured multi-track timelines, where editors manipulate clips through operations such as trimming, reordering, transitions, compositing, and visual effects. Unlike pixel-level video editing (Wu et al., 2026; Qu et al., 2025; Hu et al., 2025), which directly modifies visual appearance, NLE operates over an explicit temporal structure and preserves the underlying editing workflow. Professional tools such as Adobe Premiere Pro, Final Cut Pro, and DaVinci Resolve (Adobe, 2026; Apple, 2026; Design, 2026) expose this structure through timelines, tracks, clips, transitions, and effect controls. While powerful, these systems require substantial manual interaction and familiarity with software-specific editing abstractions. Recent advances in language models therefore offer an appealing alternative: allowing users to express editing intent directly in natural language (Zeng et al., 2025; Raheja et al., 2023).

However, translating natural-language instructions into executable non-linear edits remains challenging because editing systems rely on software-specific representations. A high-level instruction such as *"move clip2 to the first position, add a dissolve transition between them"* must be grounded into concrete timeline entities, temporal operations, and effect parameters. Yet Premiere Pro, Final Cut Pro, and DaVinci Resolve encode these decisions differently, tightly coupling language-driven systems to individual editing environments. Existing approaches often sidestep this problem by producing edited pixels, high-level scripts, or platform-specific commands, rather than a unified intermediate representation that is both structurally constrained and directly executable. As illustrated in Fig. 1(a), this representation gap limits the portability, verifiability, and reusability of language-driven editing systems.

OpenTimelineIO (OTIO) (Foundation, 2026) provides a natural foundation for addressing this gap. As an open, software-agnostic interchange format, OTIO explicitly models editorial structures such as clips, tracks, media references, transitions, and temporal ranges, enabling portable timeline-level editing. Practical editing, however, extends beyond timeline organization. Parameterized transformations, animations, masks, compositing, and other visual effects are not uniformly represented

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.

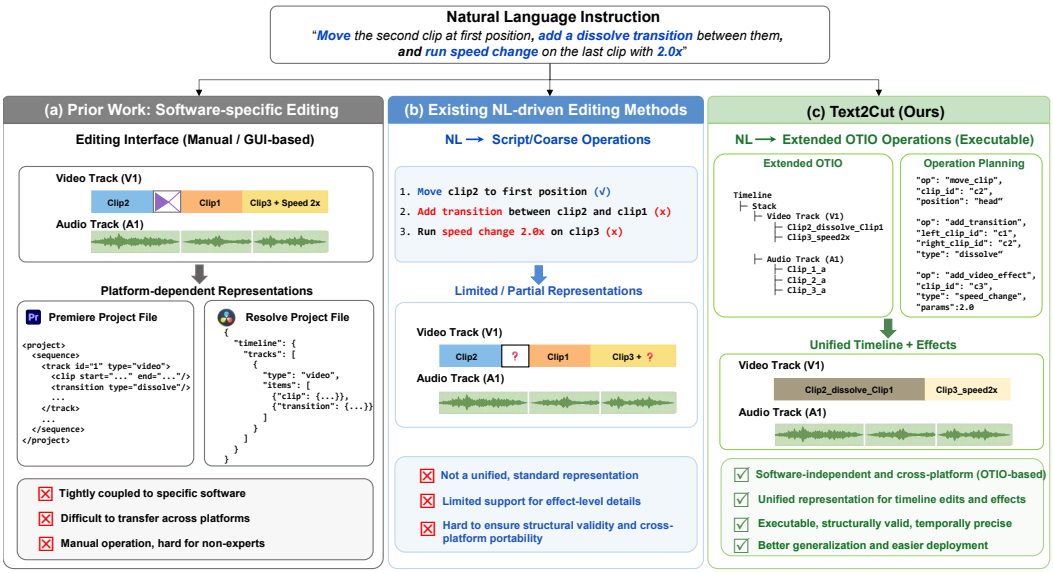

Figure 1: **Comparison of non-linear video editing.** (a) Traditional editing relies on software-specific timelines and manual operations. (b) Existing natural-language-driven methods produce scripts or partial representations with limited effect support and portability. (c) Text2Cut translates natural-language instructions into a unified OTIO representation for cross-platform, executable timeline- and effect-level editing.

by standard OTIO and often rely on editor-specific mechanisms. As illustrated in Fig. 1(b), OTIO alone does not provide a complete executable abstraction for the range of edits expressed in natural language. The key challenge is therefore to bridge free-form language with a unified executable representation that captures both timeline structure and effect-level transformations.

To address this challenge, we introduce **Text2Cut**, a language-driven non-linear video editing framework built on an extended OTIO-based executable operation space. Rather than directly predicting a complete hierarchical timeline, Text2Cut translates editing requests into atomic operations spanning both timeline-level manipulations and effect-level transformations. These operations are executed and validated over structured timeline states, while the resulting state is compiled into an executable OTIO representation. For effects not natively supported by standard OTIO, Text2Cut materializes their outputs as derived media assets and reintegrates them into the timeline, while preserving their operation semantics for subsequent revision. This design maintains a software-agnostic OTIO-based editing state without restricting the system to the effect primitives supported by standard OTIO. To learn this structured editing space, we construct a hybrid instruction–operation corpus combining synthetic trajectories with editing trajectories collected from human editors. Experiments show that Text2Cut achieves highest final-state correctness compared with general-purpose language-model baselines. Our contributions are summarized as follows:

- We formulate language-driven non-linear video editing through a unified OTIO-based executable operation space that bridges natural-language instructions with both timeline-level manipulations and effect-level transformations.
- We introduce **Text2Cut**, which represents editing requests as atomic executable operations and validates their execution over structured timeline states, while compiling the resulting edits into a portable and revisable OTIO-based representation.
- We construct a hybrid corpus of synthetic and human editing trajectories and evaluate Text2Cut on both controlled and held-out real editing workflows, demonstrating substantially improved execution reliability and final-state correctness over general-purpose language-model baselines.

2 Related Work

Language-Driven Video Editing. Language-driven video editing has been widely studied in the context of generative video manipulation (Yu et al., 2025; Vinker et al., 2025; Molad et al., 2023; Wu et al., 2023). Existing methods allow users to modify visual content through natural-language instructions, supporting tasks such as appearance editing, object manipulation, and style transfer (Molad et al., 2023; Wu et al., 2023). Most of these approaches operate directly at the pixel or latent level and produce rendered videos as their final outputs, without exposing the underlying editing structure. More recent work has explored timeline-based or programmatic video assembly and editing (Pardo et al., 2024; Sandoval-Castaneda et al., 2025), moving toward more structured editing workflows. However, these approaches generally do not provide a unified, portable, and directly executable representation that jointly covers timeline structure and effect-level transformations. In contrast, Text2Cut translates natural-language instructions into explicit executable operations over structured non-linear editing timelines.

OpenTimelineIO for Structured Editing. OpenTimelineIO (OTIO) (Foundation, 2026) is an open, software-agnostic interchange format for representing editorial timelines. It provides a common abstraction for core timeline structures, including tracks, clips, media references, transitions, and temporal ranges, and is therefore well suited for describing portable timeline-level edits across different editing environments. By separating editorial structure from the internal project formats of individual applications, OTIO offers a natural foundation for language-driven systems that aim to manipulate non-linear editing timelines in a software-independent manner. However, standard OTIO primarily focuses on editorial structure and does not uniformly represent many effect-level operations required in practical editing workflows. Parameterized transformations, animations, masks, compositing, and other visual effects are often implemented through application-specific mechanisms or metadata conventions rather than a shared executable abstraction. As a result, OTIO alone is insufficient for directly expressing the full range of edits that may be requested in natural language. To this end, Text2Cut builds on OTIO as the underlying timeline representation while introducing an executable operation layer that covers both timeline-level manipulations and effect-level transformations. For operations that are not natively represented by OTIO, Text2Cut materializes the transformed media as derived assets and preserves the associated operation semantics in metadata, allowing the resulting editing state to remain portable and subsequently revisable.

3 Method

3.1 Overview

Given a sequence of natural-language editing instructions $\mathbf{X} = (x_1, \ldots, x_K)$, an initial timeline state S_0, and an initial asset pool $\mathcal{A}_0$, Text2Cut produces an executable editing trajectory $\mathcal{O} = (o_1, \ldots, o_T)$, where o_t denotes the t-th atomic editing operation. However, directly mapping language to a complete OpenTimelineIO (OTIO) representation is challenging because timeline elements are tightly coupled by hierarchical, temporal, and asset-level dependencies: even a local edit may require coordinated changes to clips, tracks, temporal ranges, media references, and effect parameters. Text2Cut therefore introduces an executable operation layer between natural language and OTIO and performs editing over an evolving structured state. As illustrated in Fig. 2, the framework consists of three stages: (1) **operation planning**, which grounds the instruction into canonical editing operations conditioned on the current editing context; (2) **incremental execution**, which applies each operation and validates the resulting state; and (3) **OTIO compilation**, which maps the resulting state to an OTIO representation and materializes effect-level transformations when necessary. Together, these stages decompose a globally coupled timeline generation problem into a sequence of executable and verifiable editing decisions.

Operation Planning. Text2Cut plans editing operations over an evolving editing context. At step t, the planner predicts the next operation conditioned on the instruction sequence $\mathbf{X}$, the current timeline state S_{t-1}, and the current asset pool $\mathcal{A}_{t-1}$:

$$o_t \sim p_\theta(\cdot \mid \mathbf{X}, S_{t-1}, \mathcal{A}_{t-1}), \tag{1}$$

where p_θ denotes the language-model-based planner. At the first step, the planner operates on $(S_0, \mathcal{A}_0)$; subsequent operations are predicted from the context produced by preceding edits. Each

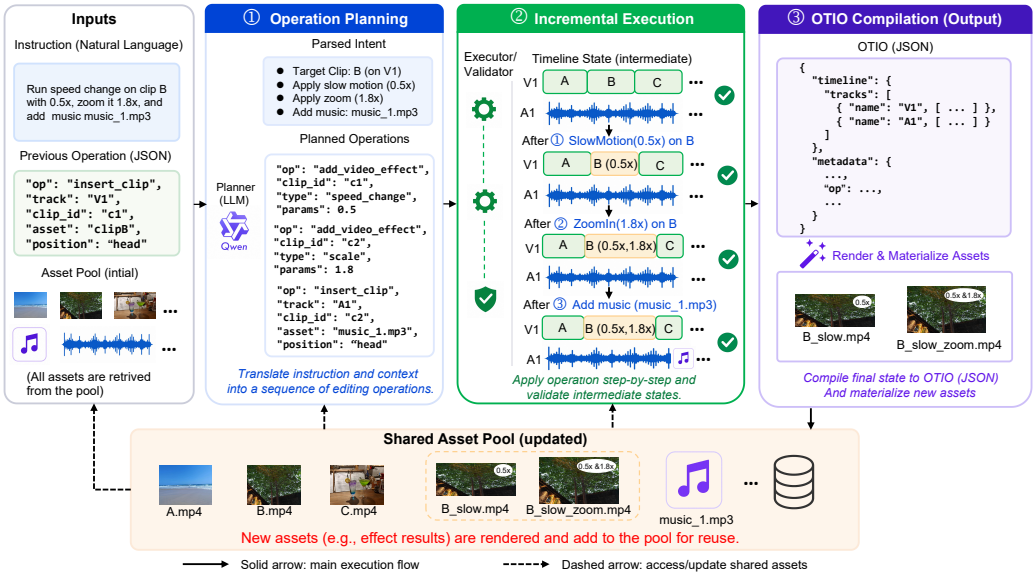

Figure 2: **The overall architecture of TextCut**, It consists of three parts: operation planning, incremental execution and OTIO compilation. Note that since our final output is an OTIO file rather than an explicit sequence of editing operations, we store the historical operations in the metadata of the OTIO file, so they can be reused in subsequent editing steps.

operation is represented as

$$o_t = (\tau_t, r_t, \phi_t), \tag{2}$$

where τ_t specifies the operation type, r_t identifies the target timeline entities or media assets, and ϕ_t contains operation-specific parameters. The operation space contains 78 executable operations spanning both timeline-level manipulations and effect-level transformations. Because preceding edits may change both the timeline structure and the available assets, later operations are grounded against the updated editing context rather than the initial state.

Incremental Execution. Once an operation o_t is predicted, Text2Cut applies it to the current editing context using a deterministic executor:

$$(S_t, \mathcal{A}_t) = E(S_{t-1}, \mathcal{A}_{t-1}, o_t), \tag{3}$$

where S_t and $\mathcal{A}_t$ denote the updated timeline state and asset pool. Each operation induces a localized state transition, avoiding the need to regenerate the complete timeline after every edit. A deterministic validator checks the resulting state for structural consistency, including valid entity references, track assignments, temporal relationships, asset availability, and operation-specific constraints. Only a valid state is carried forward as the context for subsequent editing operations. Importantly, the validator assesses structural executability rather than semantic consistency with the instruction. A trajectory may therefore be fully executable while still producing an incorrect final edit. This distinction motivates our separate evaluation of structural validity and final-state correctness.

OTIO Compilation. Text2Cut uses standard OTIO primitives to represent timeline-level structures whenever possible, directly mapping tracks, clips, media references, transitions, and temporal ranges to the OTIO hierarchy. Effect-level transformations, however, are not uniformly supported by standard OTIO. For such operations, Text2Cut materializes the transformed media as a derived asset, registers it in the shared asset pool, and updates the corresponding media reference in the timeline. The associated operation, target, and parameters are preserved in OTIO metadata, allowing previously applied effects to remain revisable: subsequent instructions can update their stored parameters and re-materialize the corresponding assets when necessary. After each edit, the updated timeline is represented again within the same OTIO-based abstraction, so subsequent operations depend on a software-agnostic editing state rather than editor-specific project states or effect implementations.

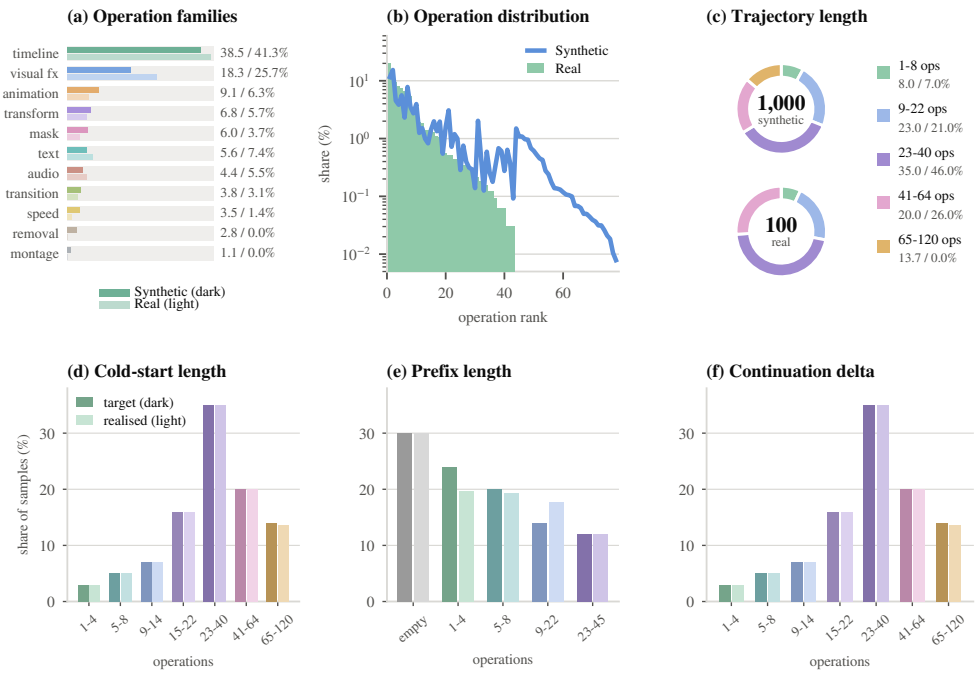

Figure 3: Composition of the training corpus and sampling fidelity. (a) Operation-family distributions of synthetic and real trajectories. (b) Per-operation distributions over the 78-operation space. (c) Trajectory-length distributions of the two subsets. (d–f) Target and realized distributions for cold-start trajectory length, continuation-prefix length, and continuation-delta length.

Once editing terminates, the final timeline state S_T, together with its operation metadata and asset references, is serialized into the output OTIO representation. This preserves OTIO as the portable timeline abstraction while extending it to executable and revisable effect-level editing.

4 Experiments

4.1 Dataset Construction for Operation Planning

We construct an instruction–operation corpus that pairs natural-language editing requests with executable operation trajectories. The training corpus contains 1,000 synthetic trajectories and 100 trajectories collected from human editing projects. The two sources serve complementary purposes: real trajectories capture how editing operations are composed in practice, whereas synthetic trajectories provide controlled coverage of the complete 78-operation space. Each sample consists of a natural-language instruction, an asset pool, an optional existing timeline, and a target operation trajectory. Fig. 3 summarizes the operation distributions, trajectory complexity, and sampling characteristics of the resulting corpus.

Real Editing Trajectories. We retain the original operation sequences from human editing projects without pruning, preserving their operation frequencies, ordering patterns, and long-range dependencies. As shown in Fig. 3(a,b), real editing trajectories are highly non-uniform: a small number of frequently used operation families account for most editing steps, while only 43 of the 78 operations appear in the collected projects. The trajectories are also strongly compositional, with long operation sequences occurring frequently in practical workflows (Fig. 3(c)). We use these empirical statistics to guide the construction of the synthetic corpus, rather than specifying its operation frequencies and trajectory complexity entirely by hand.

Synthetic Trajectories. We generate 1,000 synthetic trajectories using a state-aware sampler designed to balance realistic editing statistics, broad operation coverage, and structural validity. The

synthetic corpus contains 300 cold-start samples, which begin from an empty timeline and require generation of a complete editing trajectory, and 700 continuation samples, which begin from independently generated valid prefix timelines and predict subsequent operations conditioned on the resulting state. Continuation prefixes contain up to 47 preceding operations, exposing the planner to varying amounts of editing history. Across both settings, target trajectory lengths follow the complexity profile observed in real editing workflows, spanning 1–123 operations with a median of 32 (Fig. 3(c–f)). In additional, at each sampling step, only operations whose preconditions are satisfied by the current timeline state are eligible. A hierarchical deficit-based sampling scheme encourages the synthetic distribution to follow operation patterns observed in real projects while maintaining coverage of operations that are rare or absent in the collected trajectories. This ensures that all 78 operations are represented. Each sampled trajectory is subsequently replayed through the deterministic executor, and trajectories containing invalid references, unsatisfied dependencies, illegal state transitions, or mismatched final states are discarded. The synthetic corpus therefore extends operation coverage while preserving executability and realistic compositional complexity.

Instruction Generation. For each trajectory, we generate a corresponding natural-language instruction through a fidelity-oriented pipeline. A deterministic renderer first converts the trajectory into an explicit textual description that preserves all operation targets and parameters. GPT-5.6-Luna (OpenAI, 2026) then rewrites this description into natural editing language, while an automatic validator verifies that trajectory-specific values are preserved. Failed rewrites are retried, and samples that cannot be rewritten faithfully retain their deterministic descriptions. This procedure prioritizes semantic consistency between the instruction and its executable target over paraphrase diversity.

4.2 Experimental Setup

Evaluation Sets. We evaluate Text2Cut on both a controlled synthetic benchmark and held-out editing projects created by human editors. **SynCut-200** contains 200 samples evenly divided into four subsets: Cold-easy, Cold-hard, Cont-easy, and Cont-hard, with 50 samples per subset. Cold-start cases begin from the initial asset pool, whereas continuation cases additionally provide an existing timeline produced by preceding edits. Within each setting, easy and hard cases are separated according to target trajectory length. SynCut-200 covers all 78 operations and contains no target trajectory appearing in the training corpus. We further evaluate on **RealCut-50**, which contains 50 held-out human editing projects: 30 regular projects (Easy-30) and 20 more challenging production projects (Hard-20) with longer trajectories and deeper compositional dependencies.

Baselines. Text2Cut-4B is obtained by fine-tuning Qwen3-4B-Instruct (Yang et al., 2025). We compare against open-source and API-based language models, including Gemma-3-1B-it (Team et al., 2024), SmolLM3-3B (Bakouch et al., 2025), Qwen3-4B-Instruct (Yang et al., 2025), Mistral-7B-OpenOrca (Lian et al., 2023), Qwen3.7-Plus, Gemini-2.5-Flash (Comanici et al., 2025), and GPT-4o-mini (OpenAI, 2024). Open-source models are evaluated zero-shot, while API-based models are additionally evaluated with four in-context demonstrations that collectively cover all 78 operations.

Evaluation Protocol and Metrics. For each sample, all models receive the same natural-language instruction, asset pool, and initial editing state, including the existing timeline for continuation cases, and predict an operation trajectory. We replay each predicted trajectory using the same deterministic executor and validator used throughout our framework and report two complementary execution-based metrics: structural validity (M_V) and final-state correctness (M_S). M_V measures whether the entire predicted trajectory can be executed without violating entity, temporal, asset, or operation-specific constraints, whereas M_S measures whether executing the trajectory produces the same final timeline state as the ground truth. We treat M_S as the primary metric because an executable trajectory can remain semantically incorrect. Unlike sequence-level matching, this protocol evaluates the behavior of the generated editing program rather than its surface-form similarity to a reference trajectory.

Implementation Details. We train Text2Cut-4B on the 1,100 instruction–operation pairs described in Sec. 4.1 using supervised fine-tuning with LoRA. We use a LoRA rank of 64 with

Method	Model	Open	Cold-easy		Cold-hard		Cont-easy		Cont-hard		Overall	
			M_V	M_S	M_V	M_S	M_V	M_S	M_V	M_S	M_V	M_S
Zero-shot	Gemma-3-1B-it	✓	0.000	0.000	0.000	0.000	0.020	0.000	0.000	0.000	0.005	0.000
	SmolLM3-3B	✓	0.040	0.040	0.000	0.000	0.080	0.040	0.000	0.000	0.030	0.020
	Qwen3-4B-Instruct	✓	0.560	0.540	0.060	0.020	0.420	0.160	0.020	0.000	0.265	0.180
	Mistral-7B-OpenOrca	✓	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	Qwen3.7-Plus	✗	0.800	0.700	0.140	0.100	0.820	0.560	0.320	0.180	0.520	0.385
	Gemini-2.5-Flash	✗	0.980	0.760	0.600	0.100	0.840	0.620	0.560	0.160	0.745	0.410
	GPT-4o-Mini	✗	0.680	0.560	0.060	0.020	0.440	0.220	0.040	0.000	0.305	0.200
Few-shot	Qwen3.7-Plus	✗	1.000	0.960	0.880	0.180	0.860	0.700	0.900	0.200	0.910	0.510
	Gemini-2.5-Flash	✗	0.980	0.940	0.860	0.300	0.940	0.680	0.800	0.160	0.895	0.520
	GPT-4o-Mini	✗	0.780	0.720	0.100	0.040	0.520	0.360	0.060	0.000	0.365	0.280
Ours	Text2Cut-4B	✓	1.000	**0.960**	0.900	**0.460**	0.980	**0.840**	0.960	**0.400**	0.960	**0.665**

Table 1: Execution-based evaluation on **SynCut-200**, the 200-sample synthetic benchmark (50 samples per cold-start/continuation $\times$ easy/hard cell). M_V measures structural validity (executability); M_S measures final-state correctness and serves as the primary metric.

$\alpha = 128$, a learning rate of 2×10^{-4}, and train for 3 epochs with an effective batch size of 32. Training is performed on a single NVIDIA H200 80GB GPU.

4.3 Main Results

We compare Text2Cut with general-purpose language models on SynCut-200 and RealCut-50. Across both benchmarks, the results reveal a substantial gap between generating structurally executable trajectories and reaching the correct final editing state.

Synthetic Benchmark. Table 1 reports results on SynCut-200. General-purpose language models perform well on relatively simple cases, but final-state correctness degrades sharply as trajectory complexity increases. For example, few-shot Qwen3.7-Plus achieves $M_V = 1.000$ and $M_S = 0.960$ on Cold-easy, whereas on Cold-hard its M_S falls to 0.180 despite retaining $M_V = 0.880$. A similar discrepancy appears for Gemini-2.5-Flash, which achieves $M_V = 0.860$ but only $M_S = 0.300$ on Cold-hard. These results show that satisfying the executable interface does not necessarily imply that the correct editing state is reached. In comparison, Text2Cut achieves $M_V = 0.960$ and $M_S = 0.665$ overall without in-context demonstrations, improving final-state correctness over the strongest few-shot baseline (0.520). The difference is particularly pronounced on the hard subsets: Text2Cut obtains $M_S = 0.460$ on Cold-hard and 0.400 on Cont-hard, compared with the best few-shot baseline results of 0.300 and 0.200, respectively. These results suggest that operation-level supervision helps the model learn not only the structural constraints of the executable interface, but also the cross-step dependencies required to recover the intended final editing state.

Real Editing Projects. Table 2 evaluates generalization to held-out projects created by human editors. The discrepancy between structural validity and final-state correctness remains substantial in this setting. For example, few-shot Gemini-2.5-Flash reaches an overall structural validity of $M_V = 0.959$, but only $M_S = 0.309$ final-state correctness. Text2Cut achieves $M_S = 0.533$ on Easy-30, outperforming all evaluated baselines on this split. On the more challenging Hard-20 subset, Text2Cut reaches $M_V = 0.750$ and $M_S = 0.400$; few-shot Qwen3.7-Plus obtains a higher $M_S = 0.500$, indicating that long production trajectories remain challenging. Nevertheless, across RealCut-50, Text2Cut achieves the highest overall final-state correctness, with $M_S = 0.467$ without in-context demonstrations, compared with 0.434 for the strongest few-shot baseline. These results indicate that the operation semantics learned by Text2Cut transfer beyond the synthetic training distribution to editing trajectories constructed by human editors.

Qualitative Analysis. Table 3 presents a representative Cold-hard example involving beat-based montage construction, masking, timeline manipulation, transition editing, and dependencies on intermediate assets. Although the baseline predictions are often locally plausible, they fail for different execution-level reasons. Gemini-2.5-Flash generates a semantically plausible but unsupported operation, while GPT-4o-mini applies a timeline operation using an incorrect asset-level addressing

Method	Model	Open	Easy-30		Hard-20		Overall	
			M_V	M_S	M_V	M_S	M_V	M_S
Zero-shot	Gemma-3-1B-it	✓	0.000	0.000	0.000	0.000	0.000	0.000
	SmolLM3-3B	✓	0.000	0.000	0.000	0.000	0.000	0.000
	Qwen3-4B-Instruct	✓	0.000	0.000	0.050	0.000	0.020	0.000
	Mistral-7B-OpenOrca	✓	0.000	0.000	0.000	0.000	0.000	0.000
	Qwen3.7-Plus	✗	0.400	0.100	0.250	0.150	0.325	0.125
	Gemini-2.5-Flash	✗	0.867	0.367	0.750	0.350	0.809	0.359
	GPT-4o-Mini	✗	0.067	0.000	0.300	0.000	0.184	0.000
Few-shot	Qwen3.7-Plus	✗	0.733	0.367	0.950	**0.500**	0.842	0.434
	Gemini-2.5-Flash	✗	0.967	0.267	0.950	0.350	0.959	0.309
	GPT-4o-Mini	✗	0.067	0.000	0.200	0.000	0.134	0.000
Ours	Text2Cut-4B	✓	0.733	**0.533**	0.750	0.400	0.740	**0.467**

Table 2: Execution-based evaluation on **RealCut-50**, consisting of fifty held-out real editing projects: **Easy-30** and **Hard-20**.

Instruction:	Create a beat-cut montage using `audio_01`, starting at beat 2, with 4 beats per clip and a maximum duration of 11.37 seconds. Place it on `V2` ... Apply a circular mask to the background created for that montage ... Split the 1st clip on `V1` at 4.28 seconds. Add a linear wipe transition on `V1` ... delete the linear wipe transition on the 2nd clip on `V1` ...

GT (Ours):	`{"op_type":"detect_beats","target_asset_id":"audio_01", "bake_id":"detect_beats_1"},`
	`{"op_type":"insert_background","bake_id":"insert_background_1",...},`
	`{"op_type":"beat_cut_montage","params":{"beat_markers_id":"detect_beats_1", "placeholder_asset_id":"insert_background_1",...}},`
	`{"op_type":"mask_circle","target_asset_id":"insert_background_1", "bake_id":"mask_circle_1",...},`
	`{"op_type":"split_clip","params":{"track":"V1","clip_index":0,...}},`
	`{"op_type":"transition_linear_wipe",...},`
	`{"op_type":"delete_transition_linear_wipe","params":{"track":"V1",...}}`

Gemini-2.5-Flash:	... `{"op_type":"beat_montage","target_asset_id":"audio_01",...}` *operation not in the interface*
GPT-4o-mini:	... `{"op_type":"split_clip","target_asset_id":"insert_background_1",...}` *timeline operation addressed by asset*
Qwen3.7-Plus:	... `{"op_type":"audio_add_bgm","target_asset_id":"video_17",...}` *`bake_id` omitted; the later `transform` has no product to target*

Table 3: Cold-hard comparison of model outputs. Text2Cut matches the ground-truth canonical operations, while the baselines produce semantically plausible but execution-level incorrect programs.

scheme. Qwen3.7-Plus omits an intermediate identifier required by a subsequent operation, breaking the dependency chain. These errors illustrate that semantic plausibility alone is insufficient for non-linear editing: a single incorrect operation type, reference, or dependency can propagate through the trajectory and alter the resulting timeline. In contrast, Text2Cut preserves the canonical operation sequence, addressing conventions, and intermediate dependencies throughout the trajectory, allowing execution to recover the intended final state. This example further demonstrates why structural validity alone is insufficient: even an executable trajectory may still produce an incorrect result because of an incorrect operation, target reference, parameter, or improperly propagated intermediate state.

	Gemini-2.5-Flash (primary)	**Qwen3.7-Plus** (secondary)
Text2Cut preferred	24 / 44	29 / 44
Agentic baseline preferred	20 / 44	15 / 44

Table 4: VLM-as-judge comparison of rendered videos on 22 held-out real projects (44 judgments per judge: 22 projects $\times$ 2 A/B orderings). Judges receive only the editing brief and anonymized, temporally ordered keyframe strips from the two rendered videos.

VLM-as-Judge Comparison on Rendered Videos. Beyond state-level metrics, we further compare the *rendered videos* produced by Text2Cut and an agentic baseline, where GPT-5.6-Luna operates Codex through the ChatCut interface, using a VLM-as-judge protocol. We evaluate the 22 held-out real projects for which the zero-shot Text2Cut trajectories are state-equivalent to the human-edited reference timelines, allowing the rendered-video comparison to focus on the quality of the realized edits rather than differences in target state. Because the agentic baseline directly manipulates the editing interface without producing an operation trajectory, the operation-level metrics M_V and M_S are not directly applicable to it. For each project, both systems receive the same natural-language editing brief and asset library, and the judge is shown the brief together with temporally ordered keyframe strips uniformly sampled from the two anonymized renders (10 frames per video, or 16 for videos longer than 20 s). The judge evaluates (i) *brief adherence*, including whether the requested clips, ordering, effects, text overlays, and temporal structure are correctly realized, and (ii) *visual coherence*. To reduce positional bias, each pair is evaluated twice with the A/B ordering reversed. We use Gemini-2.5-Flash as the primary judge and Qwen3.7-Plus as a secondary judge for cross-validation; candidate judges that consistently preferred the same display position rather than the same underlying video on a pilot pair were excluded. As shown in Table 4, Gemini-2.5-Flash prefers Text2Cut in 24 of 44 ordered comparisons, while Qwen3.7-Plus prefers Text2Cut in 29 of 44.

5 Limitations

Text2Cut currently operates over a fixed space of 78 executable non-linear editing operations. While this space covers a broad range of timeline- and effect-level edits, highly specialized or editor-specific effects require additional operation definitions and execution modules. In addition, effect-level transformations that are not natively represented by OTIO are materialized as derived media assets, which introduces additional rendering and storage overhead, particularly for long or high-resolution videos. While OTIO provides a software-agnostic interchange layer, our current evaluation focuses on the Text2Cut execution environment rather than end-to-end interoperability across commercial editing systems, whose support ultimately depends on their OTIO adapters and effect implementations. Future work will focus on expanding the operation space, improving materialization efficiency, and strengthening interoperability with production editing systems.

6 Conclusion

We present **Text2Cut**, a language-driven framework for executable non-linear video editing. Text2Cut translates natural-language editing requests into atomic timeline- and effect-level operations, executes and validates them over structured editing states, and compiles the resulting edits into a portable OTIO-based representation. We train the planner on a hybrid corpus of synthetic trajectories and real editing projects. Experiments on controlled benchmarks and held-out real-world editing workflows show that Text2Cut achieves substantially higher final-state correctness while maintaining strong structural validity compared with general-purpose language-model baselines. These results highlight structured operation modeling and execution-grounded supervision as a promising foundation for practical language-driven video editing.

AI USE STATEMENT

We used generative AI primarily to improve the manuscript's language and clarity, and also to assist with drafting text. Generative models are additionally part of the released dataset, the benchmark, and the evaluation protocol, as described in Sec. 4.1. We used an LLM to generate the natural-language supervision of the corpus: GPT-5.6-Luna rewrites the deterministically rendered, parameter-complete description of each operation trajectory into natural editing language, under an automatic fidelity validator that rejects any rewrite losing a trajectory-specific value; trajectories that cannot be rewritten faithfully retain their explicit descriptions. The same pipeline produced the instructions of the held-out benchmark and the in-context examples used for the few-shot baselines. The operation trajectories themselves are not model-generated: they are either recorded from human editing projects or sampled by a state-aware program and verified by execution. For the comparison on rendered videos, Gemini-2.5-Flash serves as the primary judge and Qwen3.7-Plus as a secondary judge, each evaluating anonymized keyframe strips under both A/B orderings, whereas the state-level metrics (M_V, M_S) are computed by a deterministic executor and involve no model judgment. The authors take responsibility for the final manuscript, implementation, data, and reported results, including content produced with AI assistance.

ETHICS STATEMENT

This work is intended to support user-directed video editing: the model proposes an executable operation plan that a human can inspect, modify, or discard before anything is rendered. The dataset was produced for this project: the editing projects were commissioned from professional video editors whom we engaged and compensated for this purpose, and the footage they worked with is either original material or openly licensed stock, so the corpus carries no third-party copyright encumbrance and we can release it in full. The editors were informed that their projects would be used to build a public research dataset. The synthetic sampler's priors are measured from those same projects, so the notion of a "correct" edit in this work reflects the stylistic conventions of the editors we worked with rather than a universal standard of editing quality. Because the system automates the composition of existing footage, it inherits the misuse potential of editing tools in general — selective recomposition can change what a recording appears to show — and we therefore emphasize that every plan remains fully auditable as an explicit, human-readable operation list.

REPRODUCIBILITY STATEMENT

Implementation details, operation definitions, training settings, and evaluation protocols are provided in the main paper and appendix to support reproducibility.

REFERENCES

Adobe. Adobe premiere pro. https://www.adobe.com/products/premiere.html, 2026.

Apple. Final cut pro. https://www.apple.com/final-cut-pro/, 2026.

Elie Bakouch, Loubna Ben Allal, Anton Lozhkov, Nouamane Tazi, Lewis Tunstall, Carlos Miguel Patiño, Edward Beeching, Aymeric Roucher, Aksel Joonas Reedi, Quentin Gallouédec, et al. Smollm3: smol, multilingual, long-context reasoner. *Hugging Face Blog*, 2025.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Blackmagic Design. Davinci resolve. https://www.blackmagicdesign.com/products/davinciresolve, 2026.

Academy Software Foundation. Opentimelineio. https://github.com/AcademySoftwareFoundation/OpenTimelineIO, 2026.

Yujia Hu, Songhua Liu, Zhenxiong Tan, Xingyi Yang, and Xinchao Wang. Image editing as programs with diffusion models. In *NeurIPS*, 2025.

Wing Lian, Bleys Goodson, Guan Wang, E Pentland, A Cook, C Vong, and "Teknium". Mistralorca: Mistral-7b model instruct-tuned on filtered openorcav1 gpt-4 dataset. *HuggingFace Repository*, 2023.

Eyal Molad, Eliahu Horwitz, Dani Valevski, Alex Rav-Acha Acha, Yossi Matias, Yael Pritch, Yaniv Leviathan, and Yedid Hoshen. Dreamix: Video diffusion models are general video editors. *arXiv preprint arXiv:2302.01329*, 2023.

OpenAI. GPT-4o mini: Advancing Cost-Efficient Intelligence. `https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/`, 2024. Accessed: 2026-09-21.

OpenAI. GPT-5.6 System Card. `https://deploymentsafety.openai.com/gpt-5-6`, 2026. Accessed: 2026-09-21.

Alejandro Pardo, Jui-Hsien Wang, Bernard Ghanem, Josef Sivic, Bryan Russell, and Fabian Caba Heilbron. Generative timelines for instructed visual assembly. *arXiv preprint arXiv:2411.12293*, 2024.

Tianyuan Qu, Lei Ke, Xiaohang Zhan, Longxiang Tang, Yuqi Liu, Bohao Peng, Bei Yu, Dong Yu, and Jiaya Jia. Replan: Reasoning-guided region planning for complex instruction-based image editing, 2025.

Vipul Raheja, Dhruv Kumar, Ryan Koo, and Dongyeop Kang. Coedit: Text editing by task-specific instruction tuning. In *EMNLP*, pp. 5274–5291, 2023.

Marcelo Sandoval-Castaneda, Bryan Russell, Josef Sivic, Gregory Shakhnarovich, and Fabian Caba Heilbron. Editduet: A multi-agent system for video non-linear editing. In *SIGGRAPH*, pp. 1–11, 2025.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

Yael Vinker, Tamar Rott Shaham, Kristine Zheng, Alex Zhao, Judith E Fan, and Antonio Torralba. Sketchagent: Language-driven sequential sketch generation. In *CVPR*, pp. 23355–23368, 2025.

Jay Zhangjie Wu, Yixiao Ge, Xintao Wang, Stan Weixian Lei, Yuchao Gu, Yufei Shi, Wynne Hsu, Ying Shan, Xiaohu Qie, and Mike Zheng Shou. Tune-a-video: One-shot tuning of image diffusion models for text-to-video generation. In *ICCV*, pp. 7623–7633, 2023.

Qiucheng Wu, Jing Shi, Simon Jenni, Kushal Kafle, Tianyu Wang, Shiyu Chang, and Handong Zhao. Retouchiq: Mllm agents for instruction-based image retouching with generalist reward. In *CVPR*, pp. 12279–12288, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Shoubin Yu, Difan Liu, Ziqiao Ma, Yicong Hong, Yang Zhou, Hao Tan, Joyce Chai, and Mohit Bansal. Veggie: Instructional editing and reasoning video concepts with grounded generation. In *ICCV*, pp. 15147–15158, 2025.

Yiming Zeng, Wanhao Yu, Zexin Li, Tao Ren, Yu Ma, Jinghan Cao, Xiyan Chen, and Tingting Yu. Bridging the editing gap in llms: Fineedit for precise and targeted text modifications. *EMNLP*, pp. 2193–2206, 2025.

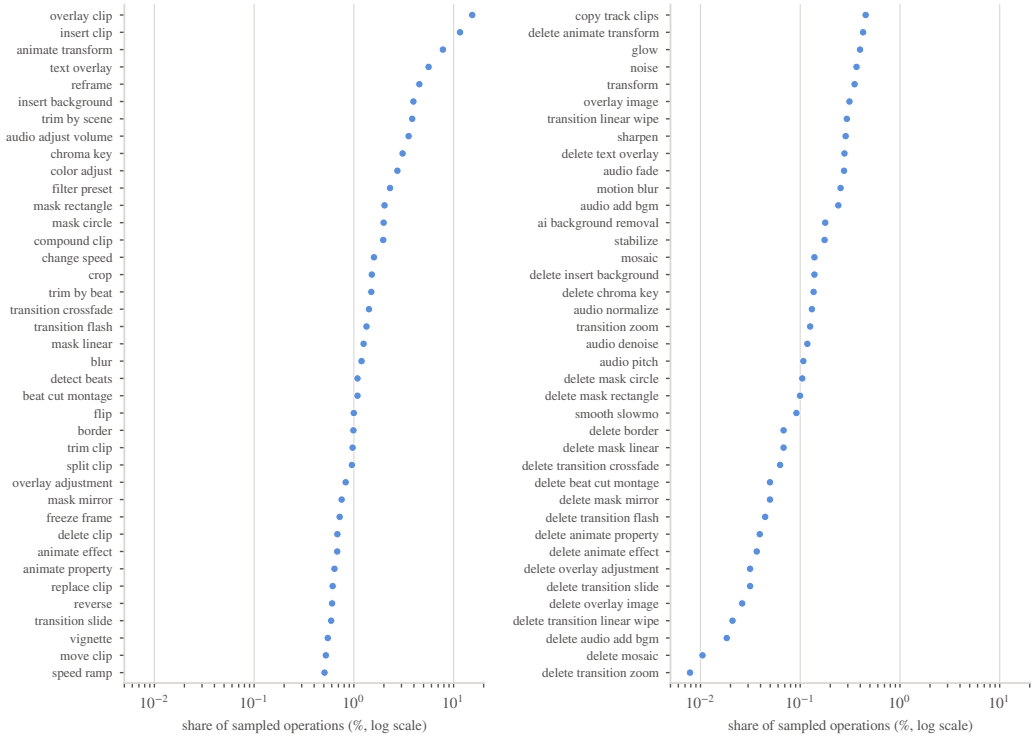

Figure 4: Per-operation distribution of the synthetic corpus on a logarithmic scale, covering 38,183 sampled operation steps. All 78 operations are instantiated, while their frequencies remain strongly long-tailed, ranging from `overlay clip` at 15.4% to `delete transition zoom` at 0.008%.

A Dataset Construction Details

A.1 Synthetic Trajectory Sampling

We provide additional details of the synthetic trajectory generation procedure introduced in Sec. 4.1. We factorize operation sampling as

$$P(o) = P(f)P(o \mid f), \tag{4}$$

where f denotes an operation family and o denotes an individual operation within that family.

To balance realistic editing statistics with broad operation coverage, we construct the target sampling distribution as a mixture of three components: the empirical distribution observed in real editing trajectories, the distribution accumulated in the synthetic corpus, and a uniform distribution over the operation space, with weights $0.80/0.10/0.10$, respectively. The empirical component preserves the dominant operation patterns observed in human editing workflows, while the uniform component assigns non-zero probability to operations that are rare or absent from the collected projects. For reversible constructs, 5% of the corresponding probability mass is allocated to their removal operations.

Sampling is both state-aware and deficit-driven. Before each step, operations whose preconditions are not satisfied by the current timeline state are removed from the candidate set. Among the remaining valid operations, the sampler prioritizes family–operation pairs that are underrepresented relative to their target frequencies. The selected operation is instantiated with valid parameters and immediately applied to the current editing state. Structural validity therefore takes precedence over exact distribution matching: when an operation is invalid under the current state, it is deferred rather than enabled by artificially inserting prerequisite operations.

The sampler also preserves locally repeated editing patterns observed in human workflows. Real editing trajectories frequently contain consecutive placement or processing operations, including repeated transformations of previously generated products. Approximately 40% of the editing steps in our collected projects belong to such local operation runs. We therefore allow valid local chains to continue rather than always selecting the globally most underrepresented operation. As shown in Fig. 4, all 78 operations are instantiated in the resulting synthetic corpus, while their frequencies remain strongly long-tailed and span approximately three orders of magnitude.

Cold-start and Continuation Samples. The synthetic corpus contains 300 cold-start samples and 700 continuation samples. Cold-start samples begin from an empty timeline and require prediction of the complete editing trajectory. Continuation samples instead begin from independently generated valid prefixes and require prediction of subsequent operations conditioned on the resulting timeline state. Prefixes for continuation samples are generated independently and are never reused from cold-start trajectories.

Across the complete synthetic corpus, empty, 1–4, 5–8, 9–22, and 23–45 operation prefixes account for 30%, 24%, 20%, 14%, and 12% of the samples, respectively. Realized non-empty prefixes span 1–47 operations, with a median length of 8. Cold-start targets and continuation deltas follow the same target-length distribution: 1–4 operations (3%), 5–8 (5%), 9–14 (7%), 15–22 (16%), 23–40 (35%), 41–64 (20%), and 65–120 (14%). The realized target trajectories span 1–123 operations, with a median length of 32 in both settings. Fig. 3(d–f) compares the target and realized distributions.

Trajectory Verification. Each retained sample contains an asset pool, an optional prefix timeline, a target operation sequence, and the intermediate products generated during execution. We deterministically replay every target trajectory using the same execution logic employed during evaluation. A trajectory is rejected if replay reveals invalid identifiers, unsatisfied dependencies, inconsistent asset references, illegal timeline transitions, or a mismatch between the recorded and replayed final states.

We additionally remove product identifiers that are never referenced by subsequent operations. Because such identifiers have no observable effect on downstream execution, retaining them would require the model to reproduce arbitrary naming choices that are irrelevant to the resulting editing state. This normalization preserves the execution semantics of the target trajectory while avoiding supervision on inconsequential identifier assignments.

A.2 Instruction Generation

Natural-language instructions are generated only after their corresponding executable trajectories have passed the verification procedure described above. For each trajectory, a deterministic renderer first produces a parameter-complete textual description containing all operation targets, numeric parameters, track references, enumerated settings, interpolation curves, and relevant default values. For cold-start samples, the description covers the complete target trajectory; for continuation samples, it describes only the newly requested operations, while the preceding editing history is supplied separately through the input timeline state.

GPT-5.6-Luna is then used to rewrite the deterministic description into more natural editing language. To prevent semantic information from being lost during rewriting, we apply an occurrence-aware fidelity validator. In addition to verifying the presence of each trajectory-specific value, the validator checks its required number of occurrences. Thus, when the same value is associated with multiple operations, the rewritten instruction must preserve the corresponding number of mentions rather than merely containing that value once.

Failed rewrites are retried for up to four attempts. If no faithful rewrite is obtained, we retain the deterministic parameter-complete description rather than introducing a semantically incomplete instruction. This fallback is used for 55 of the 1,100 training samples. Across the complete corpus, all 261,036 trajectory-specific values required by the executable targets are preserved in their corresponding instructions. Fig. 5 illustrates the complete construction pipeline, from a verified executable trajectory to deterministic rendering and the final validated natural-language instruction.

Stage 1 — Validated Trajectory (model target)


```
{"steps":[
 {"op_type":"insert_clip","target_asset_id":"sample_000081_video_02",
   "params":{"asset_id":"sample_000081_video_02","track":"V1","start":0.0,
                          "in_point":2.33,"duration":1.56}},
 {"op_type":"border","target_asset_id":"sample_000081_video_02","bake_id":"border_1",
   "params":{"color":"white","thickness":5}},
 {"op_type":"animate_transform","target_asset_id":"border_1",
   "params":{"translate":[{"time":0.0,"value":[0,0],"interp":"linear"},
                          {"time":4.0,"value":[0,0],"interp":"ease_out"}],
     "scale":[{"time":0.0,"value":1.0,"interp":"ease_in_out"},
              {"time":4.0,"value":1.24,"interp":"ease_in_out"}],
     "anchor":[0.5,0.5],"bg_color":"#000000"}}]}
```


Stage 2 — Deterministic Rendering (parameter-complete, including defaults)

1. Insert clip `sample_000081_video_02`: track=`V1`, start=0, in_point=2.33, duration=1.56. 2. Border on `sample_000081_video_02`: color=`white`, thickness=5. 3. Animate transform on the output of step 2: translate=(0, 0) at 0s (`linear`) -> (0, 0) at 4s (`ease_out`), scale=1 at 0s (`ease_in_out`) -> 1.24 at 4s (`ease_in_out`), anchor=[0.5, 0.5], bg_color=`#000000`.

Stage 3 — Validated Natural-Language Instruction (released)

Insert clip `sample_000081_video_02` on `V1`, starting at 0s, with an in-point of 2.33s and a duration of 1.56s. Add a `white` border to `sample_000081_video_02` with a thickness of 5. Then animate the transform on the output of the border step: translate it from (0, 0) at 0s with `linear` interpolation to (0, 0) at 4s with `ease_out` interpolation, and scale it from 1 at 0s with `ease_in_out` interpolation to 1.24 at 4s with `ease_in_out` interpolation. Set the anchor to [0.5, 0.5] and the background color to `#000000`.

Figure 5: Example of the three-stage instruction-generation pipeline. A verified executable trajectory is first converted into a deterministic, parameter-complete description and then rewritten into natural editing language. The occurrence-aware fidelity validator verifies both the presence and multiplicity of trajectory-specific values. In this example, the values 2.33, 1.56, and 1.24, together with both occurrences of `ease in out`, are preserved in the final instruction. Intermediate products are referred to naturally, e.g., "the output of the border step," rather than by internal product identifiers.

B Additional Qualitative Results

Tables 5, 6, and 7 provide additional examples of execution-level errors that are difficult to identify from surface-level semantic similarity alone, including unsupported operations, incorrect target addressing, inconsistent product identifiers, and confusion between semantically related but operationally distinct editing primitives.

C A Worked Example of Operation Planning

Fig. 6 traces a complete project from a natural-language editing brief to the final rendered video. The input consists of a 4,757-character brief and an asset pool containing ten video clips and audio, with no existing timeline, intermediate feedback, or in-context demonstrations. In a single zero-shot generation, Text2Cut produces the complete 20-operation plan. The trajectory includes four `mask circle` operations with keyframed reveal-radius animations, construction of a black background, and a sequence of `overlay clip` operations that place the masked products and ink footage on separate tracks with specified start times and scales.

Operations that cannot be represented directly as portable timeline edits are executed to produce derived media assets. Fig. 6 illustrates two such intermediate products, `c1_bake` and `ink_video_bake`, together with sixteen additional products generated throughout the trajectory. Each derived product is assigned an addressable identifier, allowing subsequent operations to transform, mask, or composite outputs produced by earlier steps. This explicit propagation of intermediate products enables the long dependency chains required by complex editing workflows.

The final row compares Text2Cut with the agentic baseline at three timestamps. Text2Cut reproduces the editor's staged reveal of the calligraphy panels over the ink background, whereas the baseline,



Figure 6: End-to-end example on project 0001, from a natural-language brief to the rendered video. Given only the brief and asset pool, Text2Cut generates the complete 20-operation trajectory in a single zero-shot pass. The deterministic executor materializes intermediate products as needed and renders the final timeline, which matches the human editor's result.

given the same brief and assets, places unmasked clips at incorrect scales and does not preserve the intended staging. Sec. D describes the protocol used to evaluate such rendered-video comparisons.

D VLM-AS-JUDGE EVALUATION PROTOCOL

Fig. 7 summarizes the evaluation protocol underlying the rendered-video preference results reported in the main paper.

Matched Inputs. For each project, Text2Cut and the agentic baseline receive exactly the same natural-language editing brief and asset pool. Text2Cut generates a complete operation trajectory zero-shot in a single pass, which is rendered by the deterministic executor. The baseline uses Codex operated by GPT-5.6-Luna as a multi-turn coding agent over the ChatCut interface and renders the resulting edit through ChatCut.

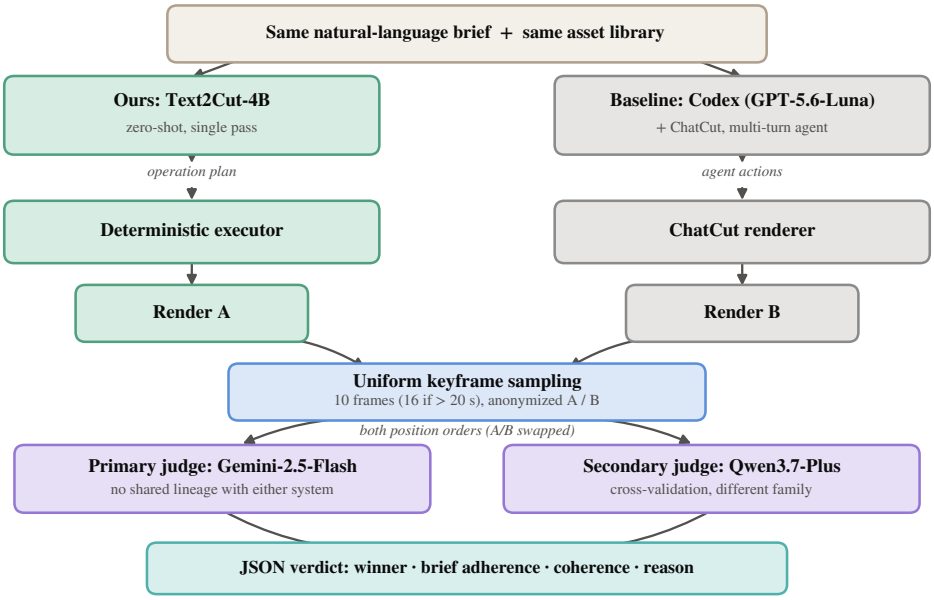

Figure 7: VLM-as-judge protocol for comparing Text2Cut with the agentic baseline. Both systems receive the same editing brief and asset pool. Their rendered outputs are uniformly sampled into anonymized keyframe strips and evaluated under both A/B orderings by two judge models, which return structured preferences for brief adherence and visual coherence.

Judge Input. The judge receives the original editing brief followed by two temporally ordered keyframe strips uniformly sampled from the rendered videos. We sample 10 frames per video, increasing to 16 frames for videos longer than 20 s, and resize all frames to 360p. The two strips are anonymized as A and B; the judge is not shown the system identities, the human reference render, or any state-level evaluation metrics. We intentionally withhold the human reference render so that the rendered-video comparison remains independent of the state-level ground truth used elsewhere in our evaluation.

Verdict Format. Each judgment returns a structured JSON response containing an overall preference and two sub-verdicts: *brief adherence*, which evaluates whether the requested clips, ordering, effects, text overlays, masks, and temporal structure are realized, and *visual coherence*, which evaluates whether the resulting composition remains visually consistent, without broken or frozen segments or illegible overlays. The judge additionally provides a short free-text rationale.

Position-Bias Control. Each video pair is evaluated twice with the A/B ordering reversed. A project is considered position-consistent for a judge when both orderings prefer the same underlying video. Candidate judges are first screened on a pilot comparison, and models that consistently select the same display position after the videos are swapped are excluded. We use Gemini-2.5-Flash as the primary judge and Qwen3.7-Plus as a secondary judge for cross-validation.

D.1 A Judging Round in Detail

Fig. 8 illustrates one complete judging round using the logged data for project 0001. The judge receives only the original editing brief, two anonymized keyframe strips labelled A and B, and the evaluation rubric; system identities, the human reference render, and state-level metrics are withheld.

The same pair is evaluated under both A/B orderings. In the first ordering, the Text2Cut render occupies slot A and the judge returns `"winner": "A"`. After swapping the videos, Text2Cut occupies slot B and the judge returns `"winner": "B"`. Although the raw labels differ, both

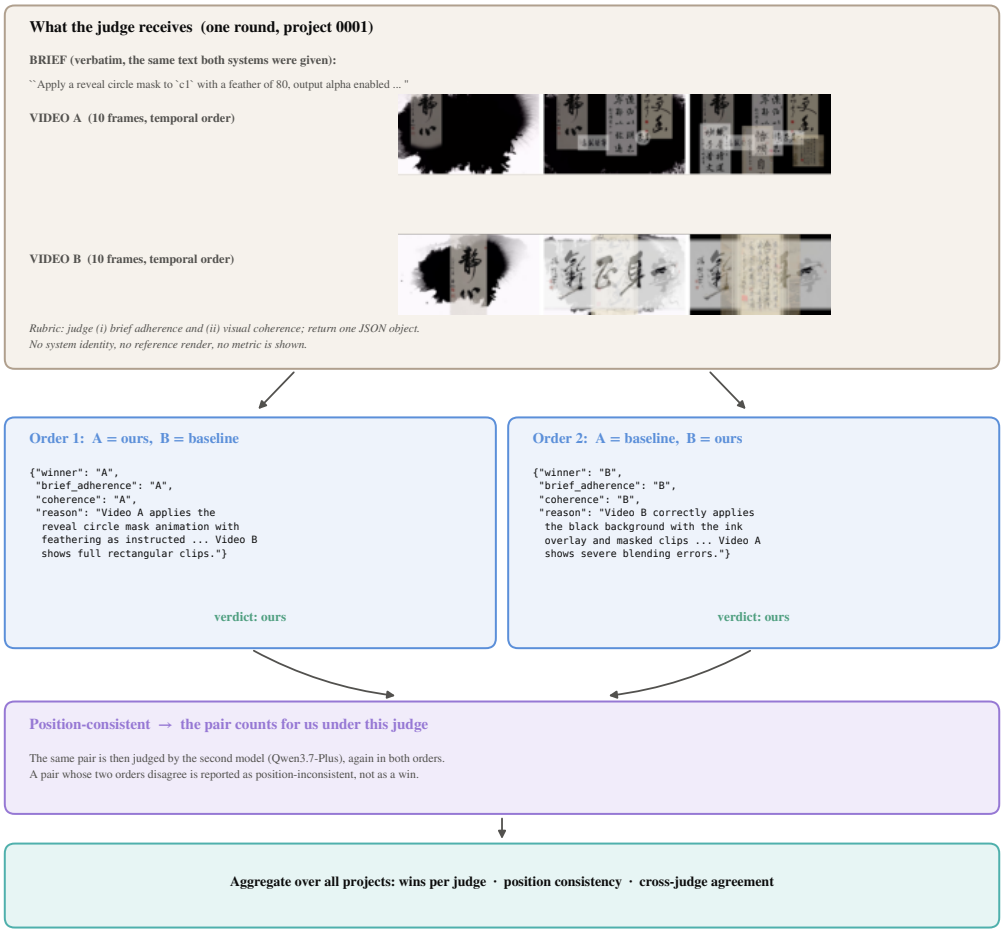

Figure 8: One judging round, with the actual prompt and verdicts recorded for project 0001. The judge receives the brief, two anonymized keyframe strips and the rubric, and returns a JSON verdict; the same pair is re-judged with the two videos swapped. Here both orders name our render, so the pair is position-consistent.

judgments prefer the same underlying video, so the project is recorded as position-consistent for this judge. Conversely, if both orderings selected the same display position, the resulting preferences would correspond to different videos and the project would be recorded as position-inconsistent rather than as a consistent preference for either system.

The same two-order evaluation is independently repeated with the secondary judge. Across projects, we report the number of preferences for each system, A/B-swap consistency for each judge, and cross-judge agreement, exposing the reliability of the preference measurement alongside the aggregate result.

Instruction:	Create a beat-cut montage using `audio_01`, starting at beat 2, with 4 beats per clip and a maximum duration of 11.37 seconds. Place it on `V2` ... Apply a circular mask to the background created for that montage ... Split the 1st clip on `V1` at 4.28 seconds. Add a linear wipe transition on `V1` ... Delete the linear wipe transition on the 2nd clip on `V1` ...

GT (Ours):	`{"op_type":"detect_beats","target_asset_id":"audio_01", "bake_id":"detect_beats_1"},`
	`{"op_type":"insert_background","bake_id":"insert_background_1",...},`
	`{"op_type":"beat_cut_montage","params":{"beat_markers_id":"detect_beats_1", "placeholder_asset_id":"insert_background_1",...}},`
	`{"op_type":"mask_circle","target_asset_id":"insert_background_1", "bake_id":"mask_circle_1",...},`
	`{"op_type":"split_clip","params":{"track":"V1","clip_index":0,...}},`
	`{"op_type":"transition_linear_wipe",...},`
	`{"op_type":"delete_transition_linear_wipe", "params":{"track":"V1",...}}`

Gemini-2.5-Flash:	`...`
	`{"op_type":"beat_montage","target_asset_id":"audio_01",...}`
	operation not defined in the interface

GPT-4o-mini:	`...`
	`{"op_type":"split_clip","target_asset_id":"insert_background_1",...}`
	timeline-level operation incorrectly addressed by asset ID

Qwen3.7-Plus:	`...`
	`{"op_type":"audio_add_bgm","target_asset_id":"video_17",...}`
	`bake_id` omitted, leaving the later `transform` without a valid product target

Table 5: Challenging cold-start example. Text2Cut reproduces the canonical reference trajectory, whereas the baselines exhibit distinct execution-level failures: an unsupported operation, incorrect asset-level addressing of a timeline operation, and a missing product identifier required by a subsequent operation.

Instruction: **Prior timeline:**	Create a white background at 1280 by 720 for 4.38 seconds at 30 fps. Then animate `blur_radius` on the result of that step from 0 at 0s with `ease_out` to 6 at 4s with `ease_in_out`. After that, split the 4th clip on `V1` at 2.29 seconds ... `V1`: `audio_denoise_1`, `product_1`, `audio_denoise_1`; `V2`: `overlay_adjustment_1`

GT (Ours):	`{"op_type":"insert_background","bake_id":"insert_background_1",...},`
	`{"op_type":"animate_property", "target_asset_id":"insert_background_1", "bake_id":"animate_property_2",`
	`"params":{"property_name":"blur_radius",...}},`
	`{"op_type":"split_clip", "params":{"track":"V1","clip_index":3,...}}, ...`

Gemini-2.5-Flash:	`...`
	`..."bake_id":"animate_property_1" ...`
	`animate_property_1` already exists on the prior timeline

Qwen3.7-Plus:	`{"op_type":"animate_effect",..."effect_name":"blur_radius"...}` *effect animation used instead of the requested property animation*
GPT-4o-mini:	`{"op_type":"animate_effect",...}` *same substitution, causing replay to fail*

Table 6: Continuation example with four target operations. Correct execution requires new product identifiers to remain consistent with products already present on the input timeline. The baselines either introduce conflicting identifiers or substitute `animate effect` for the requested `animate property`, two semantically related but operationally distinct primitives.

Instruction:	Create a compound clip from the 4th clip on `V1`, the 5th clip on `V1`, the 1st clip on `V3`, and the 8th clip on `V3`, without keeping the background. Flip `overlay_image_1` with horizontal flipping off and vertical flipping on. Animate the brightness of the vertically flipped `overlay_image_1` from -0.1 at 0s with `ease_in` to 0.15 at 4s with linear interpolation . . .
Prior timeline:	`V1` (5 clips): `mask_linear_1` $\times 2$, `overlay_image_1` $\times 2$, `video_08`; `V3` (9 clips) . . .
GT (Ours):	`{"op_type":"compound_clip","bake_id":"compound_clip_2", "params":{"sources":["V1-4th","V1-5th","V3-1st","V3-8th"], "keep_bg":false}}, {"op_type":"flip","target_asset_id":"overlay_image_1", "bake_id":"flip_3",...}, {"op_type":"animate_property","target_asset_id":"flip_3", "bake_id":"animate_property_3", "params":{"property_name":"brightness",...}}, ...`

Gemini-2.5-Flash:	`...`
	 `..."bake_id":"compound_clip_1", "sources":[...,"V3-9th"]...`
	 product numbering restarted from 1, and the 8th clip was counted as the 9th

GPT-4o-mini:	`...`
	 `..."bake_id":"compound_clip_1" ...`
	 identifier collides with a product already present on the timeline

Qwen3.7-Plus:	`{"op_type":"animate_effect","target_asset_id":"flip_3",...}` *same operation substitution*

Table 7: Continuation example with ten target operations conditioned on an existing timeline containing five clips on V1 and nine clips on V3. Baseline failures arise primarily from incorrect interpretation of the current editing state, including conflicting product identifiers, incorrect clip-ordinal grounding, and substitution of `animate effect` for `animate property`.