Plans You Can Check: Verifier-Grounded Learning of an Open-Weight Planner for Executable Video-Editing

Anonymous ACL submission

Abstract

Practical video editing is not only pixel generation: an editor must turn a brief, a clip pool, music metadata, and hard constraints into an executable timeline. We study this decision layer as *executable video-editing planning* and introduce RefineCut, which, unlike workflow systems that wrap a prompted frontier model, trains a compact open-weight planner for it. The planner edits a typed timeline through structured patches that cover clip selection, trimming, ordering, transitions, duration and music alignment, and generated-placeholder slots; a deterministic verifier applies each patch and checks it against an explicit constraint ledger. Because editing has no single ground-truth repair, we do not imitate teachers directly: RefineCut replays every multi-teacher branch through the verifier and keeps verifier-best repairs as supervision. A second stage, RefineCut-Evo, lets the student score its own repairs with the verifier and a task rubric and trains on high-margin preference pairs, so the final 8B planner runs in a closed verifier loop with no teacher calls at inference. On RefineCut-Bench, which contains 3,578 tasks, 7,971 captioned clips, 499 music tracks, and explicit ledgers, verifier-replayed distillation lifts the planner from 0.620 to 0.858 Video-Editing Score, and RefineCut-Evo reaches 0.924, with statistically significant gains in hard-constraint satisfaction, duration control, convergence, and human-rated preview preference.[1]

1 Introduction

Real-world video editing is rarely about generating new pixels. Given a pool of existing footage, an editor decides which clips to keep, how to trim and order them, which transitions to use, and how to align cuts to music. These decisions are tightly constrained: the final cut must hit a target duration, retain or exclude specific clips, respect pacing, and stay synchronized to a soundtrack. The deliverable is a structured *edit plan*, executed afterwards by a downstream toolchain. Editing, in other words, is first a *constrained planning* problem and only later a rendering problem.

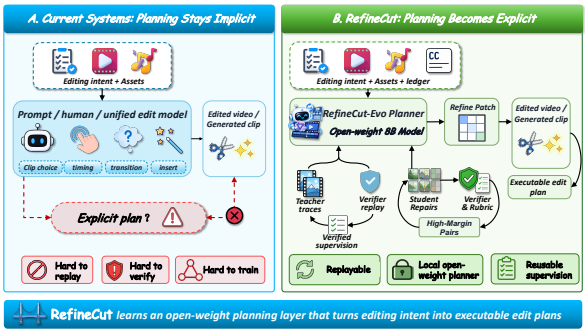

Figure 1: **From implicit to explicit planning.** RefineCut turns implicit prompted editing into explicit, verifier-trained executable planning.

Recent systems address the layer below this decision process or sidestep it entirely. Text-to-video and image-to-video models generate or transform pixels directly (Zhuang et al., 2024; Long et al., 2024), and instruction-editing frameworks modify content within a single shot (Yoon et al., 2025); none produces a clip-level plan over a real footage pool. Workflow-oriented mashup and dubbing systems (Li et al., 2026a; Zhang et al., 2025; Hu et al., 2024; Lin et al., 2026; Liang et al., 2025) do operate at the decision layer, but realize it by prompting a frozen frontier backbone inside a hand-built scaffold. The cost is real: the planning policy is never updated, so it cannot be optimized against the hard constraints a brief imposes, and is locked to a closed-source model. What the community lacks is a compact, open-weight planner that learns editing decisions and can sit in front of any rendering back-end, as shown in Figure 1.

We argue this gap is also a missed opportunity, because the editing decision layer has a property

[1] The training and inference code, together with RefineCut-Bench task records, constraint ledgers, captioned clips, and music tracks, will be released publicly upon acceptance.

open-ended generation lacks: its outputs can be checked. We formalize the layer as *executable video-editing planning*. A task instance gives the planner a brief, a real clip pool with captions and metadata, optional music with beat tracks, the current timeline state, and, crucially, an explicit *constraint ledger* enumerating what the final cut must satisfy. The planner emits a *RefinePatch*: an RFC 6902-style JSON Patch (Bryan and Nottingham, 2013) over a typed timeline state. Because every requirement is an explicit ledger entry, a *deterministic verifier*, rather than a learned LLM judge (Zheng et al., 2023), can apply each patch and recompute the ledger entry by entry, in the spirit of execution-based supervision in verifiable domains such as mathematical reasoning and code (Cobbe et al., 2021; Lightman et al., 2024; Lambert et al., 2025). This verifiability is what makes the planning layer learnable in a way pixel-level generation is not.

Turning this observation into a trained planner is still not a single-target imitation problem. The natural starting point is to distill trajectories from frontier models, since tool-use trajectories can supervise planner behavior (Zeng et al., 2024) and step-level feedback can turn successful and failed attempts into preference signals (Song et al., 2024b; Xiong et al., 2024). But editing has no single ground-truth repair: the same brief admits many valid cuts, so a teacher's first-choice branch is a guess, not a label, which is why preference-based learning, not imitation of one reference, is standard for open-ended generation (Christiano et al., 2017; Stiennon et al., 2020; Ouyang et al., 2022). A planner that imitates noisy teacher traces inherits their noise. It therefore needs feedback at every step, with each candidate branch replayed through the verifier before use, and, since teachers are themselves imperfect, a way to improve beyond any teacher, in the spirit of self-refinement and self-rewarding language models (Madaan et al., 2023; Yuan et al., 2024).

RefineCut is a two-stage framework built on this logic: editing decisions should be learned and verified, not prompted. Stage 1, the verifier-replayed teacher bootstrap, canonicalizes traces from three frontier API teachers, namely GPT-5.4[2], Qwen3-Max[3], and DeepSeek-V4-Pro (DeepSeek-AI, 2026), replays each candidate branch through the verifier, and keeps verifier-best branches as SFT targets and mixed-granularity preference pairs, yielding the Mixed-Pref 8B planner. Stage 2, RefineCut-Evo, performs verifier-centered, rubric-structured self-improvement: the student scores its own repair candidates with the verifier and a task-specific rubric (ER1–ER7), and high-margin pairs train it with Direct Preference Optimization (DPO) (Rafailov et al., 2023). Supervision thus shifts from external teachers to verifier-grounded self-improvement, and at inference the planner runs in a closed verifier loop with no teacher calls.

Contributions.

- We formulate *executable video-editing planning* and release RefineCut-Bench, a planning-level benchmark with real clip/music metadata, constraint ledgers, and multi-teacher trajectories.
- We propose *verifier-replayed trajectory distillation*: canonical normalization plus deterministic replay that converts noisy frontier traces into verified SFT targets and mixed-granularity preference pairs.
- We introduce RefineCut-Evo, an EvoLM-inspired (Li et al., 2026b) verifier-centered, rubric-structured self-improvement stage trained with DPO on verifier- and rubric-scored candidates.
- Experiments show consistent gains on hard-constraint satisfaction, duration regulation, and convergence (Section 4).

Positioning. RefineCut is closest to workflow-oriented video and multimodal creation systems such as DIRECT, LVAS-Agent, GLANCE, StoryAgent, and UniVA (Li et al., 2026a; Zhang et al., 2025; Lin et al., 2026; Hu et al., 2024; Liang et al., 2025), but those systems keep the editing policy inside prompted frontier-model workflows. It also builds on tool-executable planner learning and on self-improvement ideas from EvoLM and Rubric-Grounded RL (Li et al., 2026b; Bhattarai et al., 2026), while using a deterministic editing verifier rather than a general LLM judge as the primary training signal. Appendix A gives the full related-work discussion.

[2]Accessed via API in 2026; no public technical report at the time of writing.

[3]Qwen model-family page: https://huggingface.co/Qwen.

2 Task and Benchmark

This section formalizes planning-level video editing as a task, derives the evaluation principles used throughout the paper, and introduces RefineCut-Bench as the dataset, verifier, and multi-teacher trajectory resource that Section 3 turns into supervision.

2.1 Planning-Level Video Editing

RefineCut studies the *decision layer* of video editing: the layer that turns an editing intent and a pool of footage into an executable *edit plan*. The final video is rendered by a downstream toolchain, not by the planner, which never observes pixels and reads only textual clip captions and visual metadata from an upstream vision-language captioner, an interface with precedent in work that lets a language model reason over captioned content rather than raw frames (Zeng et al., 2023; Krishna et al., 2017).

Task definition. A task instance is a tuple

$$x = (b,\ C,\ M,\ s_0,\ L), \tag{1}$$

where b is a natural-language brief; C is a real clip pool, each clip carrying a caption and visual metadata (scene, motion intensity, camera duration); M is optional music metadata (beat times and energy); s_0 is the initial timeline state; and L is an explicit *constraint ledger* enumerating what the final cut must satisfy. A planner is a policy π_θ that at step t emits an edit patch $p_t \sim \pi_\theta(\cdot \mid x, s_t, L_t)$; a deterministic verifier applies it, advancing the state as $s_{t+1} = \mathrm{Apply}(s_t, p_t)$ and recomputing the ledger as $L_{t+1} = \mathrm{Verify}(s_{t+1})$. The objective is to reach a ledger-satisfying state within a small repair budget; the closed-loop execution is detailed in Section 3.

2.2 Inputs, Outputs, and Editing State

Clip pool and metadata. Clip captions and visual metadata are produced offline by an upstream vision-language captioner (Krishna et al., 2017; Wang et al., 2024); music metadata (beat times, energy) is extracted by a standard beat-tracking pipeline and supports music-synchronization constraints (Davis and Agrawala, 2018).

Editing state and edit artifact. The state s_t holds the selected clip sequence, segment durations, transition styles, music-sync parameters, generated placeholder slots, and per-entry ledger satisfaction. The planner edits it through a *RefinePatch*, an RFC 6902-style JSON Patch over the typed timeline state; a typed patch (rather than free-form text) is what makes each step machine-applicable and machine-checkable. When the brief calls for content absent from the pool, a placeholder slot records the requested prompt and position, with visual generation delegated downstream. Prompts use task-local clip aliases, so measured performance reflects planning ability rather than memorization of global clip identifiers. Schema details are in Appendix C.

Constraint ledger L. Each ledger entry is a tuple (*item_id*, *type*, *spec*, *satisfied*, *evidence*), spanning seven field-level constraint families: duration, transition, music_sync, clip inclusion, clip exclusion, repeat limit, and pacing; generated-placeholder requirements are represented as task-structure constraints checked by the same verifier (subtypes in Appendix B.4). The ledger makes editing requirements explicit and machine-checkable, in the spirit of verifiable instruction-following (Zhou et al., 2023; Jiang et al., 2024). An entry is a *hard constraint* when its *spec* admits a deterministic pass/fail test (e.g., a target duration within tolerance, a clip that must be kept or excluded), and HARDPASS is the event that every hard constraint holds simultaneously. Crucially, the verifier, not the planner, recomputes the *satisfied* field after every patch.

2.3 What Makes an Editing Plan Valid?

A valid edit plan must be executable, grounded in the given clips, satisfy the explicit constraints without breaking satisfied ones, and converge within a small repair budget. These requirements form a hierarchy, which our evaluation mirrors with three layers. The first layer, the execution gate, is a hard prerequisite covering schema validity, patch applicability, and timeline validity; a plan that fails it is not executable at all, mirroring execution-based evaluation in code generation (Jimenez et al., 2024). Conditioned on the gate, the second layer, planning quality, measures whether the plan is good in terms of constraint satisfaction, hard-constraint pass, clip grounding, duration and pacing, no-regression, and convergence. Both layers are decided entirely by the deterministic verifier, so every score is a checkable consequence of the ledger rather than a learned judgment, which is the evaluation-side expression of verified, not prompted. Finally, the third layer, human-rendered quality, is assessed after rendering and covers instruction fit, content relevance, tempo-

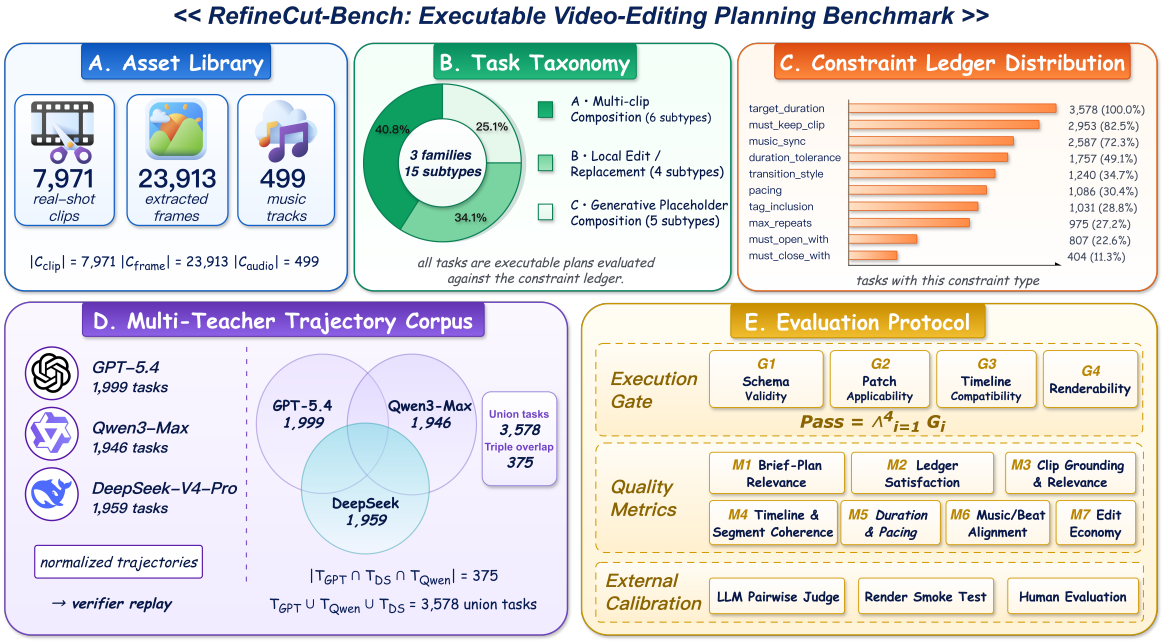

Figure 2: **RefineCut-Bench overview.** The benchmark ties real clip/music metadata, explicit ledger constraints, multi-teacher trajectories, and verifier-based evaluation into one planning-level protocol.

ral coherence, shot continuity, pacing, audio-visual rhythm, and overall quality, following established human-evaluation practice for edited video (Huang et al., 2024; Liu et al., 2024). The two families are complementary: verifier metrics test whether a plan executes under the ledger, human evaluation whether the rendered edit is perceptually preferred. We report component metrics with the VES summary; definitions and weights are in Section 4.2 and Appendix D.

2.4 RefineCut-Bench

RefineCut-Bench instantiates the task above as a controlled, executable-planning benchmark. Existing video-editing resources either evaluate rendered pixels or wrap a prompted frontier model in an agent scaffold whose decisions are never checked against an explicit specification; to our knowledge, RefineCut-Bench is the first to tie four ingredients into one planning-level release: real clip and music metadata, explicit constraint ledgers, multi-teacher trajectories, and a deterministic verifier-based evaluation protocol (Appendix B, Figure 2, Table 5).

Assets, tasks, and splits. The asset library draws from 7,971 captioned clips, 23,913 caption-anchored frames, and 499 music tracks. Tasks span three families, (A) multi-clip composition, (B) targeted insertion or replacement, and (C) generative composition with placeholder slots, totaling 15 subtypes and 3,578 canonical tasks. Raw records are deduplicated into canonical tasks and split at the record level. Because resampled variants of one task can otherwise straddle the train/test boundary, we mark a *canonical-clean* test subset whose canonical identifiers never appear in training, and report headline comparisons on it so that gains cannot be attributed to canonical-id overlap. Split sizes, per-subtype and per-constraint distributions, and the captioning pipeline are in Appendix B and Table 5.

2.5 Multi-Teacher Trajectories as Noisy Supervision

RefineCut-Bench also releases a multi-teacher *trajectory resource*, in the spirit of work that releases agent trajectories as a training signal rather than only an evaluation target (Zeng et al., 2024; Song et al., 2024a). For every training task we run one refinement rollout from each of three independent frontier API teachers (GPT-5.4, Qwen3-Max, and DeepSeek-V4-Pro); at each refine step a teacher emits several candidate branches, each a proposed *RefinePatch*, with its own first-choice flag (coverage and schema in Appendix B.6).

This resource is *noisy by construction*. Because editing admits no single ground-truth repair, teachers disagree on schema, branch structure, and which branch is best; a first-choice flag is a guess, not a label, and even one teacher's branches vary in executable quality. Multi-teacher distillation

that imitates such heterogeneous demonstrations directly would inherit their inconsistency (Tian et al., 2025). The constraint ledger offers a way out: every branch can be canonicalized and replayed through the deterministic verifier, which scores it against the same explicit ledger regardless of its source. Verifier replay thus converts an inherently noisy resource into a consistently graded supervision signal, the mechanism Section 3 builds on.

3 Method: Verifier-Replayed Distillation and Verifier-Centered Self-Improvement

3.1 Overview and Trajectory Collection

Our method has two offline learning stages and one runtime loop, shown in Figure 3. Stage 1, verifier-replayed teacher bootstrap, canonicalizes raw multi-teacher trajectories and replays every candidate branch through the verifier; verifier-best branches yield verified SFT targets and mixed-granularity preference pairs that produce an initial 8B planner, Mixed-Pref. Stage 2, RefineCut-Evo, samples $K{=}4$ student repairs on training states, scores them by verifier and rubric ER1–ER7, and trains with DPO on high-margin pairs. At runtime, the planner and verifier run the Apply/Verify closed loop defined in Section 2.1 for at most $T{=}3$ steps.

Each teacher-covered task receives one refinement rollout from each available frontier API teacher (GPT-5.4, Qwen3-Max, and DeepSeek-V4-Pro). Each refine step contains multiple candidate RefinePatch branches. The teacher's first-choice flag is recorded but not treated as ground truth: the verifier picks the training target. Per-teacher counts and the branch schema are in Appendix B.6.

3.2 Canonical Trajectory Normalization

Raw teacher outputs are heterogeneous: some wrap the task in an envelope, some flatten multi-step trajectories, and others nest branches as lists. We rewrite them into canonical patch trajectories keyed by (`task_id`, `teacher_id`), map JSON Pointer paths to a canonical namespace, and validate clip references against task-local aliases. Schema, path-alias table, and validation rules are in Appendix C.

3.3 Verifier Replay and Branch Arbitration

For every canonicalized step we keep up to four branches with distinct `repair_operators` and Jaccard-distinct clip selections, then replay each through the deterministic verifier. For branch b at state s_t, the verifier validates and applies the patch, recomputes the ledger L', and scores six signals: constraint-satisfaction change, targeted repair, required-clip recall, patch applicability, no-regression, and locality. The exact weighted definition is in Appendix D; we write the scalar score as $V(b)$. The verifier-best branch advances the state, and branch score gaps drive preference pairing. The full procedure is summarized in Appendix F, Algorithm 1.

3.4 Verifier-Replayed Distillation

Replay gives two supervision signals. Verified SFT retains the verifier-best branch as $p^\star$ when it canonicalizes and applies, repairs at least one violated ledger entry, regresses no satisfied constraint, and, for clip-grounding repairs, references a legitimate task-local alias. We fine-tune on these canonicalized patches with standard next-token cross-entropy. Mixed-granularity preference combines step-level pairs, in which the verifier-best branch is paired against a low-scoring branch separated by margin δ, with trajectory-level pairs ranked by terminal verifier score. We fine-tune the SFT checkpoint with offline DPO (Rafailov et al., 2023):

$$\begin{aligned} \mathcal{L}_{\text{DPO}} &= -\mathbb{E}_{(x,p^+,p^-)} \log \sigma(\beta z)\,, \\ z &= \Delta_\theta(p^+, x) - \Delta_\theta(p^-, x), \\ \Delta_\theta(p, x) &= \log \pi_\theta(p \mid x) - \log \pi_{\text{ref}}(p \mid x). \end{aligned} \tag{2}$$

Here p^+ and p^- are the preferred and rejected branches, π_{ref} is the frozen SFT checkpoint, and β controls deviation from it. Preferences come from executable verifier replay rather than LLM judgments; the resulting checkpoint is Mixed-Pref.

3.5 RefineCut-Evo: Verifier-Centered Student Self-Improvement

RefineCut-Evo improves the student with its own repairs scored by the verifier and a task-specific editing rubric, following recent rubric-guided self-improvement work (Li et al., 2026b; Bhattarai et al., 2026). We do not train a rubric generator.

Candidates and rubric. For each training state we sample $K{=}4$ RefinePatches from Mixed-Pref, parse and replay them, and filter schema or apply failures. The rubric ER1–ER7 covers *Intent*, *Ledger Satisfaction*, *Clip Grounding*, *Timeline Coherence*, *Duration/Pacing*, *Music/Beat*, and *Edit Economy*; per-family weights and deterministic fallback proxies are in Appendix F.

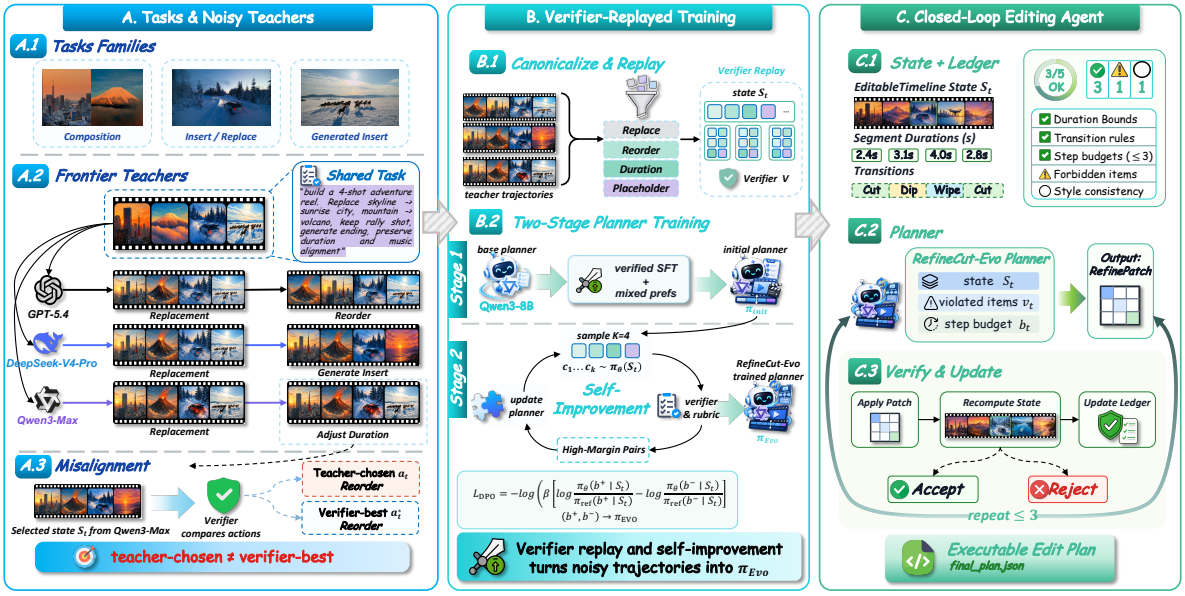

Figure 3: **Framework.** Noisy teacher trajectories become verified supervision; verifier- and rubric-scored student repairs drive self-improvement and closed-loop planning.

Joint scoring and training. Writing $R(c) = \sum_{m=1}^{7} \alpha_m r_m(c)$ with $\sum_m \alpha_m = 1$, each candidate receives $S(c) = \lambda V(c) + (1-\lambda)R(c)$, where $V(c)$ is the verifier score from Section 3.3 and $\lambda = 0.65$. At each state we choose

$$c^+ = \arg\max_{c \in \mathcal{C}(s)} S(c), \qquad c^- \in \{c \in \mathcal{C}(s) : S(c^+) - S(c) \geq \tau\}. \tag{3}$$

Thus c^- is a lower-scoring hard negative rather than the lowest sampled candidate. RefineCut-Evo starts from Mixed-Pref and trains with DPO on these pairs; hyperparameters and the dev100 selection rule are in Appendix E. Section 4.6 analyzes the verifier and rubric contributions.

3.6 Closed-loop Deployment

At test time the planner reads s_t and the still-violated ledger entries and emits one RefinePatch; the verifier validates, applies, and recomputes the ledger. No teacher is called at inference. The loop repeats for at most T=3 steps, after which the final state is dispatched to the editing toolchain. All training is offline, with no online RL or teacher interaction.

4 Experiments

We address four research questions (RQ1–RQ4), a robustness and failure analysis, and a rendered-video human evaluation.

4.1 Experimental Setup

Benchmark. We evaluate on Common-100 under the unified long PatchPlanner prompt; canonical-clean (N=92) is reserved for robustness, and dev100 only selects the RefineCut-Evo checkpoint (step 300) before any test measurement. The planner runs in a closed verifier loop with no teacher calls.

Loop horizon and candidate budget. The closed loop uses T=3 repair steps, matching teacher-trajectory collection (Appendix B.6) and the Converged@3 metric. During Evo training we sample K=4 student candidates per state, the teacher branch budget. Both are fixed design choices, not sweeps.

Models. All planners use Qwen3-8B-Instruct with LoRA (Hu et al., 2022). We compare Base, Raw, Verified, Traj-Pref, Mixed-Pref, and RefineCut-Evo; teacher trajectories are from Section 2.5. Hyperparameters, LoRA settings, and prompts are in Appendices E–I.

4.2 Evaluation Metrics

Automatic metrics evaluate executable planning, not rendered-video quality: schema validity, patch applicability, timeline validity, constraint satisfaction, hard-constraint pass, clip grounding, duration/pacing, no-regression, and convergence. Human evaluation (Section 4.8) covers perceptual edit quality. Video-Editing Score (VES) is a fixed weighted sum: constraint adherence receives 0.60

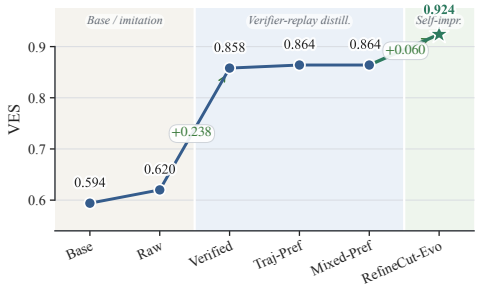

Figure 4: **Training-stage progression.** Common-100 VES across checkpoints; annotations mark the Raw→Verified and Mixed-Pref→Evo aggregate gains.

total weight (FCSR, HARD, REQCR), execution and duration checks 0.35 (PASR, DUR, TIMELINEVALIDITY), and NoRegression 0.05. Full definitions, the equation, and weight sensitivity are in Appendices D–G; tables abbreviate Converged@3 as CVG.

4.3 RQ1: Can an Open-Weight Planner Learn Executable Editing Planning?

Table 1 and Figure 4 show that an open-weight planner can learn executable editing planning. The gains are uneven: verifier-replayed SFT lifts VES from 0.620 to 0.858 (+0.238), mixed-granularity DPO reaches 0.864, and RefineCut-Evo adds +0.060 to 0.924. Thus most progress comes from making teacher traces executable, while Evo repairs residual hard cases. Its gain concentrates on HARDPASS (0.670→0.820), DUR@2S (0.830→0.980), and CONVERGED@3 (0.800→0.950); all are planning-layer gains.

	Constraint adherence			Loop quality		Overall
Model	**FCSR**	**Hard**	**ReqCR**	**Dur**	**Cvg**	**VES**
Base and naive imitation						
Base	0.608	0.150	0.273	0.660	0.210	0.594
Raw	0.559	0.160	0.723	0.440	0.250	0.620
Verifier replay distillation						
Verified	0.880	0.630	0.975	0.820	0.790	0.858
Traj-Pref	0.877	0.670	0.969	0.840	0.790	0.864
Mixed-Pref	0.882	0.670	0.969	0.830	0.800	0.864
Self-improvement (final)						
RefineCut-Evo⋆	**0.953**$_{+0.071}$	**0.820**$_{+0.150}$	**0.981**$_{+0.012}$	**0.980**$_{+0.150}$	**0.950**$_{+0.150}$	**0.924**$_{+0.060}$

Table 1: **Closed-loop planning results on Common-100.** Colored subscripts are aggregate changes relative to Mixed-Pref; ⋆ marks the final RefineCut checkpoint.

4.4 RQ2: Why is Verifier Replay Needed?

Verifier replay is needed because teacher branches are partial repairs, not solved plans. Each frontier teacher emits multiple candidate branches per refine step, but the raw traces differ in schema, branch structure, and executable quality. We canonicalize every candidate, apply it to the actual editing state, and re-score it with the deterministic verifier before using it as supervision. Table 2 reports the recomputed verifier-best branch diagnostics for the three teachers.

The verifier-best branches have high patch-apply success but low one-step HardPass and modest duration control. The replay signal therefore identifies the most useful branch for the current state rather than assuming any single branch solves the full ledger. The downstream effect is visible in Table 1: Raw imitation of teacher-selected branches reaches only VES 0.620, while verifier-best SFT reaches 0.858. Verifier replay thus serves both as a data-cleaning step and as the mechanism that turns heterogeneous teacher traces into learnable supervision. Extended components are in Appendix G, Table 10.

	Replay scale		Verifier components				
Teacher	N	**Branches**	**FCSR**	**Hard**	**PASR**	**ReqCR**	**VES**
GPT-5.4	2,000	6,000	0.379	0.015	0.872	0.698	0.521
Qwen3-Max	1,946	3,892	0.446	0.016	0.989	0.712	0.557
DeepSeek-V4-Pro	1,959	3,920	0.473	0.028	0.946	0.703	0.559

Table 2: **Verifier-best replay diagnostics by teacher.** Per-branch values are recomputed from replay logs; this diagnoses the teacher resource, not student closed-loop performance.

4.5 RQ3: Can RefineCut-Evo Self-Improve Beyond Mixed-Pref?

The step-300 RefineCut-Evo checkpoint, selected on dev100 (Appendix G, Table 11), reaches VES = 0.924 on Common-100, an aggregate +0.060 over Mixed-Pref. The gain is concentrated: Evo improves 22 tasks, ties 73, and degrades 5, as expected for a second stage targeting residual hard cases. The paired mean is +0.059 VES (95% CI $[0.028, 0.092]$, $p = 0.0015$), with the same direction on canonical-clean (+0.058, $p = 0.0025$) and against Verified and Traj-Pref (Appendix G, Table 12).

4.6 RQ4: What Drives RefineCut-Evo?

Table 3 isolates Evo scoring under the same backbone, candidate pool, and DPO recipe. Verifier-only DPO already recovers most of the gain; rubric scoring adds a consistent but modest +0.015 VES by shaping cleaner margins. Removing the margin filter costs more, and UCPO-lite (Lochab et al., 2026) underperforms, so verifier-scored student self-improvement remains the main signal.

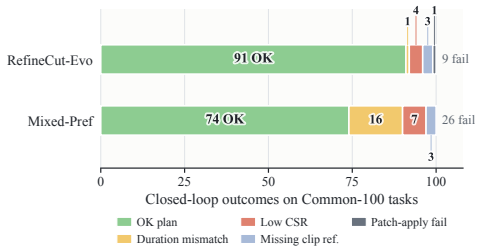

Figure 5: **Failure composition.** RefineCut-Evo increases OK plans from 74 to 91 and reduces duration mismatches from 16 to 1 on Common-100; the small residual tail shows all remaining closed-loop failure types.

	Component metrics				Overall
Variant	**FCSR**	**Hard**	**Dur**	**Cvg**	**VES**
Rubric-margin DPO	0.9530	0.820	0.980	0.950	**0.9237**
Verifier-only DPO	0.9395	0.780	0.940	0.900	0.9085 -0.015
No-margin DPO	0.9178	0.760	0.910	0.870	0.8965 -0.027
UCPO-lite DPO	0.9012	0.740	0.890	0.840	0.8800 -0.044

Table 3: **RefineCut-Evo ablation.** Rubric scoring gives a consistent but modest gain over verifier-only scoring; margin filtering also matters.

4.7 Robustness and Failure Analysis

Canonical-clean and failures. The Evo gain is not an artifact of canonical-id overlap. On the canonical-clean subset (N=92), RefineCut-Evo attains VES $= 0.917$ against 0.859 for Mixed-Pref ($+0.058$), and the variant ordering is preserved (Appendix G, Table 15). The per-task view shows that residual hard cases are converted into passing plans: OK plans rise from 74 to 91 and duration mismatches drop from 16 to 1 (Figure 5; full taxonomy in Appendix G, Table 14). The gain holds across all three task families (A $0.865 \rightarrow 0.892$, B $0.871 \rightarrow 0.975$, C $0.853 \rightarrow 0.890$) and on 50 human-written briefs, where the model ordering is preserved and the Evo-stage gap remains similar (Mixed-Pref 0.848 vs. RefineCut-Evo 0.902, $+0.054$; Appendix H, Table 20).

Reward-hacking and diversity. RefineCut-Evo does not exploit the verifier through trivial rewriting: over-rewrite does not increase, no-regression remains 1.0, duration deviation drops from 1.55 s to 0.34 s, and operation-count entropy rises to 0.99 instead of collapsing onto one operator.

LLM-judge audit. The earlier blind API-judge panel is kept only as a distillation-stage audit in Appendix G; Evo claims rest on verifier metrics, ablation, failure reduction, and human evaluation.

4.8 Rendered-Video Human Evaluation

We complement verifier metrics with blind randomized pairwise evaluation on rendered storyboard previews. Each comparison has 150 judged pairs and three annotators with one to five years of video-editing experience. RefineCut-Evo is preferred to Mixed-Pref (100W/34T/16L, preference 0.780, non-tie win rate 0.862) with substantial agreement ($\kappa = 0.620$, Krippendorff's $\alpha = 0.580$); dimensions and construct-validity details are in Appendix H.

	Sample	Preference			Agreement		
Comparison	N	**W/T/L**	**Pref.**	**WR**	**Agree**	κ	α
Evo vs Mixed	150	100/34/16	**0.780**	**0.862**	0.873	0.620	0.580
Evo vs Traj	150	95/37/18	0.757	0.841	0.860	0.590	0.553
Evo vs SFT	150	97/35/18	0.763	0.843	0.867	0.601	0.562
Mixed vs Base	150	124/18/8	0.887	0.939	0.927	0.781	0.742

Table 4: **Rendered-preview human evaluation.** Preference score is $(W + 0.5T)/N$; higher values prefer the first model. Human ratings complement automatic executable-planning checks by evaluating perceptual edit quality.

4.9 Summary of Findings

Verifier-replayed distillation accounts for most of the gain over the Raw baseline, while RefineCut-Evo adds a smaller, concentrated improvement by repairing residual hard cases. The Evo gain transfers to canonical-clean and human-written briefs, is significant under paired tests, and is driven mainly by verifier-scored self-improvement with rubric structure shaping margins. Human evaluation confirms that planning-layer gains survive rendering.

5 Conclusion

We studied *executable video-editing planning*: producing a verifiable edit plan over a typed timeline rather than generating pixels. RefineCut makes the editing decision flow explicit and learnable through two stages of verifier-grounded supervision. Verifier-replayed teacher bootstrap replays heterogeneous frontier-teacher branches through a deterministic verifier, turning noisy trajectories into verified SFT targets and preference pairs; RefineCut-Evo then improves the open-weight planner on its own repair candidates with high-margin DPO. Distillation supplies executable competence and self-improvement repairs the residual hard cases. The broader takeaway: for executable creative tasks, a deterministic verifier can serve as the primary training signal.

Limitations

Planning-layer scope. RefineCut is deliberately scoped to executable edit planning over a typed timeline. The verifier checks schema validity, patch applicability, ledger satisfaction, clip grounding, duration control, and no-regression; rendered-video human evaluation checks whether these planning gains are visible in perceived edit quality.

Upstream perception. The planner reads upstream captions, motion metadata, and music metadata rather than raw pixels, so caption or beat-tracking errors can bound edit-plan quality. Better metadata should transfer directly, but cross-source robustness remains future work.

Generalization and future work. The benchmark covers three task families, one primary asset pool, a 50-brief human-written validation set, and one 8B Qwen3 backbone with LoRA. RefineCut-Evo does not claim a complete EvoLM reproduction, online RL, or general video-editing SOTA; broader real-user deployments, asset domains, backbones, and longer-form editing tasks remain future work. Task briefs and constraint ledgers are LLM-generated under controlled templates: this enables reproducible verifier-based evaluation, but validating ledger realism against human-authored editing specs at scale remains future work.

Ethical Considerations

RefineCut studies planning-level editing rather than unrestricted video generation, but executable edit plans can connect to rendering or generation tools and could be misused to create misleading edits, omit context, or insert synthetic content without disclosure. Our benchmark uses controlled tasks, explicit ledgers, and clip-level metadata; deployment on real media should respect copyright, consent, provenance, and disclosure requirements. Clips come from public research-licensed sources and are not annotated with person identities; captions are short scene/action descriptions and do not contain personal names. The planner reads only captions and metadata, not raw pixels. The verifier checks structural correctness, not factual accuracy, fairness, or social appropriateness, so we view RefineCut as a research framework for verifier-guided planner training and encourage future work on provenance tracking, misuse detection, and oversight.

Declaration of Generative AI Usage

During preparation, the authors used AI assistants for language polishing, LaTeX formatting, consistency checking, and coding assistance. AI tools did not generate experimental results or automatically validate references. All claims, analyses, citations, and final decisions were manually verified and approved by the authors, who bear full responsibility for the manuscript.

References

Manish Bhattarai, Ismael Boureima, Nishath Rajiv Ranasinghe, Scott Pakin, and Dan O'Malley. 2026. Rubric-grounded RL: Structured judge rewards for generalizable reasoning. *Preprint*, arXiv:2605.08061.

Paul C. Bryan and Mark Nottingham. 2013. JavaScript Object Notation (JSON) patch. RFC 6902.

Jiaxin Cheng, Tianjun Xiao, and Tong He. 2024. Consistent video-to-video transfer using synthetic dataset. In *The Twelfth International Conference on Learning Representations*.

Paul F. Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, volume 30.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and Christopher Hesse. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

Abe Davis and Maneesh Agrawala. 2018. Visual rhythm and beat. *ACM Transactions on Graphics*, 37(4).

DeepSeek-AI. 2026. DeepSeek-V4: Towards highly efficient million-token context intelligence. Technical report and Hugging Face model card. Accessed 2026-05-26.

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. 2024. Model alignment as prospect theoretic optimization. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 12634–12651. PMLR.

Mohammad Gheshlaghi Azar, Zhaohan Daniel Guo, Bilal Piot, Remi Munos, Mark Rowland, Michal Valko, and Daniele Calandriello. 2024. A general theoretical paradigm to understand learning from human preferences. In *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*, volume 238 of *Proceedings of Machine Learning Research*, pages 4447–4455. PMLR.

Jiwoo Hong, Noah Lee, and James Thorne. 2024. ORPO: Monolithic preference optimization without reference model. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11170–11189, Miami, Florida, USA. Association for Computational Linguistics.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations*.

Panwen Hu, Jin Jiang, Jianqi Chen, Mingfei Han, Shengcai Liao, Xiaojun Chang, and Xiaodan Liang. 2024. StoryAgent: Customized storytelling video generation via multi-agent collaboration. *Preprint*, arXiv:2411.04925.

Ziqi Huang, Yinan He, Jiashuo Yu, Fan Zhang, Chenyang Si, Yuming Jiang, Yuanhan Zhang, Tianxing Wu, Qingyang Jin, Nattapol Chanpaisit, Yaohui Wang, Xinyuan Chen, Limin Wang, Dahua Lin, Yu Qiao, and Ziwei Liu. 2024. VBench: Comprehensive benchmark suite for video generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21807–21818.

Yuxin Jiang, Yufei Wang, Xingshan Zeng, Wanjun Zhong, Liangyou Li, Fei Mi, Lifeng Shang, Xin Jiang, Qun Liu, and Wei Wang. 2024. FollowBench: A multi-level fine-grained constraints following benchmark for large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4667–4688, Bangkok, Thailand. Association for Computational Linguistics.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can language models resolve real-world GitHub issues? In *The Twelfth International Conference on Learning Representations*.

Weijie Kong, Qi Tian, Zijian Zhang, Rox Min, Zuozhuo Dai, Jin Zhou, Jiangfeng Xiong, Xin Li, Bo Wu, Jianwei Zhang, and 1 others. 2024. HunyuanVideo: A systematic framework for large video generative models. *Preprint*, arXiv:2412.03603.

Ranjay Krishna, Kenji Hata, Frederic Ren, Li Fei-Fei, and Juan Carlos Niebles. 2017. Dense-captioning events in videos. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 706–715.

Max Ku, Cong Wei, Weiming Ren, Huan Yang, and Wenhu Chen. 2024. AnyV2V: A tuning-free framework for any video-to-video editing tasks. *Transactions on Machine Learning Research*.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, and 4 others. 2025. Tulu 3: Pushing frontiers in open language model post-training. In *The Second Conference on Language Modeling*.

Ke Li, Maoliang Li, Jialiang Chen, Jiayu Chen, Zihao Zheng, Shaoqi Wang, and Xiang Chen. 2026a. DIRECT: Video mashup creation via hierarchical multi-agent planning and intent-guided editing. *Preprint*, arXiv:2604.04875.

Shuyue Stella Li, Rui Xin, Teng Xiao, Yike Wang, Rulin Shao, Zoey Hao, Melanie Sclar, Sewoong Oh, Faeze Brahman, Pang Wei Koh, and Yulia Tsvetkov. 2026b. EvoLM: Self-evolving language models through co-evolved discriminative rubrics. *Preprint*, arXiv:2605.03871.

Zhengyang Liang, Daoan Zhang, Huichi Zhou, Rui Huang, Bobo Li, Yuechen Zhang, Shengqiong Wu, Xiaohan Wang, Jiebo Luo, Lizi Liao, and Hao Fei. 2025. UniVA: Universal video agent towards open-source next-generation video generalist. *Preprint*, arXiv:2511.08521.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

Zihao Lin, Haibo Wang, Zhiyang Xu, Siyao Dai, Huanjie Dong, Xiaohan Wang, Yolo Y. Tang, Yixin Wang, Qifan Wang, and Lifu Huang. 2026. GLANCE: A global-local coordination multi-agent framework for music-grounded non-linear video editing. *Preprint*, arXiv:2604.05076.

Yaofang Liu, Xiaodong Cun, Xuebo Liu, Xintao Wang, Yong Zhang, Haoxin Chen, Yang Liu, Tieyong Zeng, Raymond Chan, and Ying Shan. 2024. EvalCrafter: Benchmarking and evaluating large video generation models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22139–22149.

Anamika Lochab, Bolian Li, and Ruqi Zhang. 2026. Uniform-correct policy optimization: Breaking RLVR's indifference to diversity. *Preprint*, arXiv:2605.00365.

Fuchen Long, Zhaofan Qiu, Ting Yao, and Tao Mei. 2024. VideoStudio: Generating consistent-content and multi-scene videos. In *Computer Vision – ECCV 2024*, pages 468–485. Springer.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pages 46534–46594.

Yu Meng, Mengzhou Xia, and Danqi Chen. 2024. SimPO: Simple preference optimization with a reference-free reward. In *Advances in Neural Information Processing Systems*, volume 37.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744.

Adam Polyak, Amit Zohar, Andrew Brown, Andros Tjandra, Animesh Sinha, Ann Lee, Apoorv Vyas, Bowen Shi, Chih-Yao Ma, Ching-Yao Chuang, and 1 others. 2024. Movie Gen: A cast of media foundation models. *Preprint*, arXiv:2410.13720.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, pages 53728–53741.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R. Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36.

Yifan Song, Weimin Xiong, Xiutian Zhao, Dawei Zhu, Wenhao Wu, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. 2024a. AgentBank: Towards generalized LLM agents via fine-tuning on 50000+ interaction trajectories. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 2124–2141, Miami, Florida, USA. Association for Computational Linguistics.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024b. Trial and error: Exploration-based trajectory optimization of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7584–7600, Bangkok, Thailand. Association for Computational Linguistics.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F. Christiano. 2020. Learning to summarize with human feedback. In *Advances in Neural Information Processing Systems*, volume 33, pages 3008–3021.

Yijun Tian, Yikun Han, Xiusi Chen, Wei Wang, and Nitesh V. Chawla. 2025. Beyond answers: Transferring reasoning capabilities to smaller LLMs using multi-teacher knowledge distillation. In *Proceedings of the Eighteenth ACM International Conference on Web Search and Data Mining*, WSDM '25, pages 251–260. Association for Computing Machinery.

Team Wan, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, Jianyuan Zeng, Jiayu Wang, Jingfeng Zhang, Jingren Zhou, Jinkai Wang, Jixuan Chen, Kai Zhu, Kang Zhao, Keyu Yan, and 43 others. 2025. Wan: Open and advanced large-scale video generative models. *Preprint*, arXiv:2503.20314.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Yang Fan, Kai Dang, Mengfei Du, Xuancheng Ren, Rui Men, Dayiheng Liu, Chang Zhou, Jingren Zhou, and Junyang Lin. 2024. Qwen2-VL: Enhancing vision-language model's perception of the world at any resolution. *Preprint*, arXiv:2409.12191.

Renxi Wang, Xudong Han, Yixuan Zhang, Timothy Baldwin, and Haonan Li. 2025. NAT: Enhancing agent tuning with negative samples. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7385–7398, Albuquerque, New Mexico. Association for Computational Linguistics.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Xin Guo, Dingwen Yang, Chenyang Liao, Wei He, Songyang Gao, Lu Chen, Rui Zheng, Yicheng Zou, Tao Gui, Qi Zhang, Xipeng Qiu, Xuanjing Huang, Zuxuan Wu, and Yu-Gang Jiang. 2025. AgentGym: Evaluating and training large language model-based agents across diverse environments. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 27914–27961, Vienna, Austria. Association for Computational Linguistics.

Weimin Xiong, Yifan Song, Xiutian Zhao, Wenhao Wu, Xun Wang, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. 2024. Watch every step! LLM agent learning via iterative step-level process refinement. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1556–1572, Miami, Florida, USA. Association for Computational Linguistics.

Zhuoyi Yang, Jiayan Teng, Wendi Zheng, Ming Ding, Shiyu Huang, Jiazheng Xu, Yuanming Yang, Wenyi Hong, Xiaohan Zhang, Guanyu Feng, Da Yin, Yuxuan Zhang, Weihan Wang, Yean Cheng, Bin Xu, Xiaotao Gu, Yuxiao Dong, and Jie Tang. 2025. CogVideoX: Text-to-video diffusion models with an expert transformer. In *The Thirteenth International Conference on Learning Representations*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Jaehong Yoon, Shoubin Yu, and Mohit Bansal. 2025. RACCooN: Versatile instructional video editing with

auto-generated narratives. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 27972–28008, Suzhou, China. Association for Computational Linguistics.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason E. Weston. 2024. Self-rewarding language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 57905–57923. PMLR.

Andy Zeng, Maria Attarian, Brian Ichter, Krzysztof Marcin Choromanski, Adrian Wong, Stefan Welker, Federico Tombari, Aveek Purohit, Michael S. Ryoo, Vikas Sindhwani, Johnny Lee, Vincent Vanhoucke, and Pete Florence. 2023. Socratic models: Composing zero-shot multimodal reasoning with language. In *The Eleventh International Conference on Learning Representations*.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2024. AgentTuning: Enabling generalized agent abilities for LLMs. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3053–3077, Bangkok, Thailand. Association for Computational Linguistics.

Yehang Zhang, Xinli Xu, Xiaojie Xu, Doudou Zhang, Li Liu, and Ying-Cong Chen. 2025. Orchestrating audio: Multi-agent framework for long-video audio synthesis. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 22267–22282, Suzhou, China. Association for Computational Linguistics.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-Bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36, pages 46595–46623.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *Preprint*, arXiv:2311.07911.

Sidan Zhu, Yutong Wang, Hongteng Xu, and Dixin Luo. 2025. Weakly-supervised movie trailer generation driven by multi-modal semantic consistency. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, pages 10234–10242.

Shaobin Zhuang, Kunchang Li, Xinyuan Chen, Yaohui Wang, Ziwei Liu, Yu Qiao, and Yali Wang. 2024. Vlogger: Make your dream a vlog. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8806–8817.

A Related Work

A.1 AI-Assisted Video Editing and Generation

A first line of work focuses on creating or modifying visual content. Text-to-video and image-to-video models generate clips de novo (Zhuang et al., 2024; Long et al., 2024; Yang et al., 2025; Kong et al., 2024; Wan et al., 2025; Polyak et al., 2024), and instruction-based editing models change appearance or style within a fixed shot or short sequence (Cheng et al., 2024; Ku et al., 2024; Yoon et al., 2025). Trailer, montage, and mashup systems (Li et al., 2026a; Lin et al., 2026; Zhu et al., 2025) assemble cuts from a footage library according to a script or theme, and UniVA (Liang et al., 2025) packages generation, editing, and segmentation under one tool-calling loop. These systems can produce or rearrange content. However, they do not train a reusable open-weight planner: a planner that learns editing decisions over a real clip pool from a constraint ledger and an editable timeline.

A.2 Workflow Decomposition for Multimodal Creation

Hierarchical multi-agent systems break complex creation into staged professional roles. DIRECT (Li et al., 2026a) decomposes mashup creation into Screenwriter, Director, and Editor agents under a hierarchical multimodal coherency objective. LVAS-Agent (Zhang et al., 2025) mirrors a long-video dubbing studio with Storyboarder, Scriptwriter, Designer, and Generator. StoryAgent (Hu et al., 2024) takes a similar role-based view of storytelling video production. These systems share two properties: the planning logic is built from prompted frontier backbones, and the editing decision flow is encoded in the prompt scaffolding rather than learned. RefineCut keeps the workflow-decomposition view but inverts the operating mode. We collect trajectories from such workflows, replay them through a deterministic verifier, and train an open-weight planner that internalizes the editing decision flow.

A.3 Tool-Executable Planners from Trajectories

A complementary line of work studies agents that reason, act, or revise through tool-like trajectories. Prompt-time and self-correction methods such as ReAct (Yao et al., 2023), Self-Refine (Madaan et al., 2023), and Reflexion (Shinn et al., 2023) demonstrate iterative reasoning or feedback use without necessarily updating model weights. Training-oriented systems instead use interaction or tool-use trajectories to improve agent behavior: AgentTuning (Zeng et al., 2024) fine-tunes on tool-use traces, AgentBank (Song et al., 2024a) scales trajectory fine-tuning across diverse agent skills, and AgentGym (Xi et al., 2025) evolves agents across environments from collected trajectories. Trajectory feedback can also be turned into a preference signal: ETO (Song et al., 2024b) pairs successful and failed trajectories for Direct Preference Optimization (DPO) (Rafailov et al., 2023), IPR (Xiong et al., 2024) estimates step-level rewards by Monte-Carlo sampling along expert trajectories, and negative-aware training (Wang et al., 2025) incorporates failed trajectories during fine-tuning. RefineCut belongs to this broader executable-agent line, but differs by using deterministic replay of typed video-editing patches (Bryan and Nottingham, 2013) to construct both supervised fine-tuning (SFT) targets and preference pairs.

A.4 Rubric-Guided and Verifier-Guided Self-Improvement

Recent work explores rubric-structured self-improvement of language models. EvoLM (Li et al., 2026b) uses a co-trained rubric model to score self-generated candidates and drive student improvement. Rubric-Grounded RL (Bhattarai et al., 2026) applies multi-criterion weighted rewards, and UCPO-style diversity penalties (Lochab et al., 2026) target collapse in multiple-correct settings. RefineCut adapts this idea to executable video-editing planning. The student samples candidate repairs at training states, each candidate is scored by the deterministic verifier together with a task-specific editing rubric, and high-margin pairs train the planner with DPO. We do not train a rubric generator, do not run a complete EvoLM system, and do not adopt additional RL-style objectives in this paper; UCPO-lite is reported only as a negative diagnostic.

Preference-optimization objectives. RefineCut uses DPO because verifier replay and Evo candidate scoring naturally yield paired chosen/rejected repairs. Recent alternatives such as IPO (Gheshlaghi Azar et al., 2024), KTO (Ethayarajh et al., 2024), ORPO (Hong et al., 2024), and SimPO (Meng et al., 2024) modify the preference objective, reduce reference-model dependence, or

Item	Value
Raw video clips	7,971
Extracted caption frames	23,913
Music tracks	499
Raw task records	3,960
Canonical unique tasks	3,578
Train / Dev / Test (records)	2,773 / 596 / 591
Test canonical-clean	518 / 591
Removed by canonical overlap	73 (12.35%)
Common-100 ∩ canon.-clean	92
Task-level record overlap	0 / 0 / 0
Raw teacher files (GPT-5.4/Qwen3-Max/DeepSeek-V4-Pro)	2,000 / 1,962 / 1,962
Replayed teacher files (GPT-5.4/Qwen3-Max/DeepSeek-V4-Pro)	2,000 / 1,946 / 1,959
Raw canonical triple intersection	381
Replayed canonical triple intersection	375
Canonical teacher union	3,578

Table 5: Dataset, splits, and teacher coverage.

use unary/desirable–undesirable signals. These objectives are orthogonal to our central contribution: deterministic verifier replay creates the supervision source. We therefore use DPO as a standard paired-preference optimizer and leave objective swaps to future work.

B RefineCut-Bench Construction Details

This appendix lists the construction steps required to reproduce RefineCut-Bench: asset collection, captioning, task generation, ledger design, clip-pool sampling, teacher trajectory collection, and the canonical-id split. The main benchmark overview appears in Figure 2.

B.1 Asset Pool and Caption Sources

The asset pool contains 7,971 captioned clips drawn from five public sources (Table 6). Each row in the clip caption table carries the fields `clip_id`, `source`, `duration`, `subject`, `action`, `scene`, `camera`, `scene_category`, `motion_intensity`, `caption_short`, `model_used`, and `ts`. The `model_used` field records which caption model emitted each row; we do not assume a single global captioner and we do not claim a specific VLM-generated captioning provenance. The supplementary artifact records the exact `model_used` distribution.

B.2 Frame Sampling and Captioning

We use an adaptive frame sampler ("v2") that estimates motion intensity from optical-flow magnitude on a coarse temporal grid and then samples 3, 5, or 7 frames per clip for low / medium / high motion. The motion bucket is also stored in the `motion_intensity` field of the caption row, so that downstream task generation can use it as a metadata signal.

The captioning module is an upstream component: the planner does not read pixels and only sees the caption fields. We do not specify a single VLM provider in this paper; the exact captioner configuration is recorded per-row in `model_used` and is released with the dataset.

Source	Captioned clips
panda70m_sample	3,990
pexels	3,219
pixabay	255
openvid	100
long_video_segment	407
Total	7,971

Table 6: Captioned clip counts per source. The total matches the asset count reported in Section 2.4.

B.3 Task Generation

Tasks are generated by an LLM conditioned on a sampled clip pool, the family and subtype slot in Table 7, and a family-conditioned ledger template. The generated record separates task identity, the natural-language brief, planner inputs, and subtype-specific structure.

Record group	Fields
Identity	`task_id`, `task_type`, `task_subtype`
Brief	`brief`, `target_duration`
Planner inputs	`clip_pool`, `constraint_ledger`
Structure	`structure`, `subtype_constraints`

Task-record schema. The appendix names implementation fields here; the surrounding text uses plain-language descriptions.

Family	Subtypes
A (multi-clip composition)	themed_montage, pacing_progression, motion_dynamics_montage, scene_traversal, character_focus, abstract_rhythm
B (local edit/repair)	b_roll_insert, clip_swap, cut_extend, transition_repair
C (generative composition)	generated_opener, generated_bridge, generated_text_overlay, generated_replacement, generated_full_assembly

Table 7: The 15 task subtypes grouped by family.

B.4 Constraint Ledger Design

The constraint ledger is the explicit specification of what a successful cut has to satisfy. To keep the appendix readable, we group the 14 fine-grained ledger types by the editing requirement they express.

Requirement	Ledger types
Duration	`target_duration`, `duration_tolerance`
Transition	`transition_style`
Music sync	`music_sync_bpm`, `music_sync_beat`
Clip inclusion	`must_keep_clip`, `must_open_with` `tag_inclusion`
Clip exclusion	`must_exclude_clip`, `must_close_with` `tag_exclusion`
Repeat limit	`no_repeat_within_seconds` `max_repeats_per_clip`
Pacing	`pacing`

Ledger type groups. Fine-grained types are shown as implementation field names; the main text uses the broader requirement names.

Every ledger ships with at least four items, and every ledger includes a `target_duration` item. The seven field-level constraint families mentioned in Section 2.2 (duration, transition, music synchronization, clip inclusion, clip exclusion, repeat limit, and pacing) group these field-level checks; generated-placeholder requirements are checked separately as task-structure constraints. Per-type counts on the 3,578 canonical task universe are shown in Figure 2 (C); the full numeric table is released with the dataset.

B.5 Clip-Pool Sampling

The `clip_pool` of a task is sampled from the captioned asset pool under the following rules.

(i) The planner may only choose clips from the given `clip_pool`; references to a `clip_id` outside the pool are flagged as invalid by the verifier.

(ii) When the candidate pool (*given_pool*) contains at least 30 clips, the sampled `clip_pool` size is between 30 and 60; otherwise the entire candidate pool is used.

(iii) The ledger is generated jointly with the `clip_pool` so that every `must_keep_clip` and `must_exclude_clip` item refers to a clip that exists in the pool.

B.6 Teacher Trajectory Generation

For the teacher-covered subset of canonical tasks we collect a teacher trajectory from each available frontier API teacher (GPT-5.4, Qwen3-Max, and DeepSeek-V4-Pro); coverage is partial, and the per-teacher replayed counts and canonical triple intersection are reported in Table 5. At every refine step a teacher must emit *exactly four* candidate branches; each branch is a `RefinePatch` that carries: a list of `operations` (RFC 6902 add / remove / replace), a `rationale_against_ledger` naming the ledger items it tries to repair, a `repair_operator` label from the fixed vocabulary, a free-text `thought`, and a numeric `confidence`.

`repair_operator`	Intent
`modify_prompt`	rewrite the generative prompt
`change_tool`	switch the downstream tool slot
`adjust_parameter`	tweak a numeric or enum field
`reselect_clip`	replace a clip in the sequence
`retry_with_fallback`	redo with a safer default
`abort_and_skip`	drop the step

Table 8: The six `repair_operator` values.

B.7 Splits and Canonical IDs

Raw task records are deduplicated to canonical task ids by collapsing minor surface variants (resampled `clip_pools` of the same underlying task) into a single canonical id. We then split at the record level into train / dev / test of $2{,}773/596/591$. Of the 591 test records, 518 carry a canonical id that does not appear in the train set (*canonical-clean*); the remaining 73 records overlap by canonical id with train. The intersection of canonical-clean with the Common-100 evaluation subset is $N = 92$ records and is used for the robustness check in Section 4.7.

B.8 Licensing and Intended Use

All clip sources in Table 6 are publicly available and permit research use; the music tracks and a subset of in-house clips are private assets used for research only. We use every source within its stated terms and solely for non-commercial research. RefineCut-Bench (task records, ledgers, captioned-clip and music metadata, multi-teacher trajectories) and the training/inference code will be released under CC BY-NC 4.0 upon acceptance. Derived artifacts inherit the research-only restriction of their underlying sources and should not be used outside research contexts.

C Schemas and Interfaces

This appendix gives compact schemas for the five interfaces that appear throughout the paper. Full JSON Schemas and longer worked examples are in the supplementary release.

RefinePatch – a planner step

```
{
  "operations": [
    {"op": "replace",
     "path": "/args/selected_sequence",
     "value": ["C003","C001","C007","C012"]},
    {"op": "replace",
     "path": "/args/segment_durations",
     "value": [2.0, 3.5, 4.0, 3.5]}
  ],
  "rationale_against_ledger":
    ["target_duration#0", "must_keep_clip#2"],
  "repair_operator": "reselect_clip",
```

```
  "affected_clips": ["C001","C007","C012"],
  "thought": "Swap C005 -> C007 to keep duration.",
  "confidence": 0.71
}
```

EditPlan / Timeline IR – closed-loop terminal artifact

```
{
  "selected_sequence": ["C003","C001","C007","C012"],
  "segment_durations": [2.0, 3.5, 4.0, 3.5],
  "transitions": ["cut","cross_dissolve","cut"],
  "music_sync": {"bpm": 96, "beat_offset_ms": 120},
  "placeholders": [],
  "renderer_hints": {"target_fps": 30}
}
```

ConstraintLedger entry – one row

```
{
  "item_id": "target_duration#0",
  "type": "target_duration",
  "spec": {"value_sec": 13.0, "tolerance_sec": 2.0},
  "satisfied": false,
  "evidence": {"current_sec": 15.4}
}
```

VerifierOutput – one verifier call

```
{
  "schema_ok": true, "patch_apply_ok": true,
  "timeline_valid": true,
  "delta_csr": 0.20,
  "targeted_repair": true,
  "no_regression": true,
  "req_clip_recall": 1.0,
  "locality": 1.0,
  "score": 0.49
}
```

RubricScore – Evo rubric weighted aggregate ER1–ER7 (illustrative per-task instance)

```
{
  "ER1": 0.50, "ER2": 1.00, "ER3": 0.75,
  "ER4": 0.50, "ER5": 1.00, "ER6": 0.50, "ER7": 1.00,
  "weights":
    {"ER1":0.10,"ER2":0.25,"ER3":0.20,
     "ER4":0.10,"ER5":0.15,"ER6":0.05,"ER7":0.15},
  "total": 0.755,
  "fallback_used": ["ER1", "ER3_judge"]
}
```

D Metric Definitions

Execution Gate. SCR (schema-conformance rate): output parses against the RefinePatch JSON schema. PASR (patch-apply success rate): canonicalized patch applies cleanly to s_t. TIMELINEVALIDITY: the resulting state serializes to a Timeline IR that the renderer parser accepts.

Planning Quality. FINALCSR (FCSR; terminal constraint-satisfaction rate): terminal mean per-entry pass rate over the ledger. HARDPASS (Hard): $\mathbb{1}[\text{FINALCSR}=1]$. TARGETEDREPAIR: fraction of ledger items named in the patch's `rationale_against_ledger` that are satisfied after `Apply`. FINALREQCLIPRECALL (ReqCR): recall over the required clip ids on the terminal sequence (denominator rules in Appendix J). VALIDCLIPPRECISION: fraction of clip references in the terminal sequence that lie inside the pool. DURATIONPASS@2S (Dur): $\mathbb{1}[|\text{DUR}_{\text{final}} - \text{DUR}_{\text{target}}| \leq 2]$. NOREGRESSIONALLSTEPS: no previously satisfied ledger item becomes unsatisfied at any step. CONVERGED@3 (Cvg): $\mathbb{1}[\text{FINALCSR} \geq 0.8]$ within T=3 closed-loop steps (operational rule in Appendix J).

VES summary. VES is a fixed weighted sum:

$$\begin{aligned}\text{VES} = 0.30\,&\text{FinalCSR}\\ &+ 0.15\,\text{HardPass} + 0.15\,\text{PASR}\\ &+ 0.15\,\text{FinalReqClipRecall}\\ &+ 0.10\,\text{DurationPass}\\ &+ 0.10\,\text{TimelineValidity}\\ &+ 0.05\,\text{NoRegression}.\end{aligned}$$

Branch-score signals. The replay score of a single candidate branch combines six signals. ΔCSR is the increase in the per-entry ledger pass rate between the pre-patch state s_t and the post-patch state s_{t+1}. TARGETEDREPAIR, REQCLIPRECALL, PASR, and NOREGRESSION are the per-branch counterparts of the planning-quality metrics above, evaluated for one applied branch rather than the terminal state. LOCALITY is the fraction of patch operations that edit only fields tied to the ledger items named in `rationale_against_ledger`; it penalizes patches that modify state unrelated to the items being repaired.

Branch score weights. The replay branch score uses the six signals named in the method section:

$$\begin{aligned}V(b; s_t, L_t) = w_1&\Delta\text{CSR}\\ &+ w_2\text{TargetedRepair}\\ &+ w_3\text{ReqClipRecall}\\ &+ w_4\text{PASR}\\ &+ w_5\text{NoRegression}\\ &+ w_6\text{Locality}.\end{aligned} \tag{4}$$

It uses fixed weights $(w_1, w_2, w_3, w_4, w_5, w_6) = (0.35, 0.20, 0.20, 0.10, 0.10, 0.05)$ applied in order to ΔCSR, TARGETEDREPAIR, REQCLIPRECALL, PASR, NOREGRESSION, and LOCALITY.

Sensitivity (pre-Evo). An earlier sensitivity study on the pre-Evo planners (Base, Raw, Verified, Traj-Pref, Mixed-Pref) checked the ranking under CSR-heavy, clip-heavy, and execution-heavy reweightings of the VES summary and found the

relative ordering preserved. The corresponding table is released with the supplementary artifacts. We do not restate it here: RefineCut-Evo improves FCSR, HARD, and DUR, which together carry the largest weight among the VES components, so its ranking above Mixed-Pref is robust to any reweighting that keeps the same component set.

E Training Details

Backbone and LoRA. All planners initialize from Qwen3-8B-Instruct with LoRA (Hu et al., 2022). The LoRA configuration (rank, α, target modules) follows the default Qwen3-8B-Instruct LoRA recipe; the exact values and adapter checksums are released with the training artifacts.

Verified SFT. Examples 3,317 (one verifier-best target per replayed step that passes the verified-SFT filter; Section 3.4). Max steps 1,200. Learning rate 5e−5. Per-device batch 2 with gradient accumulation 8. Loss is standard next-token cross-entropy on the canonicalized patch.

Mixed-Pref DPO. Examples 2,380 chosen / rejected pairs, mixing step-level and trajectory-level signals. Max steps 800. Learning rate 2e−6. $\beta = 0.05$. Initialized from Verified.

RefineCut-Evo DPO. Candidate generation: $K = 4$ candidates per training state, on 1,500 training states. Pair construction: 779 rubric-margin pairs oversampled to 1,651 training rows (Appendix F). Max steps 600 with an intermediate checkpoint at step 300. Learning rate 1e−6. $\beta = 0.05$. Initialized from Mixed-Pref. Step 300 is selected on dev100 VES (Table 11).

Ablation variants. Verifier-only, No-margin, and UCPO-lite share the same backbone, candidate pool, and DPO recipe as Rubric-margin; they differ only in the pair-scoring function and the margin filter (Table 13). Hyperparameters and elapsed wall-clock times are matched across variants.

Compute budget. All training ran on a single NVIDIA A100 GPU. The main pipeline reported in the paper consumed approximately 14.5 GPU-hours in total: Verified SFT (1,200 steps, ∼7.5 h), Mixed-Pref preference fine-tuning (800 steps, ∼2.1 h), and the RefineCut-Evo Rubric-margin DPO run (600 steps, ∼4.9 h; the step-300 checkpoint is selected by dev VES). The three step-300 DPO ablations (Verifier-only, No-margin, UCPO-lite) each consumed an additional ∼2.5 wall-clock hours, matched across variants as stated above. Multi-teacher trajectory collection used frontier API calls to GPT-5.4, Qwen3-Max, and DeepSeek-V4-Pro (Section 2.5); no model training was performed through those APIs.

Decoding. All closed-loop evaluations use R=1 greedy decoding at generation time (a single sample per step). An R=4 verifier-reranking decoding variant (four samples re-ranked by the verifier) was attempted as a diagnostic and underperforms; it is reported as a negative diagnostic only (Appendix G). R is the inference-time sampling/reranking pool size and is distinct from the training-time student-candidate count K=4 (Section 3.5).

F RefineCut-Evo Details

Verifier replay procedure. Algorithm 1 formalizes the verifier replay and branch-arbitration procedure used by the Stage 1 distillation (Section 3.3).

Algorithm 1 Verifier replay and branch arbitration.

Require: task x, initial state s_0, teacher trajectory $\tau = (step_1, \ldots, step_T)$
Ensure: replayed trajectory with per-branch scores

```
s ← s_0;  L ← InitLedger(x, s_0)
for t = 1, ..., T do
    R_t ← ∅
    for each candidate branch b_t^k in step_t do
        p_t^k ← Canonicalize(b_t^k)
        if not Valid(p_t^k, s) then
            r_t^k ← Invalid; continue
        end if
        s' ← Apply(s, p_t^k)
        L' ← Recompute(x, s')
        r_t^k ← Score(s, s', L, L')          ▷ Eq. 4
        R_t ← R_t ∪ {(p_t^k, r_t^k)}
    end for
    b_t^* ← arg max_k r_t^k
    s ← Apply(s, p_t^*)
    L ← Recompute(x, s)
end for
return replayed trajectory with {R_t, b_t^*}_{t=1}^T
```

Rubric construction. For each canonical task we construct a task-specific editing rubric with seven criteria ER1–ER7 (Section 3.5). The criteria and their average weights across the train set are listed in Table 9. Weights are computed per task-family from a template that puts mass on the criteria most relevant to the family.

Candidate generation. States with sampled candidates: 1,500. Total candidates: 6,000 (K=4 per state). JSON parse success 0.9968; patch-apply success 0.9960.

ID	Criterion	Avg. weight
ER1	Intent and Story Alignment	0.09
ER2	Ledger Satisfaction	0.25
ER3	Clip Grounding and Relevance	0.22
ER4	Timeline and Segment Coherence	0.11
ER5	Duration and Pacing Alignment	0.15
ER6	Music / Beat Alignment	0.05
ER7	Edit Economy / Minimality	0.11

Table 9: Average rubric weights across the train set. Family A puts more mass on ER4; families B and C put more mass on ER3.

Rubric scoring. Mean rubric score 0.67 (std 0.13). The candidate pool shows a high verifier / rubric correlation $\rho = 0.94$, which indicates that the rubric mostly tracks the verifier signal on these states. The joint score used for pair construction is $S(c) = 0.65V(c) + 0.35R(c)$, making the self-improvement stage verifier-centered while retaining rubric-structured margins. Criterion ER1 (intent) and the judge-side component of ER3 (clip relevance, judge-side) fall back to deterministic proxies when external judge features are unavailable; the `fallback_used` flag is recorded for every one of the 6,000 candidates.

Pair construction. 779 chosen / rejected pairs are kept after applying a fixed margin threshold. Mean margin 0.18; median margin 0.19; hard-negative rate 0.987. The training set is oversampled to 1,651 rows. The number of training states skipped because the highest-scoring valid repair and the lower-scoring hard negative fell inside the margin band is 635.

Ablation caveats. Rubric-structured scoring provides a consistent but modest gain over verifier-only scoring (+0.015 VES). The main signal remains verifier-scored student self-improvement, while rubric scoring improves margin construction and interpretability. No-margin DPO is weaker, which says that margin filtering matters. UCPO-lite underperforms in this setting and is reported as a negative diagnostic. Temporal contrast was not trained: the data summary records 0 temporal-contrast pairs.

G Additional Results

Extended teacher replay audit corresponding to Table 2.

Teacher	N	Branches	FCSR	Hard	PASR	ReqCR	NoReg	Dur	VES
GPT-5.4	2,000	6,000	0.379	0.015	0.872	0.698	1.000	0.202	0.521
Qwen3-Max	1,946	3,892	0.446	0.016	0.989	0.712	1.000	0.157	0.557
DeepSeek-V4-Pro	1,959	3,920	0.473	0.028	0.946	0.703	1.000	0.156	0.559

Table 10: **Extended teacher replay audit corresponding to Table 2.** This table expands the main teacher replay diagnostic with NoRegression and Dur@2s. It reports recomputed verifier-best branch diagnostics from the final replay logs and is not a separate student closed-loop result.

dev100 checkpoint selection.

Model	FCSR	Hard	Dur	Cvg	VES
SFT-Verified	0.473	0.240	0.360	0.260	0.496
Mixed-Pref	0.534	0.260	0.400	0.310	0.543
Evo step300	**0.608**	**0.450**	**0.570**	**0.470**	**0.604**
Evo step600	0.590	0.440	0.570	0.470	0.587

Table 11: **dev100 checkpoint selection for RefineCut-Evo.** Step-300 is selected by dev VES; step-600 shows mild over-optimization.

Paired significance.

Subset	vs.	ΔVES	95% CI	W/T/L	p
Common-100	Mixed-Pref	+0.059	[0.028, 0.092]	22/73/5	0.0015
Common-100	SFT-Verified	+0.066	[0.036, 0.096]	24/71/5	0.0005
Common-100	Traj-Pref	+0.060	[0.029, 0.091]	22/73/5	0.0015
Canonical-clean	Mixed-Pref	+0.058	[0.027, 0.090]	21/66/5	0.0025
Canonical-clean	SFT-Verified	+0.068	[0.036, 0.100]	23/64/5	0.0009
Canonical-clean	Traj-Pref	+0.059	[0.028, 0.091]	21/66/5	0.0025

Table 12: **Paired significance for RefineCut-Evo.** ΔVES is the mean paired per-task difference, not the aggregate table-level difference; the sign-test p-value treats ties as halves.

RefineCut-Evo ablations.

Variant	FCSR	Hard	Dur	Cvg	VES	ΔVES vs. final
Rubric-margin DPO	**0.9530**	**0.820**	**0.980**	**0.950**	**0.9237**	0.0000
Verifier-only DPO	0.9395	0.780	0.940	0.900	0.9085	-0.0152
No-margin DPO	0.9178	0.760	0.910	0.870	0.8965	-0.0272
UCPO-lite DPO	0.9012	0.740	0.890	0.840	0.8800	-0.0437

Table 13: **RefineCut-Evo ablations** on Common-100. Rubric-margin DPO is the final checkpoint; verifier-only, no-margin, and UCPO-lite variants are weaker under the same backbone, candidate pool, and DPO recipe.

Closed-loop failure taxonomy.

Outcome	Mixed-Pref	RefineCut-Evo	Change
OK plan	74	**91**	+17
DurationMismatch	16	**1**	-15
LowCSRNoSingleFailure	7	**4**	-3
MissingClipReference	3	3	0
PatchApplyFail	0	1	+1

Table 14: **Closed-loop failure taxonomy on Common-100.** The compact view highlights the main shift: RefineCut-Evo turns more tasks into fully OK plans, primarily by removing duration mismatches. Counts are task counts out of 100.

Canonical-clean closed-loop.

Model	N	FCSR	Hard	Dur	Cvg	VES
Base	92	0.607	0.152	0.652	0.217	0.591
Raw	92	0.552	0.152	0.424	0.228	0.612
Verified	92	0.872	0.609	0.804	0.772	0.849
Traj-Pref	92	0.872	0.652	0.837	0.783	0.858
Mixed-Pref	92	0.879	0.652	0.826	0.793	0.859
RefineCut-Evo	92	**0.949**	**0.804**	**0.978**	**0.946**	**0.917**

Table 15: Closed-loop results on the canonical-clean subset of Common-100 ($N = 92$).

Per-family breakdown.

Family	Mixed-Pref VES	RefineCut-Evo VES
A (composition)	0.865	**0.892**
B (insert / repair)	0.871	**0.975**
C (generative)	0.853	**0.890**

Table 16: Per-family closed-loop VES on Common-100. Per-subtype counts are released in the supplementary artifacts.

Reward-hacking and diversity diagnostics.

Model	Op.Count	Op.Ent.	DurDev (s)	InvalidClip
Mixed-Pref	0.00	≈ 0	1.55	0
RefineCut-Evo	3.77	0.99	**0.34**	0

Table 17: Reward-hacking and operator-diversity diagnostics. The over-rewrite rate and no-regression rate are unchanged (0 and 1.0 respectively).

Distillation-stage API-judge audit. Three independent blind API judges are run on distillation-stage variants using a stripped pairwise prompt with model identifiers, file-path leaks, and judge-side self-identification removed (full prompt in Appendix I). Per-judge raw counts on the 20-pair Common-100 panel are in Table 18. These audits are not used as RefineCut-Evo evidence. Pooled non-tie win rates, aggregated across judges, are 0.962 for Verified vs. Raw, 0.880 for Mixed-Pref vs. Verified, 0.764 for Mixed-Pref vs. Base, 0.805 for Traj-Pref vs. Verified, and 0.655 for Traj-Pref vs. Base.

Judge	Comparison	W	T	L	n	$\mathrm{WR}_{\mathrm{ex}}$
Judge A	Verified vs. Raw	17	2	1	20	0.944
Judge B†	Verified vs. Raw	18	1	0	**19**	1.000
Judge C	Verified vs. Raw	16	3	1	20	0.941
Judge A	Mixed vs. Verified	8	10	2	20	0.800
Judge B†	Mixed vs. Verified	6	13	0	**19**	1.000
Judge C	Mixed vs. Verified	8	11	1	20	0.889
Judge A	Mixed vs. Base	15	1	4	20	0.789
Judge B	Mixed vs. Base	14	2	4	20	0.778
Judge C	Mixed vs. Base	13	2	5	20	0.722
Judge A	Traj-Pref vs. Verified	11	7	2	20	0.846
Judge B	Traj-Pref vs. Verified	10	6	4	20	0.714
Judge C	Traj-Pref vs. Verified	12	6	2	20	0.857
Judge A	Traj-Pref vs. Base	10	2	8	20	0.556
Judge B	Traj-Pref vs. Base	14	0	6	20	0.700
Judge C	Traj-Pref vs. Base	14	0	6	20	0.700

Table 18: Per-(judge, comparison) raw counts for the distillation-stage blind API-judge audit. $\mathrm{WR}_{\mathrm{ex}} = W/(W + L)$. †Judge B parse coverage is n=19 on these two rows.

Negative and inconclusive diagnostics.

R=4 **verifier reranking.** An inference-time R=4 verifier-reranking variant (four samples reranked by the verifier) underperforms the R=1 closed-loop setting and is reported as a negative diagnostic; the closed-loop results are run at R=1 throughout, and the R / K distinction is given under *Decoding* in Appendix E.

Step-Pref. A step-level-only preference variant (Step-Pref) is inconclusive: gains over Verified on the closed-loop summary are within the variance of the multi-seed runs we performed. We do not promote it as a method contribution.

Final-stage Evo API-judge diagnostic. A separate corrected API-judge diagnostic was run for final-stage Evo comparisons, but we do not use it as primary evidence because the main claim is supported by verifier-grounded planning metrics and rendered-video human evaluation. After parse filtering, the aggregate non-tie win rates are 0.574 for Evo vs. Mixed-Pref (108W/108T/80L), 0.545 for Evo vs. SFT-Verified (103W/110T/86L), and 0.548 for Evo vs. Traj-Pref (102W/108T/84L).

H Human Evaluation and Human-Written Brief Validation

This appendix reports the final human-evaluation and construct-validity tables. Human evaluators compare blind randomized A/B rendered storyboard previews. Each comparison has 150 judged pairs and three annotators per pair; annotators have

one to five years of video-editing experience. Preference score is $(W + 0.5T)/N$ for the first model in each comparison. Annotators were recruited from university campuses and partner companies, all with one to five years of video-editing experience, and were compensated at a rate consistent with local standards for their region. They received a written instruction sheet covering the rating axes, blinded protocol, compensation, opt-out policy, and a brief content-risk disclaimer; the full instruction text is released with the supplementary artifacts.

Comparison	W/T/L	Pref. Score	Non-tie WR	Maj. Agree.	κ	α
Evo vs Mixed	100/34/16	**0.780**	**0.862**	0.873	0.620	0.580
Evo vs Traj	95/37/18	0.757	0.841	0.860	0.590	0.553
Evo vs SFT	97/35/18	0.763	0.843	0.867	0.601	0.562
Mixed vs Base	124/18/8	0.887	0.939	0.927	0.781	0.742

Table 19: **Final pairwise human evaluation** ($N = 150$ per comparison, three annotators per pair). Higher preference scores and non-tie win rates prefer the first model. The sanity comparison confirms that the protocol separates a known large quality gap.

Model	FCSR	Hard	Dur	Cvg	ReqCR	VES
Base	0.582	0.140	0.620	0.180	0.270	0.578
Raw	0.534	0.160	0.420	0.220	0.710	0.608
Verified	0.856	0.580	0.780	0.740	0.960	0.839
Traj-Pref	0.852	0.620	0.800	0.740	0.960	0.842
Mixed-Pref	0.864	0.620	0.800	0.760	0.960	0.848
RefineCut-Evo	**0.928**	**0.760**	**0.940**	**0.900**	**0.980**	**0.902**

Table 20: **Human-written brief validation** ($N = 50$). The ranking from Common-100 is preserved; the Evo-stage gap remains $+0.054$ VES over Mixed-Pref.

I Full Prompts

This appendix collects compact versions of the six prompts used throughout the pipeline. The full text of each prompt is included in the supplementary artifacts; each entry below records the prompt role, expected inputs, output schema, and a compact excerpt for readability.

Task-generation prompt. Purpose: generate benchmark tasks and ledgers from clip pools. **Input fields:** clip pool, task family, subtype, ledger template. **Output schema:** task record JSON. **Version/source path:** supplementary prompt artifact; compact excerpt below.

Task-generation prompt (compact)

```
system: You are a video-editing task author.
task: Given a clip pool and a family/subtype slot,
emit one task record with fields
  task_id, task_type, task_subtype,
  brief, target_duration,
  constraint_ledger (>= 4 items, includes target_duration)
      ,
  clip_pool, structure, subtype_constraints.
constraints:
  - clip_ids in clip_pool MUST come from given_pool.
  - clip_pool size in [30, 60] when given_pool>=30.
  - ledger items use the 14 allowed constraint types.
output: STRICT JSON; no commentary.
```

Caption prompt. Purpose: convert sampled clip frames into metadata read by the planner. **Input fields:** clip frames, motion bucket, clip id, source. **Output schema:** one caption metadata row. **Version/source path:** supplementary prompt artifact; compact excerpt below.

Caption prompt (compact)

```
system: You are a video-clip captioner.
input: a clip (sampled frames; sampling rate
  determined by motion bucket: low=3, med=5, high=7).
task: emit one JSON row with fields
  clip_id, source, duration, subject, action,
  scene, camera, scene_category,
  motion_intensity, caption_short,
  model_used (record which captioner produced this row),
  ts (ISO timestamp).
output: STRICT JSON; no commentary.
```

Trajectory-teacher prompt. Purpose: collect frontier-teacher candidate repair branches. **Input fields:** brief, clip pool, music metadata, timeline state, constraint ledger. **Output schema:** four RefinePatch candidates. **Version/source path:** supplementary prompt artifact; compact excerpt below.

Trajectory-teacher prompt (compact)

```
system: You are a frontier video-editing planner.
input: brief, clip_pool with captions and metadata,
  music metadata, current timeline state,
  constraint_ledger (with per-item satisfied flags).
task: at this refine step, emit EXACTLY 4 candidate
  RefinePatch branches. Each branch must contain
  operations[], rationale_against_ledger[],
  repair_operator, thought, confidence.
output: a list of 4 RefinePatch objects in STRICT JSON.
```

Unified PatchPlanner prompt. Purpose: run the trained planner in closed-loop evaluation. **Input fields:** brief, clip pool, captions, music metadata, current state, ledger, violated subset. **Output schema:** one RefinePatch. **Version/source path:** supplementary prompt artifact; compact excerpt below.

Unified PatchPlanner prompt (compact, used at test)

```
system: You are an executable video-editing planner.
input: brief, clip_pool, captions, music_metadata,
  current_state, constraint_ledger (with satisfied flags
  and a violated_subset listing currently unmet items).
task: emit one RefinePatch that targets ledger items
  in violated_subset and does not regress satisfied items.
constraints:
  - operations limited to RFC 6902 add/remove/replace
    on canonical paths of current_state.
  - reference only clip_ids that appear in clip_pool.
  - keep rationale_against_ledger aligned with operations.
output: STRICT JSON; no commentary.
```

Rubric construction template. **Purpose:** build task-specific ER1–ER7 scoring weights for RefineCut-Evo. **Input fields:** task family and subtype. **Output schema:** rubric JSON with criteria, weights, and fallbacks. **Version/source path:** supplementary prompt artifact; compact excerpt below.

Rubric construction template (compact)

```
template: For task_subtype X under family F:
  - assign ER1..ER7 a non-negative weight; sum to 1.
  - default weights start from the family template
    (A puts mass on ER4; B and C put mass on ER3).
  - for each criterion list:
      criterion_name, weight,
      verifier_signals_used,
      fallback_proxy_if_judge_absent.
output: a rubric JSON used by the Evo scorer.
```

Stripped blind judge prompt. **Purpose:** audit distillation-stage plans without model identity leakage. **Input fields:** brief, ledger, plan A, plan B. **Output schema:** strict JSON winner plus reason. **Version/source path:** supplementary prompt artifact; compact excerpt below.

Stripped blind judge prompt (compact)

```
system: You are a blind video-editing-plan judge.
You will see two plans for the SAME task.
The model identity and any file path are removed.
input: brief, ledger, plan_A, plan_B.
task: pick the plan that better satisfies the brief and
  the ledger; output one of {A, B, tie} with a brief
  reason. Do not reveal the model identity in your
  reasoning. Do not mention "as an LLM".
output: STRICT JSON {"winner": "A"|"B"|"tie",
                     "reason": "..."}.
```

J Verifier Implementation

The verifier is deterministic: it validates a patch, applies it to the current state, recomputes the ledger, and emits both gate signals and branch-score components.

Algorithm 2 Verifier replay for one candidate patch.

Require: state s_t, patch p_t, task record τ
if p_t fails schema validation **then**
  return reject(schema)
end if
$s_{t+1} \leftarrow \mathrm{Apply}(s_t, p_t)$
if s_{t+1} is invalid **then**
  return reject(apply)
end if
if timeline is not renderable **then**
  return reject(timeline)
end if
$L_t \leftarrow \mathrm{RecomputeLedger}(s_t, \tau)$
$L_{t+1} \leftarrow \mathrm{RecomputeLedger}(s_{t+1}, \tau)$
compute ΔCSR, TargetedRepair, ReqClipRecall, PASR, NoRegression, and Locality
score $\leftarrow w_1\Delta\mathrm{CSR} + w_2\mathrm{TargetedRepair} + w_3\mathrm{ReqClipRecall} + w_4\mathrm{PASR} + w_5\mathrm{NoRegression} + w_6\mathrm{Locality}$
return accepted patch, new state, new ledger, score terms

Metric	Rule
ReqClipRecall denominator	Number of required keep-clip items; tasks with none are excluded from this component average.
NoRegression	Items satisfied at L_t must remain satisfied after the patch; the all-steps variant conjoins this across the loop.
Converged@3	Fires when terminal constraint satisfaction reaches at least 0.8 within three repair steps, including early termination.
TargetedRepair	Counts only listed ledger items that were unsatisfied before the patch and become satisfied after replay.
VES	Uses the fixed weights in Appendix D; undefined ReqClipRecall tasks use the normalized remaining components.

Table 21: **Verifier denominator and aggregation rules.** These rules keep branch scoring deterministic and avoid treating undefined clip-recall denominators as successes or failures.

K Worked Example

This compact example illustrates one verifier-replay step for a `b_roll_insert` task. It is schematic and is not a released-record identifier.

Item	Illustrative value
Brief	Insert a short b-roll cutaway between a speaker close-up and a wide street shot.
Pool	C001 speaker close-up; C002 wide street at dusk; C007 traffic-light cutaway; C012 empty alley.
Initial state	sequence [C001, C002], durations [5.4, 10.0], total 15.4s. The 13s duration target is violated; required clips C001 and C002 are already present.
Branch 1	Replace the sequence with [C001, C007, C002] and set durations to [5.0, 3.0, 5.0]; operator `reselect_clip`.
Branch 2	Keep [C001, C002] but set durations to [4.0, 9.0]; operator `adjust_parameter`.

Replay outcome. Branch 1 is selected: it fixes duration, preserves required clips, inserts C007, and touches only ledger-relevant fields. Branch 2 fixes duration but misses the b-roll intent, so it is the hard negative. After Branch 1, all four ledger items pass and the loop ends at step 1.