VIDEOEVOLVE: EVOLVING AGENT HARNESSES FOR VIDEO TEMPORAL GROUNDING

Anonymous authors
Paper under double-blind review

ABSTRACT

Video temporal grounding aims to localize events in videos from natural-language queries. For agents built around frozen video-language models, the harness determines how queries guide temporal predictions and how those predictions are refined. Manually refining these harnesses requires diagnosing grounding failures and coordinating changes to both agent workflows and instructions. We introduce VideoEvolve, a framework that automatically evolves agent harnesses for video temporal grounding. VideoEvolve uses a Cloze-Structured Harness Representation that preserves stage interfaces while leaving agent workflows and instructions open to evolution. Branch-Guided Harness Evolution preserves promising code branches for continued refinement, using execution feedback to guide local edits and validation to determine which improvements are carried forward. Experiments demonstrate improved grounding performance across multiple benchmarks. Component analyses identify instruction refinement as a consistent source of gains, while the benefits of evolved code vary across evaluation settings. Together, these results support automated harness evolution as an effective approach to improving video temporal grounding.

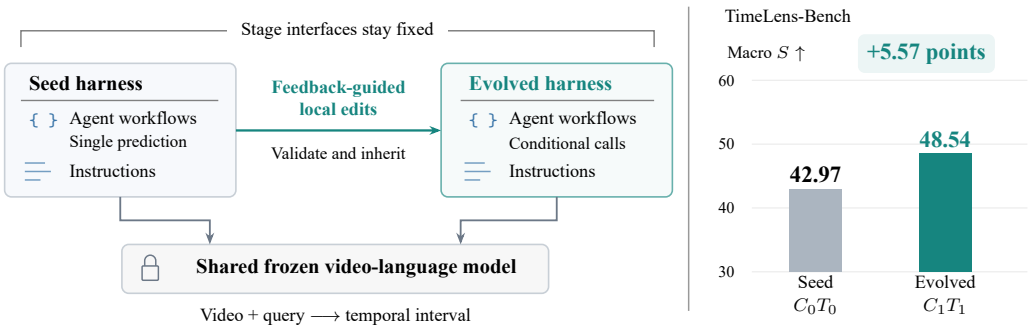
1 INTRODUCTION

Video temporal grounding localizes the interval in a video that matches a natural-language query, connecting video understanding to temporally grounded retrieval and reasoning. Identifying the right event and its boundaries can remain difficult when similar actions recur or a query specifies relations between events (Zhang et al., 2026c;d). For example, the same action may be clearly recognizable at several points in a video, while its position before or after another event determines which occurrence matches the query. An agent may need to inspect additional evidence, revisit an initial prediction, or resolve competing interval hypotheses. How these decisions are organized affects the path from interpreting video content to selecting a temporal interval. For agents built around video-language models, this process depends on both the model and the execution procedure surrounding it.

Recent work improves temporal grounding through instruction tuning (Ren et al., 2024; Huang et al., 2024), reinforcement learning (Wang et al., 2025a), and self-evolving model training (Jung et al., 2026). Alongside these advances in model learning, agentic approaches organize query decomposition, grounding, verification, and adaptive reasoning into executable workflows (Zheng et al., 2024; Liu et al., 2026; Zhang et al., 2026d). The workflows and instructions surrounding a model form its *harness* (Lee et al., 2026; Zhang et al., 2026a). Improving a grounding harness requires coordinating how instructions describe an event with how code obtains and combines predictions. For example, a revised rule for distinguishing repeated events may call for changes to when a prediction is checked or which alternative interval is selected. Manually making such changes requires diagnosing failures and translating them into reusable procedures. A correction for one ambiguous query may change behavior elsewhere, so successful predictions also need to inform how the harness is revised. This motivates automatically improving the harness while keeping the target video-language model frozen.

Program-based agent design already offers a foundation for this approach: ADAS and Meta-Harness search over executable agent programs (Hu et al., 2025; Lee et al., 2026), while the Darwin Gödel Machine retains historical programs as starting points for further improvement (Zhang et al., 2026b).

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.



(a) Evolving the harness **(b) Grounding gains**

Figure 1: **VideoEvolve improves the harness around a frozen model.** (a) Workflows and instructions evolve within stable stage interfaces, guided by execution feedback and validation (schematic). (b) Evolving the harness improves macro S on TimeLens-Bench (vertical axis starts at 30).

We organize harness evolution for temporal grounding around two requirements. First, changes to hypothesis generation must remain compatible with downstream verification and selection. The representation should therefore preserve the interfaces that carry observations, interval hypotheses, and verification evidence, while leaving internal workflows and instructions open to revision. Second, an intermediate code change may require further edits before the complete harness is ready for adoption. Search should allow such programs to continue developing, while using evaluation of the complete candidate to decide which changes are inherited. A local edit must be assessed in the context of the full harness, since its effect can depend on existing instructions and downstream decision rules. Together, these requirements connect an editable grounding representation with a search procedure that separates continued exploration from acceptance.

We introduce **VideoEvolve**, a framework that evolves grounding harnesses using labeled development feedback while keeping the target model frozen (Figure 2). Its *Cloze-Structured Harness Representation* preserves stage interfaces while allowing executable workflows and instructions within them to evolve. Execution traces, grounding errors, and successful predictions from the current harness provide evidence about which decisions should be revised and which behavior should be preserved. *Branch-Guided Harness Evolution* uses execution feedback to guide local edits, preserves promising code branches for continued refinement, and validates complete candidates before adopting them as the basis for subsequent iterations. We evaluate the resulting harnesses on five grounding benchmarks. Figure 1 illustrates the approach and its gain over the initial harness. VideoEvolve exceeds the compared frozen-model workflows on Charades-STA and ActivityNet, and matched comparisons show gains over its seed on all five benchmarks. Component analyses identify instruction refinement as a consistent source of gains, while code contributions vary across benchmarks. This distinction motivates evaluating both grounding quality and inference usage when assessing an evolved harness.

Our contributions are threefold:

- We introduce VideoEvolve and a Cloze-Structured Harness Representation that supports evolving grounding workflows and instructions within stable stage interfaces around a frozen video-language model.
- We develop Branch-Guided Harness Evolution, connecting feedback-guided local edits and continued code-branch refinement with validation-based inheritance of complete harnesses. Development feedback guides branch retention, while validation determines which complete candidates are accepted and inherited by subsequent iterations.
- We evaluate the evolved harnesses on five grounding benchmarks, using search controls, code–instruction exchanges, and deployment-cost measurements to characterize performance, component contributions, and inference cost.

2 Related Work

2.1 Video Temporal Grounding

Feature-based grounding methods learn temporal predictions from video–text representations. MH-DETR (Xu et al., 2024) and UVCOM (Xiao et al., 2024) improve cross-modal interaction and temporal modeling for moment retrieval and highlight detection. Video-language models extend this line through architectures and training procedures that connect visual content to timestamps. TimeChat (Ren et al., 2024) introduces time-aware encoding and instruction tuning, VTimeLLM (Huang et al., 2024) develops boundary-aware training, and LLaVA-MR (Lu et al., 2026) combines frame and time encoding with generative moment prediction. Time-R1 (Wang et al., 2025a) and TimeLens (Zhang et al., 2026c) investigate reinforcement learning with verifiable rewards for temporal localization. More recently, EvoGround (Jung et al., 2026) constructs training signals from raw videos through a self-reinforcing proposer–solver loop that updates both models. These approaches improve learned grounding components through task-specific training. VideoEvolve uses labeled development feedback to optimize a reusable harness around a frozen video-language model, directing improvement toward its executable workflow and instructions.

2.2 Agentic Video Understanding

Agentic and workflow-based methods organize video evidence into intermediate decisions before producing a prediction. For temporal grounding, TFVTG (Zheng et al., 2024) decomposes queries into sub-events and combines their localization results using temporal relations, while ChatVTG (Qu et al., 2024) combines clip captioning, query matching, and moment refinement. DART (Zhang et al., 2026d) adapts the reasoning route to query difficulty. Beyond these grounding workflows, VideoTree (Wang et al., 2025b) expands a query-relevant hierarchy of visual evidence, and VideoMind (Liu et al., 2026) coordinates planning, grounding, verification, and answering through role-specific LoRA adapters. These approaches illustrate how evidence selection, additional observations, and prediction refinement can be organized within an agent. Their procedures can adapt to individual videos and queries; VideoEvolve additionally treats the programs implementing such decisions as objects of development-time search. Execution feedback guides changes that persist in the harness used for subsequent grounding tasks.

2.3 Automated Agent Design and Harness Evolution

Automated agent design broadens optimization from model inputs to the systems that construct and act on them. ADAS (Hu et al., 2025) represents agent designs in code, allowing search over executable workflows. Meta-Harness (Lee et al., 2026) optimizes task-specific harness programs using prior source code, scores, and execution traces. Self-Harness (Zhang et al., 2026a) uses the evaluated model itself to diagnose weaknesses and propose bounded harness edits, which are then evaluated through regression tests. VideoEvolve follows this program-based view and structures the editable space around the information exchanged during video grounding. Its Cloze-Structured Harness Representation preserves stage interfaces while allowing internal workflows and instructions to evolve. This structure maintains the passage of observations, interval hypotheses, and verification evidence as their processing rules change; an external coding agent proposes the modifications.

Search history provides alternative starting points for further improvement. The Darwin Gödel Machine (DGM) (Zhang et al., 2026b) maintains an archive of agents whose code can be modified, including intermediate designs that serve as stepping stones. AutoHarness (Lou et al., 2026) organizes code hypotheses in a tree and selects nodes for feedback-guided refinement. Meta-Harness retains evaluated programs and allows its proposer to inspect prior artifacts without imposing a parent-selection rule. VideoEvolve's Branch-Guided Harness Evolution organizes exploration around a current champion and a retained code branch, allowing further edits to a program that has not yet replaced the champion. Validation determines whether a complete candidate is adopted and inherited by subsequent iterations. This separates continued exploration from acceptance within the stage-structured grounding harness.

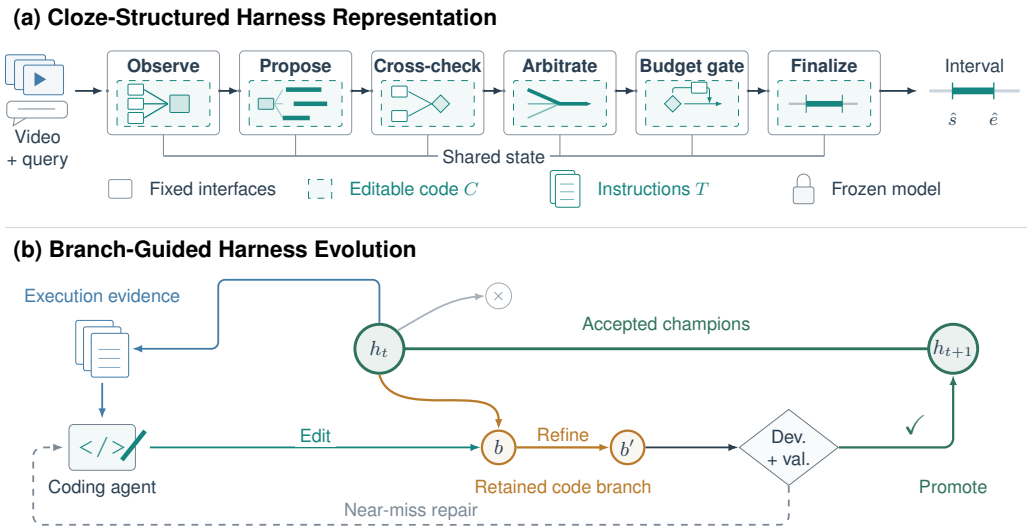


Figure 2: **VideoEvolve.** (a) Fixed stage interfaces contain editable workflows, with instructions evolving alongside code. (b) Retained code branches are refined off the accepted-harness mainline. Development comparison and validation gate promotion, while eligible validation rejections provide a repair path.

3 VIDEOEVOLVE

A video grounding harness coordinates how a model interprets a query, revisits visual evidence, and resolves competing interval predictions. VideoEvolve improves this procedure by evolving the harness around a frozen video-language model. Its *Cloze-Structured Harness Representation* preserves the interfaces through which grounding stages exchange information, while leaving their workflows and instructions editable. *Branch-Guided Harness Evolution* uses execution feedback to refine candidate programs, retaining intermediate code branches for further exploration. Validation determines which complete harness is inherited by subsequent iterations. Figure 2 illustrates the framework.

3.1 TASK AND HARNESS OPTIMIZATION

Given a video V of duration d and a natural-language query q, video temporal grounding predicts an interval $\hat{y} = (\hat{s}, \hat{e})$ with $0 \le \hat{s} \le \hat{e} \le d$. We represent a harness as $h = (C, T)$: executable code C controls model calls, state updates, and interval selection, while text T supplies task instructions and reusable guidance (*skills*). Together, they determine the agent workflow. With a frozen target model M and tool interfaces U, the prediction is $\hat{y} = \mathrm{Exec}(M, U, h; V, q)$.

Following program-based agent design and harness optimization (Hu et al., 2025; Lee et al., 2026), we optimize h while keeping M and U fixed. Let $\mathcal{H}_B$ contain executable harnesses that satisfy the stage interfaces and per-instance resource limits B. Starting from a working seed h_{seed}, we seek to

$$\max_{h \in \mathcal{H}_B} S_{\text{dev}}(h; M), \tag{1}$$

where S_{dev} measures grounding performance on labeled development data. Search proceeds through local edits under the same resource limits and returns the final accepted harness.

3.2 CLOZE-STRUCTURED HARNESS REPRESENTATION

A change to how interval hypotheses are generated must remain compatible with how later stages verify and select them. We therefore separate a stable execution skeleton from editable stage implementations and instructions. The representation is *cloze-structured*: designated parts of a working

harness can be rewritten while the interfaces connecting them remain fixed. The editable contents are executable code and text, allowing search to introduce new decision rules within this structure.

The six stages in Figure 2 organize observation, hypothesis generation, cross-checking, arbitration, additional resource use, and finalization. Shared state carries visual observations, candidate intervals, verification evidence, and the current selection between stages. Within each stage, code can revise model interactions, tool use, and hypothesis processing; instructions can revise event matching and boundary judgments. Fixed stage order thus supports flexible internal workflows. For example, the `budget_gate` stage can obtain another prediction and revise the selected interval before finalization, allowing refinement to occur within the existing interfaces.

Each proposal modifies one permitted code or text surface and inherits the rest of its parent harness. Candidates must pass interface checks before evaluation as complete executable harnesses. Successive inherited edits can therefore combine changes to code and instructions across iterations, while each proposal targets a local modification.

3.3 Branch-Guided Harness Evolution

An intermediate code change may need further revision before the complete harness is suitable for adoption. VideoEvolve therefore maintains two roles: the *champion* contains accepted improvements and supplies the current execution evidence, while a retained *code branch* provides an alternative parent for continued refinement. A branch can accumulate edits before it qualifies to replace the champion.

At iteration t, executing the champion h_t on the current development batch produces predictions, execution traces, and ground-truth-based evaluations. We assemble these into evidence E_t, including failures and a weakness summary to guide revision, successful cases to preserve, and feedback from earlier attempts. An external coding agent uses this evidence and the permitted edit surface to propose candidate j:

$$h_{t,j} = \mathrm{Edit}(p_{t,j}, E_t), \tag{2}$$

where $p_{t,j}$ is the parent being edited. The current evidence describes the champion's behavior even when the edit starts from a different parent.

Ordinary proposals start from h_t. Code proposals can instead extend the retained code attempt with the highest recorded development score among those evaluated under the current champion. Retention makes an unaccepted program available for another edit; it does not change the active champion. The branch resets after a promotion or when its retry allowance is exhausted, anchoring subsequent exploration to the current accepted harness.

Validation rejections provide targeted repair feedback. A candidate that improves the development score but fails validation can become a *near miss*. Eligible near misses receive a bounded repair opportunity: the proposal starts from the champion, or the retained branch for code edits, and uses the rejected candidate's changes together with aggregate regression diagnostics to guide the edit. Branch retention uses development scores, while repair responds to a specific validation rejection. Both mechanisms generate candidates for the same selection procedure.

3.4 Validation-Guided Harness Selection

Selection determines which exploratory changes become the next accepted harness. Because development batches vary across iterations, each valid candidate is compared with the champion on the same batch. Only the highest-scoring candidate that strictly improves on the champion proceeds to validation; otherwise, the champion is retained and code attempts remain eligible for branch continuation.

The candidate and champion are then compared on a validation batch separated from the development batch by video. Acceptance requires grounding performance to remain within the permitted regression limits and execution to satisfy resource and reliability constraints. A fixed reference harness also anchors the quality check (Appendix C.3). Rejection preserves the champion and provides aggregate diagnostics for possible repair. Per-example validation content is withheld from the proposer, but its aggregate feedback can influence later edits; validation is therefore part of search, separate from the final endpoint evaluation.

At most one complete candidate replaces the champion in each iteration. Subsequent proposals inherit its evaluated combination of code and instructions. Search returns the last champion, which is deployed with the frozen model and tools. Algorithm 1 summarizes the loop.

Algorithm 1 Branch-Guided Harness Evolution

Input: frozen model M, tools U, limits B, seed h_{seed},
development and validation data
$h \leftarrow h_{\text{seed}}$; initialize code-branch and repair state
while stopping condition is not met **do**
  Execute h on the current development batch; construct evidence E
  Select parents from h or the retained code branch; add eligible near misses as repair context
  Propose local edits and retain candidates passing interface checks
  Evaluate candidates and h on the same development batch
  $c \leftarrow$ highest-scoring valid candidate, if any
  if c exists and its development score exceeds that of h **then**
    Evaluate c and h on the video-disjoint validation batch
    if the validation gate accepts c **then**
      $h \leftarrow c$
    else
      Record the near miss and aggregate diagnostics for possible repair
  Update the retained code branch from evaluated code attempts
  Reset the branch after promotion or exhausted retries
return h

4 Experiments

We evaluate grounding performance, disentangle code and instruction contributions, and examine the search and inference costs of the resulting harnesses. We then connect retained edits to observed execution behavior.

4.1 Experimental Setup

Benchmarks. We evaluate on five benchmarks: Charades-STA (test), ActivityNet (val_2), QVHighlights (validation), ReXTime (validation), and TimeLens-Bench (Zhang et al., 2026c). ActivityNet includes 17,016 evaluated queries after excluding five videos with inconsistent decoded durations; ReXTime includes all 921 validation queries. TimeLens-Bench contains 9,404 queries across Charades-TimeLens, ActivityNet-TimeLens, and QVHighlights-TimeLens. We use the queries and temporal annotations released by Zhang et al., who refined these datasets through manual annotation and cross-validation.

Evaluation metrics. We report mean temporal IoU (mIoU) and recall at IoU thresholds 0.5 and 0.7 (R@.5 and R@.7), expressed as percentages. On the original QVHighlights split, predictions are scored by their maximum IoU over all annotated windows. Following the TimeLens-Bench evaluation (Zhang et al., 2026c), we report recall and mIoU; we aggregate each metric equally over its three datasets. For search and component analyses, we summarize the grounding metrics as $S = 0.5\,\mathrm{mIoU} + 0.3\,\mathrm{R@.5} + 0.2\,\mathrm{R@.7}$, averaged equally across the three datasets. Failed TimeLens queries receive zero IoU and remain in the denominator; on the original benchmarks, valid fallback intervals are scored and invalid intervals receive zero IoU.

Models and search configuration. Unless otherwise stated, our harness experiments use frozen Qwen3.6-35B-A3B-FP8 (Qwen-35B). Additional experiments use gemma-4-26B-A4B-it. A coding agent using gpt-5.6-sol proposes harness edits. Each search allocates 60 proposal slots over 20 rounds, subject to per-query limits of four solver calls and 96 observed frames.

Comparisons. Published training-based methods and frozen-model workflows provide task-level references under their reported settings. Controlled comparisons hold the target model fixed: we compare the evolved harness with its exact Cloze seed and exchange seed and evolved components. These evaluations use frozen endpoints without further optimization. A separate Full/No-Branch

search pair tests branch retention while preserving champion inheritance, historical feedback, validation, and repair.

Each search comparison uses one completed run per arm; the reported results compare the resulting frozen endpoints.

4.2 Main Results on Video Temporal Grounding

Table 1: Video temporal grounding results on five benchmarks, comparing VideoEvolve with training-based and agentic methods. R@.5 and mIoU are reported as percentages; higher is better. Bold indicates the best result in each column.

	Charades-STA		ActivityNet		QVHighlights		ReXTime		TimeLens-Bench	
Method	R@.5	mIoU	R@.5	mIoU	R@.5	mIoU	R@.5	mIoU	R@.5	mIoU
Training-based Methods										
MH-DETR	56.37	–	**47.15**	–	60.84	–	–	–	–	–
TimeChat	32.20	–	–	–	–	–	7.61	11.65	–	–
VTimeLLM	27.50	31.20	27.80	30.40	–	–	17.41	20.14	–	–
UVCOM	59.25	–	–	–	65.10	–	–	–	–	–
Time-R1-7B	60.80	–	39.00	–	–	–	–	–	38.17	39.63
EvoGround	60.50	53.10	–	–	–	–	–	–	–	–
VideoMind-7B	59.10	50.20	30.30	33.30	–	–	25.52	27.61	–	–
Agentic and Workflow-based Methods										
ChatVTG	33.01	34.87	–	–	–	–	–	–	–	–
TFVTG	49.97	44.51	27.02	34.10	–	–	–	–	–	–
DART	52.04	48.93	32.14	39.89	–	–	–	–	–	–
VideoEvolve	**70.05**	**61.35**	40.97	**43.28**	**71.68**	**66.27**	**59.17**	**53.93**	**53.53**	**50.73**

Table 1 places VideoEvolve in the broader grounding landscape. VideoEvolve surpasses the listed frozen-model workflows on Charades-STA and ActivityNet and compares favorably with the listed training-based methods across the reported benchmarks, although MH-DETR retains higher ActivityNet recall. It also exceeds the published Time-R1 reference on both TimeLens macro metrics. Strong performance in both recall and mIoU indicates that the evolved harness improves not only the frequency of predictions above the overlap threshold but also average temporal alignment. Together, these results show that harness design offers a practical route to stronger temporal grounding without parameter updates.

Matched seed-to-evolved comparisons provide direct evidence for improvement within the same model. Relative to the exact Cloze seed, VideoEvolve improves mIoU on all five benchmarks. The gains on Charades-STA and QVHighlights extend beyond videos used for harness development and selection. Because the solver is unchanged in these comparisons, the improvements are attributable to the evolved harness as a whole, including its instructions and inference procedure. The following analyses examine branch retention, the resulting harness components, and computational cost.

4.3 Ablation Studies

Contribution of branch retention. A separate Full/No-Branch search pair tests whether continuing promising non-champion branches helps harness discovery. Both arms start from the same seed and use the same development sampling schedule and 60 proposal slots. No-Branch removes branch continuation while retaining champion inheritance, historical feedback, validation, and repair. As Table 2 shows, branch retention improves mIoU on Charades-STA and QVHighlights and increases TimeLens macro S by 1.58 points. ReXTime also improves, while ActivityNet mIoU is slightly lower. The gains on the two video-disjoint benchmarks support keeping alternatives available beyond the current champion. Full also uses fewer development-time solver calls than No-Branch, so the advantage in this pair does not come from greater solver usage. No-Branch still receives historical feedback and can revise the champion, so its lower performance on most benchmarks suggests

Table 2: Branch-retention ablation; endpoints come from a separate search pair.

	Charades-STA		ActivityNet		QVHighlights		ReXTime		TimeLens-Bench	
Search variant	R@.5	mIoU	R@.5	mIoU	R@.5	mIoU	R@.5	mIoU	R@.5	mIoU
No-Branch	66.59	58.82	41.11	**44.24**	67.81	64.22	56.89	53.38	51.56	49.64
Full	**70.81**	**61.90**	**41.99**	44.04	**72.32**	**67.24**	**58.96**	**54.56**	**53.72**	**51.05**

that remembering previous outcomes does not fully replace the ability to continue editing a retained program.

Table 3: Grounding quality and per-query inference cost of harness components. TimeLens-Bench quality is macro-averaged.

Dataset	Harness configuration	R@.5↑	mIoU↑	Calls↓	Input tokens (k)↓
ReXTime	Seed	51.14	48.71	**0.996**	**3.24**
	Evolved instructions only	55.70	52.00	**0.996**	3.97
	Evolved code only	52.99	49.85	1.963	6.53
	Full VideoEvolve	**59.17**	**53.93**	1.948	7.89
TimeLens-Bench	Seed	46.86	45.68	**1.000**	**3.73**
	Evolved instructions only	53.02	50.35	**1.000**	4.45
	Evolved code only	49.73	47.97	1.776	6.50
	Full VideoEvolve	**53.53**	**50.73**	1.726	7.59

Analysis of evolved components. We exchange seed and evolved code and instructions while keeping the target model fixed, without further optimization. Instructions comprise the grounding prompt and textual skills. On ReXTime and TimeLens-Bench, replacing either seed component with its evolved counterpart improves recall and mIoU, and the full harness achieves the highest values of both metrics (Table 3). Restoring either component to its seed version lowers the full harness's performance, showing that both evolved components contribute in these evaluations. Their effects differ across settings: the conditional mIoU gain from evolved code is larger on ReXTime than on TimeLens-Bench. These results connect improvements in query interpretation with changes to prediction refinement, with the strongest evaluated configuration retaining both evolved components.

4.4 Search and Inference Cost

Development-time search. The seed-to-evolved improvements in Table 3 result from a search using 60 proposal slots and 217,767 solver calls. Search cost measures the computation used to discover the harness; deployment cost measures its per-query inference usage. The search budget includes repeated candidate evaluations that are incurred during development. Once an endpoint is selected, subsequent queries execute that fixed harness without invoking the coding agent or repeating candidate selection. This separates the upfront cost of discovering a reusable procedure from the cost paid each time it is applied.

Quality–cost tradeoff at inference. Table 3 also reports deployment usage. The full harness uses more calls and input tokens than the seed and instruction-only configurations on both benchmarks, while all four configurations retain the same 64 sampled frames. Its additional computation therefore revisits existing observations. Average usage varies across benchmarks, reflecting the conditional refinement policy; the quality gains come with increased inference cost. Instructions alone improve quality without adding solver calls, although their longer text increases input-token usage. The full harness uses slightly fewer calls than the code-only configuration on both benchmarks, yet consumes more input tokens. Call counts and token volume therefore capture different aspects of deployment cost: the former reflects how often predictions are revisited, while the latter also accounts for the instructions supplied on each call.

4.5 Evolved Harness Behaviors

From generic localization to explicit evidence tests. The seed grounding instruction asks for the segment that best matches the question. The evolved instruction adds explicit occurrence and boundary tests: "Sweep the full timeline in chronological order and form a candidate interval for every plausible occurrence, rather than committing to the first or most salient partial match." It also requires that "immediately before the start the first required action or state must not yet hold, and immediately after the end the last required action or state must no longer hold." These retained instructions specify how to distinguish repeated events and check temporal boundaries.

From a single prediction to conditional refinement. The seed harness returns one grounding prediction from 64 uniformly sampled frames. The evolved harness keeps these observations but changes how predictions are obtained and combined: it requests another whole-video hypothesis when the timeline is coarsely sampled or the predicted interval occupies an unusually small or large fraction of the video. Substantial disagreement can trigger a third call, after which start and end coordinates are combined by their medians. Evolution thus changes both when to revisit a prediction and how to reconcile alternatives.

Refinement can recover a missed interval. Figure 3 compares the seed, instruction-only, and full configurations on two queries. For the kettle query, the seed and instruction-only predictions precede the annotated event. The full harness executes two calls and returns the annotated interval, [33,36] s. In this example, replacing instructions alone does not recover the event, whereas the complete evolved procedure does. The recorded execution follows the two-hypothesis median path.

Refinement can still miss the event. For the light-switch query, the seed correctly predicts [0,1] s, but the instruction-only configuration shifts the interval to [1,3] s. The full harness uses three calls and returns [1,4] s, still missing the annotated event. This regression illustrates a limitation of combining model-generated hypotheses: combining additional predictions does not guarantee correct temporal alignment. Together, these cases illustrate why the aggregate benefit of executable refinement can depend on the evaluation setting: additional hypotheses can recover a missed event, but they can also preserve or amplify a localization error.

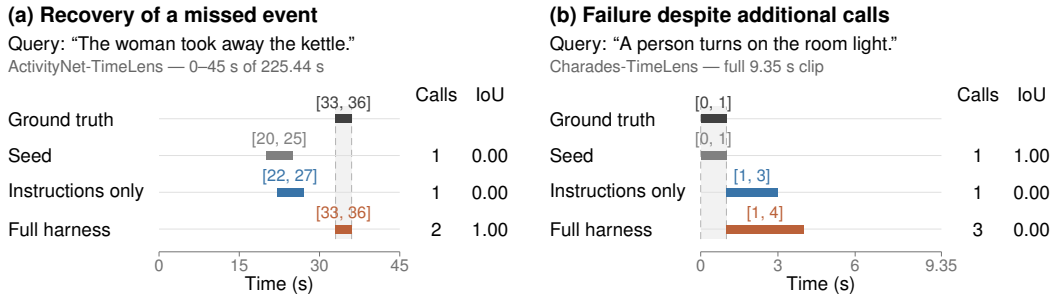


Figure 3: **Grounding refinement and failure.** Final predictions from the seed (C_0T_0), evolved instructions alone (C_0T_1), and full harness (C_1T_1). Shading marks the ground-truth interval; calls count solver invocations.

5 Conclusion

We presented VideoEvolve for automatically evolving video temporal grounding harnesses around frozen models. Cloze-Structured Harness Representation preserves stage interfaces while exposing workflows and instructions to revision; Branch-Guided Harness Evolution combines feedback-guided edits, continued branch exploration, and validation-based inheritance. Experiments show gains over initial harnesses across multiple benchmarks, with component analyses revealing context-dependent contributions from code and instructions. These results support harness evolution as a practical route to stronger grounding without parameter updates.

AI USE STATEMENT

In this work, we used a language-model-driven coding agent to generate and modify the executable code and task instructions of video temporal grounding agents, and used generative AI tools to assist with code editing, figure and table preparation, and reviewing the manuscript for clarity of expression. The authors reviewed the AI-assisted content and take responsibility for the final manuscript, code, and scientific claims.

REFERENCES

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, pp. 21344–21377, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/36b7acf6f6010652b3f2a433774a66fe-Paper-Conference.pdf.

Bin Huang, Xin Wang, Hong Chen, Zihan Song, and Wenwu Zhu. VTimeLLM: Empower LLM to grasp video moments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14271–14280, 2024. URL https://arxiv.org/abs/2311.18445.

Minjoon Jung, Byoung-Tak Zhang, and Lorenzo Torresani. EvoGround: Self-evolving video agents for video temporal grounding. *arXiv preprint arXiv:2605.13803*, 2026. URL https://arxiv.org/abs/2605.13803v1.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-Harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026. URL https://arxiv.org/abs/2603.28052.

Ye Liu, Kevin Qinghong Lin, Chang Wen Chen, and Mike Zheng Shou. VideoMind: A chain-of-LoRA agent for temporal-grounded video reasoning. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2503.13444.

Xinghua Lou, Miguel Lázaro-Gredilla, Antoine Dedieu, Carter Wendelken, Wolfgang Lehrach, and Kevin P. Murphy. AutoHarness: Improving LLM agents by automatically synthesizing a code harness. In *ICLR 2026 Workshop on Recursive Self-Improvement*, 2026. URL https://openreview.net/forum?id=g9rEYVNn5T.

Weiheng Lu, Jian Li, An Yu, and Ming-Ching Chang. LLaVA-MR: Large language-and-vision assistant for video moment retrieval. In *Pattern Recognition – ICPR 2026*, Lecture Notes in Computer Science, pp. 678–693. Springer, 2026. doi: 10.1007/978-3-032-31666-0_44.

Mengxue Qu, Xiaodong Chen, Wu Liu, Alicia Li, and Yao Zhao. ChatVTG: Video temporal grounding via chat with video dialogue large language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 1847–1856, 2024. doi: 10.1109/CVPRW63382.2024.00191. URL https://openaccess.thecvf.com/content/CVPR2024W/PVUW/html/Qu_ChatVTG_Video_Temporal_Grounding_via_Chat_with_Video_Dialogue_Large_CVPRW_2024_paper.html.

Shuhuai Ren, Linli Yao, Shicheng Li, Xu Sun, and Lu Hou. TimeChat: A time-sensitive multimodal large language model for long video understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024. URL https://arxiv.org/abs/2312.02051v2.

Ye Wang, Ziheng Wang, Boshen Xu, Yang Du, Kejun Lin, Zihan Xiao, Zihao Yue, Jianzhong Ju, Liang Zhang, Dingyi Yang, Xiangnan Fang, Zewen He, Zhenbo Luo, Wenxuan Wang, Junqi Lin, Jian Luan, and Qin Jin. Time-R1: Post-training large vision language model for temporal video grounding. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 83330–83364, 2025a. doi: 10.52202/085713-2793. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/7801b29c93b599b8d0c44138596bdeed-Paper-Conference.pdf.

Ziyang Wang, Shoubin Yu, Elias Stengel-Eskin, Jaehong Yoon, Feng Cheng, Gedas Bertasius, and Mohit Bansal. VideoTree: Adaptive tree-based video representation for LLM reasoning on long videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3272–3283, 2025b. doi: 10.1109/CVPR52734.2025.00311.

Yicheng Xiao, Zhuoyan Luo, Yong Liu, Yue Ma, Hengwei Bian, Yatai Ji, Yujiu Yang, and Xiu Li. Bridging the gap: A unified video comprehension framework for moment retrieval and highlight detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 18709–18719, 2024. doi: 10.1109/CVPR52733.2024.01770.

Yifang Xu, Yunzhuo Sun, Benxiang Zhai, Youyao Jia, and Sidan Du. MH-DETR: Video moment and highlight detection with cross-modal transformer. In *2024 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2024. doi: 10.1109/IJCNN60899.2024.10650814.

Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. Self-Harness: Harnesses that improve themselves. *arXiv preprint arXiv:2606.09498*, 2026a. URL https://arxiv.org/abs/2606.09498v3.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents. In *International Conference on Learning Representations*, 2026b. URL https://arxiv.org/abs/2505.22954v3.

Jun Zhang, Teng Wang, Yuying Ge, Yixiao Ge, Xinhao Li, and Limin Wang. TimeLens: Rethinking video temporal grounding with multimodal LLMs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10419–10429, 2026c. URL https://openaccess.thecvf.com/content/CVPR2026/html/Zhang_TimeLens_Rethinking_Video_Temporal_Grounding_with_Multimodal_LLMs_CVPR_2026_paper.html.

Zhengbo Zhang, Mark He Huang, Zhigang Tu, and Ming-Hsuan Yang. DART: Difficulty-adaptive routing for zero-shot video temporal grounding. *arXiv preprint arXiv:2607.00672*, 2026d. URL https://arxiv.org/abs/2607.00672v1.

Minghang Zheng, Xinhao Cai, Qingchao Chen, Yuxin Peng, and Yang Liu. Training-free video temporal grounding using large-scale pre-trained models. In *European Conference on Computer Vision*, 2024. URL https://arxiv.org/abs/2408.16219.

A Additional Model Results

We report the native VideoEvolve endpoint and harness controls for the frozen gemma-4-26B-A4B-it solver. Original-benchmark results and TimeLens-Bench results use their respective evaluation protocols and are reported separately. Table 4 covers all four original benchmarks. QVHighlights uses maximum IoU over all annotated windows, and ReXTime pools all 921 queries, matching the Qwen aggregation. The evolved harness improves mIoU over its exact seed on all four original benchmarks. Failed-request scoring follows Section 4.1.

The TimeLens-Bench comparison follows Section 4.1. The evolved gemma harness averages 83.5 observed frames and 0.61 retrieval calls per query, compared with 64 frames and no tool calls for its seed.

Table 4: Seed and evolved harnesses on the original benchmarks with gemma-26B.

Benchmark	Evaluation	Harness	R@.5	R@.7	mIoU
Charades-STA	test ($n = 3{,}720$)	Seed	22.63	10.35	31.15
		Evolved	**31.05**	**15.27**	**38.63**
ActivityNet	val_2 ($n = 17{,}016$)	Seed	24.58	14.33	30.41
		Evolved	**30.44**	**17.78**	**36.58**
QVHighlights	val ($n = 1{,}550$), all GT	Seed	42.13	**27.23**	41.24
		Evolved	**44.77**	26.97	**44.47**
ReXTime	val ($n = 921$), pooled	Seed	32.57	21.06	34.45
		Evolved	**43.43**	**28.12**	**43.14**

Table 5: Harness comparison on TimeLens-Bench with gemma-26B. Bold denotes best macro S.

	Macro metrics				Calls
Harness	R@.5	R@.7	mIoU	S	/ query
Cloze seed	32.40	20.04	32.07	29.76	1.000
VideoEvolve	35.93	22.66	35.95	**33.28**	2.585

B Published Comparisons and Evaluation Protocols

The training-based reference results in Table 1 come from MH-DETR, TimeChat, VTimeLLM, UVCOM, Time-R1, EvoGround, and VideoMind (Xu et al., 2024; Ren et al., 2024; Huang et al., 2024; Xiao et al., 2024; Wang et al., 2025a; Jung et al., 2026; Liu et al., 2026). The frozen-model workflow results come from ChatVTG, TFVTG, and DART (Qu et al., 2024; Zheng et al., 2024; Zhang et al., 2026d).

Original-benchmark sources. MH-DETR's Tables 2–4 report Charades-STA test, ActivityNet val_2, and QVHighlights validation results; we use its I3D configuration on Charades-STA. UVCOM's Charades-STA result comes from Table 2, and its QVHighlights validation result is the full configuration in Table 5. VTimeLLM's official evaluation instructions identify ActivityNet's test annotation as `val_2.json`. The ReXTime results for TimeChat, VTimeLLM, and VideoMind use the 7B rows without target fine-tuning in VideoMind's Table 3; VideoMind's Tables 6–7 provide its Charades-STA and ActivityNet results. Its official benchmark summary specifies ReXTime validation and ActivityNet val_2.

VideoMind's QVHighlights result uses the test split and is therefore omitted from our validation column. ChatVTG and EvoGround report ActivityNet scores, but the consulted sources do not resolve their evaluation annotations to val_2; these entries are also omitted. A dash denotes an unavailable result for the specified metric and confirmed split.

TimeLens-Bench sources and aggregation. The Time-R1-7B reference in Table 1 comes from Table 1 of the TimeLens-Bench paper (Zhang et al., 2026c). We average each metric equally across its three constituent datasets. Published reference means are computed from the reported rounded values; our means use full-precision exported metrics. All TimeLens-Bench result tables use the same equal-weight aggregation, including macro S; calls per query instead pool recorded calls over the 9,404 evaluated queries.

Comparison settings. Training-based methods in Table 1 update model parameters or components through supervised learning or reinforcement learning; this includes VideoMind's trained LoRA roles. The agentic and workflow-based block keeps pretrained models frozen, although design or evolution may use labels. Model sizes, task inputs, and inference costs differ across methods. ReXTime references use their published grounded-QA settings, including answer options where applicable.

C Component, Cost, and Behavioral Analyses

C.1 Additional component results

Table 6 gives the standard grounding metrics for the exchanged components. For Qwen, the conditional code gain $C_1T_1 - C_0T_1$ is $+0.40$ macro S points, while the conditional instruction gain $C_1T_1 - C_1T_0$ is $+3.18$. Writing S_{ij} for the macro score of C_iT_j, the interaction $I = S_{11} - S_{10} - S_{01} + S_{00}$ is -1.99, indicating overlapping gains on this score scale. On gemma, the conditional code and instruction gains are 3.82 and 2.49, respectively. Its interaction is positive, at 2.79, whereas replacing instructions alone gives -0.30. These results show that the balance of conditional contributions and the sign of interaction depend on the target model.

Table 6: Macro-averaged frozen-component results on TimeLens-Bench.

Harness	R@.5	R@.7	mIoU	S
Qwen-35B				
C_0T_0	46.86	30.33	45.68	42.97
C_0T_1	53.02	35.31	50.35	48.14
C_1T_0	49.73	32.31	47.97	45.36
C_1T_1	53.53	35.58	50.73	48.54
gemma-26B				
C_0T_0	32.40	20.04	32.07	29.76
C_0T_1	31.93	20.00	31.77	29.46
C_1T_0	33.34	20.79	33.25	30.79
C_1T_1	35.93	22.66	35.95	33.28

C.2 Original-benchmark controls

Tables 7 and 8 report frozen-endpoint evaluations on the four original benchmarks. The original full harness supplies C_1T_1; the branch-retention pair uses separately frozen Full and No-Branch endpoints. All configurations remain fixed during evaluation. Reused full-harness predictions are verified against the frozen endpoint. The seed and original evolved endpoint sample 64 frames and make no tool calls; the separate branch pair can use different observations and tools.

We report mIoU differences separately for each original benchmark.

C.3 Search protocol and deployment accounting

Sampling and edit configuration. Each round uses a seeded development batch of 999 queries, balanced across Charades-STA, ActivityNet, and ReXTime training data. The champion and all candidates use the same batch. Validation combines reserved Charades and ReXTime examples, ActivityNet val_1 annotations, and QVHighlights training examples. Sampling manifests specify the eligible videos and queries for each run. Each proposal changes one code or instruction file, with at most 120 changed lines, while preserving the stage interfaces and tool permissions. A code branch is reset after three consecutive rounds without promotion. Near misses are ranked by their development gain relative to the same-round champion; each is eligible for one repair, with at most two repair slots per round.

Selection checks. The development-round winner is compared with the current champion on a video-disjoint validation batch. The gate permits at most a 0.5-point decrease in the validation macro score relative to the champion and to a fixed baseline harness evaluated on the same batch, while checking resource use and execution failures. From round 11, it additionally rejects per-dataset decreases of 0.5 points or more. These tolerance-based checks do not require validation scores to increase at every promotion. A rejected winner can enter the repair pool with aggregate diagnostics; per-example validation content is not exposed to the proposer. The complete candidate is either accepted or rejected, and independently evaluated patches are not merged.

Table 7: Original-benchmark endpoint performance and per-query resource usage.

Harness	R@.5	R@.7	mIoU	Calls	Input tokens (k)	Tools
Charades-STA test						
Seed (C_0T_0)	68.04	48.98	59.88	1.000	4.12	0.000
Evolved (C_1T_1)	70.05	50.78	61.35	1.096	5.32	0.000
Branch-pair Full	70.81	50.97	61.90	1.155	4.90	0.000
No-Branch	66.59	44.54	58.82	2.081	9.84	0.046
ActivityNet val_2						
Seed (C_0T_0)	37.15	23.64	39.85	1.000	2.71	0.000
Evolved (C_1T_1)	40.97	25.75	43.28	1.971	6.87	0.000
Branch-pair Full	41.99	26.54	44.04	2.563	7.00	0.000
No-Branch	41.11	25.82	44.24	2.182	6.48	0.139
QVHighlights val						
Seed (C_0T_0)	61.74	47.55	59.00	0.999	5.30	0.000
Evolved (C_1T_1)	71.68	56.13	66.27	2.052	12.40	0.000
Branch-pair Full	72.32	57.10	67.24	2.687	13.38	0.000
No-Branch	67.81	53.48	64.22	2.632	13.27	0.129
ReXTime val						
Seed (C_0T_0)	51.14	37.68	48.71	0.996	3.24	0.000
Evolved (C_1T_1)	59.17	42.67	53.93	1.948	7.89	0.000
Branch-pair Full	58.96	43.54	54.56	2.474	8.12	0.000
No-Branch	56.89	41.80	53.38	2.233	7.90	0.083

Table 8: Original-benchmark mIoU differences between frozen harnesses.

	Evolved − seed	Branch-pair Full − No-Branch
Charades-STA	+1.47	+3.08
ActivityNet	+3.43	−0.20
QVHighlights	+7.27	+3.02
ReXTime	+5.22	+1.18

Code. We provide the code of our VideoEvolve, including the initial harness and instructions, execution and budget interfaces, proposal generation and checks, search loop, validation gate, and scoring functions. Model and dataset integrations connect through the runtime backend and evaluator interfaces.

Search accounting. The original evolved endpoint improves over its seed by 5.57 macro S points. Table 9 separates proposer usage from search-time solver execution. Cached proposer input is included in input totals, and reasoning output is included in output totals. Solver question counts include repeated development evaluations, not unique examples. Reported times separately sum proposer-process durations and serialized solver evaluation times.

Table 9: Search cost for 60 proposal slots per method, separated into proposer and solver usage.

	Valid	Proposer tokens (M)		Solver	Solver	Wall time (h)	
Search	proposals	Input	Output	queries	calls	Proposer	Solver
Qwen Full	60	26.17	0.349	130,389	217,767	4.19	9.48
gemma Full	60	27.85	0.370	142,977	246,076	3.90	23.24

Deployment usage. For TimeLens-Bench, per-query input tokens in Table 3 divide recorded totals by all 9,404 evaluated queries. ReXTime costs use its 921 evaluated queries as the denominator. Each TimeLens configuration has one failed query without a usage record; it remains in the denominator and contributes zero recorded tokens. Sampled-frame counts measure distinct sampled observations, excluding repeated presentation in subsequent solver calls. Calls per query are rounded to three decimals; the one-call baseline mean is $9403/9404$.

C.4 Controlled branch-retention ablation

Full and No-Branch each use 20 rounds with three proposal slots per round, the same Cloze seed, frozen solver and proposer configurations, and a shared development sampling schedule. No-Branch disables the archive of non-champion code candidates, so code edits start from the champion. It retains historical feedback, the editable code and instruction space, validation gates, and near-miss repair; slots that would continue a branch remain available. Both arms produce 59 valid proposals from 60 slots. The last accepted champion is frozen before endpoint evaluation. These are separate searches from those used for the main results and component exchanges.

The registered primary comparison is Full minus No-Branch macro S over the same 9,404 TimeLens-Bench queries. Subset scores are averaged equally. Each arm has one failed QVHighlights query, retained with zero IoU.

Full averages 2.386 solver calls, 79.92 observed frames, and zero tool calls per query; No-Branch averages 2.470 calls, 80.92 frames, and 0.1675 tool calls. Full achieves its higher score with fewer calls and observed frames in this comparison. The original endpoint remains the reference for component controls: the new Full endpoint differs from it by only $+0.30$ macro S.

Search costs. Full uses 137,772 solver question-evaluations and 263,022 calls; No-Branch uses 150,417 and 304,239. Their serialized solver times are 10.59 and 13.35 hours, with proposer times of 3.56 and 3.94 hours, respectively. Development sampling schedules match across all rounds, while validation samples differ as promotion histories diverge. Evaluations on the original benchmarks use these same two frozen endpoints.

C.5 Case selection and trace coverage

Figure 3 shows a selected improvement and regression from frozen predictions, with the instruction-only configuration alongside seed and full predictions. The successful kettle example is ActivityNet-TimeLens video `v_sRN_crwj3B4`, query 3; the failure is Charades-TimeLens video `Q5MDU`, query 1. Times use the evaluated video's coordinates. The logs provide solver-call counts, the median-consensus path, and final predictions; intermediate intervals are unavailable.

The second improvement illustrates a different pattern: for a QVHighlights-TimeLens query about opening an iron door, C_0T_1 and C_1T_1 both return [132,135] s (IoU 1.0), whereas the seed returns the entire [0,150] s clip (IoU 0.02). The extra call does not change the final prediction in that example. In the high-overlap soccer control, IoU is 0.703 for the seed, 0.973 for C_0T_1, and 0.978 for C_1T_1. These selected cases illustrate the range of observed outcomes.