VideoAgent: All-in-One Framework for Video Understanding and Editing

Hengji Zhou[1,2*], **Lingxuan Huang**[3*], **Jian Wang**[4], **Bing Zhou**[4]
Si Wu[2], **Lianghao Xia**[1†], **Chao Huang**[3]
[1]Harbin Institute of Technology, Shenzhen
[2]South China University of Technology
[3]The University of Hong Kong, [4]Snap Inc.
hengjizhou01@gmail.com, xvrhuang@connect.hku.hk,
jianwang.cmu@gmail.com, zhoubinwy@gmail.com,
cswusi@scut.edu.cn, aka_xia@foxmail.com, chaohuang75@gmail.com

Abstract

Video editing has become essential in digital media creation, yet existing automated systems are restricted to short segment processing and domain-specific tasks. They face two critical limitations: i) inability to handle diverse video comprehension and editing operations, and ii) lack of long-video understanding for coherent narrative creation. We propose **VideoAgent**, an all-in-one agentic framework addressing these challenges through two key innovations. First, we develop automated video shot creation with shot planning agents for coherent narratives and cross-modal retrieval for aligned visual content. Second, we design a multi-agent orchestration framework integrating over thirty specialized editing agents. Intent parsing filters relevant tools while textual-gradient graph optimization assembles complex editing pipelines. Extensive experiments on our newly-proposed **VideoEdit** benchmark and public datasets demonstrate VideoAgent's superiority over existing multimodal LLMs and agentic systems. VideoAgent achieves 87-95% orchestration success rates while reducing API costs by 60%. Human evaluation across six video categories shows VideoAgent produces professional-quality content approaching human-level performance, with ratings only 4% below human-created videos. We release our code at https://github.com/HKUDS/VideoAgent.

1 Introduction

Video editing has become indispensable in modern digital media creation (Esser et al., 2023), as evidenced by the explosive growth of platforms like YouTube and TikTok where millions of creators produce content daily (Chai et al., 2023). However, professional video editing requires complex tools, specialized skills, and intricate multi-step workflows that create significant barriers for ordinary users. This gap between widespread content creation demand and technical accessibility has driven the need for automated video editing systems (Wang et al., 2025) that interpret natural language instructions and execute sophisticated editing tasks without professional expertise.

Recent advances in Large Language Models (LLMs) (Minaee et al., 2024) and Vision Language Models (VLMs) (Zhang et al., 2024b) have made it increasingly feasible to leverage AI for multimodal data understanding and editing. Deep Video Discovery (DVD) employs LLM-coordinated tool ensembles to enable autonomous video exploration through systematic "observe-reason-act" cycles (Zhang et al., 2025b). VideoRAG introduces a dual-channel architecture combining knowledge graphs with multimodal encoding for complex reasoning over long videos (Ren et al., 2025). mPLUG-2 demonstrates modular multimodal foundations using dual visual encoders for unified text, image, and video understanding (Xu et al., 2023b). Despite these pioneering efforts, none have adequately addressed the complex requirements of automated video editing. Existing approaches suffer from two critical limitations: first, they focus on specific understanding or editing tasks and cannot handle the diverse range of operations required in real-world scenarios. Second, they lack the long-video understanding and planning capabilities essential for human-level video editing, preventing the creation of videos with sufficient length and coherent narrative structure.

To address these limitations and achieve human-level automated video creation capabilities, we identify two fundamental challenges:

- **Coherent Long-Form Video Planning**. Professional video creation demands coherent narratives spanning multiple shots and extended durations, requiring sophisticated top-down shot plan-

[1]*Hengji Zhou and Lingxuan Huang have equal contribution to this work.
[2]†Lianghao Xia is the corresponding author.

Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing, pages 21860–21906
October 24-29, 2026 ©2026 Association for Computational Linguistics

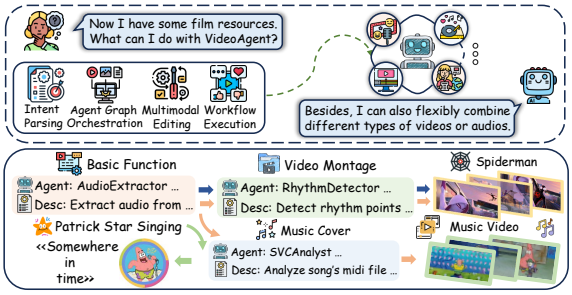


Figure 1: Automated video editing with VideoAgent.

ning with multi-step decomposition and strong reasoning capabilities. For instance, creating a movie trailer requires understanding character arcs, identifying climactic moments, and sequencing emotional beats to build narrative tension—a process that involves complex hierarchical planning from story structure down to individual shot specifications. Such shot planning must be conducted with comprehensive awareness of available visual materials, demanding global understanding and precise retrieval across extensive content libraries. Consider the task of creating a highlight reel from a 2-hour film: the system must possess deep comprehension of both granular visual details and overarching narrative elements to select and sequence appropriate clips. Current approaches predominantly focus on isolated segment understanding and editing, lacking the holistic story creation capabilities essential for professional-quality content.

- **Multi-Agent Workflow Orchestration**. Video editing requires orchestrating diverse tools—from foundational utilities (audio extraction, rhythm detection) to sophisticated creative agents (dialogue creation, narrative structuring)—across modalities with interdependencies. Effective production demands systems that can parse natural language instructions, identify creative patterns, and dynamically compose non-linear workflow networks coordinating multi-modal processing. For instance, creating a music video synchronized to beat drops requires seamlessly integrating audio analysis, rhythm detection, visual retrieval, and temporal alignment, where each component's output feeds subsequent operations. Existing methods are constrained to specific domains or predefined workflows, lacking comprehensive tool libraries and adaptive orchestration capabilities for general-purpose creation.

To tackle these challenges, we propose VideoAgent, an all-in-one agentic framework for video understanding and editing. Our approach first develops an automated video shot creation system that employs planning agents to generate coherent narrative structures, coupled with cross-modal retrieval mechanisms to extract semantically aligned visual content from extensive material libraries. To enable general-purpose creation across diverse genres, we further design a multi-agent orchestration framework that integrates over thirty specialized editing agents through dynamic workflow composition, where intent parsing mechanisms efficiently filter tools and textual-gradient graph optimization adaptively assembles complex editing pipelines.

Our contributions are summarized as follows:

- **All-in-one Agentic Framework**. We propose VideoAgent, a comprehensive agentic framework that unifies video understanding and editing tasks, enabling product-level automated video creation across diverse genres and production workflows.

- **Novel Technical Solutions**. We introduce global-aware shot creation and textual-gradient graph optimization, addressing the coherent long-video planning and multi-agent orchestration challenges that hinder fully automated video editing.

- **Comprehensive Evaluation**. We construct the VideoEdit benchmark with high-quality instructions and human-aligned evaluation. Experiments show VideoAgent's superiority over existing systems, achieving 87–95% success rates while reducing API costs by 60%. Human evaluation shows quality ratings only 4% below professional human-created videos.

2 Methodology

2.1 Video Editing Task Formalization

We formally define the video editing task as follows. **Given i)** a natural language instruction I describing the desired video editing objectives, and **ii)** a collection of multimodal content resources $\mathcal{M} = \{m_1, m_2, \ldots, m_{|\mathcal{M}|}\}$ (*e.g.*, raw video clips, audio tracks, and textual documents for narration), the system **produces** a coherent and semantically aligned edited video V that satisfies the specified instruction I. The output video V comprises a temporally ordered sequence of video shots $V = (v_1, v_2, \ldots, v_{|V|})$, where each shot v_i represents a coherent video segment containing synchronized visual and audio components. In essence, the

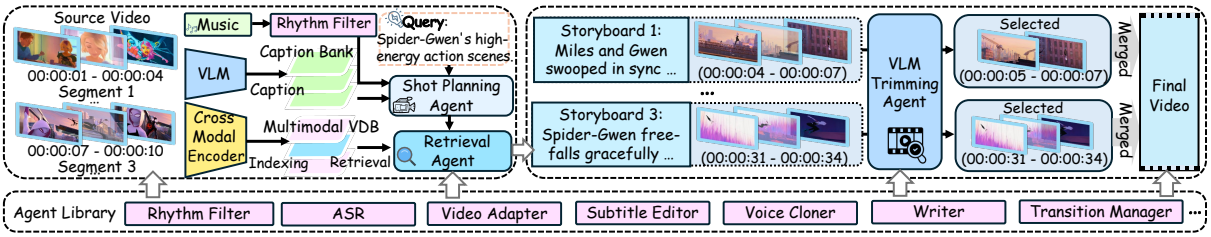


Figure 2: Automated video shot creation with shot planning, video retrieval and trimming.

video editing task is to build:

$$f : (I, \mathcal{M}) \rightarrow V = (v_1, v_2, \ldots, v_{|V|}) \quad (1)$$

2.2 Automated Video Shot Creation

2.2.1 Shot Planning Agent

To generate the video shot sequence $(v_1, v_2, \ldots, v_{|V|})$, we develop a shot planning agent that produces structured shot-level storyboards in natural language. Each storyboard provides a detailed textual description specifying the visual content for the corresponding shot, ensuring all shots collectively form a coherent narrative fulfilling the video creation instruction. This process requires global awareness: the shot planning agent must understand both the overall story structure specified by the user instruction I and the available visual resources $\mathcal{M}$. Formally, the shot planning agent operates as follows:

$$S = (s_1, \ldots, s_{|V|}) = \mathrm{SP}(I, C, |V|; p_s), \quad (2)$$

$$C = \mathrm{LLM}(c_i; p_c)_{i=1}^{|\mathcal{M}|}, \quad c_i = \mathcal{C}(\mathcal{K}(m_i)), \quad (3)$$

where S is the generated shot sequence, C is the compressed visual context summarizing all available materials, c_i is the caption for the i-th visual material obtained by applying a captioning function $\mathcal{C}$ to its keyframe $\mathcal{K}(m_i)$, p_s and p_c are prompts for shot planning and visual context generation respectively, and $|V|$ is the desired number of shots.

The shot planning agent operates through two sequential steps. First, it processes all keyframe captions to generate a compressed visual summary C that captures available video materials. Second, it combines visual summary with the user instruction I to generate the required number of shot descriptions that form the overall narrative structure.

2.2.2 Shot Visual Content Retrieval

Given the shot-level storyboards S, VideoAgent employs paired video indexing and retrieval modules to identify and fetch appropriate video clips for each shot in an efficient manner. The indexing module first extracts keyframes from each video segment using the KeyFrames method mentioned above, then employs cross-modal representation methods (*i.e.*, ImageBind (Girdhar et al., 2023) and CLIP (Radford et al., 2021)) to generate embeddings for these keyframes within a unified text-visual representation space. For retrieval, VideoAgent generates embeddings for each shot's storyboard s_i using the same cross-modal representation approach, then selects the visual content with the highest cosine similarity to the storyboard embedding in the current shot. The indexing and retrieval process are defined as follows:

$$r_i = \arg\max_j \cos(e_j, \mathcal{E}(s_i)), \quad (4)$$

$$e_i = \mathcal{E}(\mathcal{K}(m_i)), \quad i = 1, \ldots, |\mathcal{M}| \quad (5)$$

where r_i represents the retrieved material index for the i-th shot, $e_i = \mathcal{E}(\mathcal{K}(m_i))$ denotes the embedding of the i-th material's keyframe extracted by the cross-modal encoder $\mathcal{E}$, and $\cos(\cdot, \cdot)$ denotes cosine similarity. With this method, our visual content retrieval module fetches semantically aligned visual content to form a coherent narrative following the foregoing text-based shot planning.

2.2.3 Fine-grained Video Trimming

Although the previous modules fetch semantically aligned visual materials, these raw materials are trimmed to a fixed uniform length. In real creative work, shot lengths vary to match the rhythm of background music and dialogue duration (see Listing 9 for shot length decision agents). To address this, VideoAgent introduces a fine-grained video trimming agent that employs VLMs to select the most suitable sub-segment from the retrieved material m_{r_i} based on the current shot's storyboard text, yielding the final visual material for that shot:

$$[t_{\text{start}_i}, t_{\text{end}_i}] = \mathrm{VLM}(m_{r_i}, s_i, t_{i+1} - t_i; p_t), \quad (6)$$

$$v_i = m_{r_i}[t_{\text{start}_i} : t_{\text{end}_i}], \quad i = 1, 2, \ldots, |V| \quad (7)$$

where v_i is the final trimmed video segment for the i-th shot, m_{r_i} is the retrieved raw material, t_{start_i} and t_{end_i} are the start and end timestamps for trimming, t_i denotes the target start timestamp of the

i-th shot in the output video timeline (determined by the shot length decision agent, see Listing 9), $t_{i+1} - t_i$ represents the target duration for the shot, and p_t is the prompt for video trimming guidance. Through this adaptive trimming design, the system achieves precise temporal alignment between visual content and narrative requirements.

2.3 Multi-Agent Graph Orchestration

2.3.1 Multimodal Editing Agent Library

To enhance VideoAgent with advanced video editing capabilities beyond simple concatenation, we develop a multi-agent system for sophisticated video editing operations. This system enables complex creative transformations requiring deep understanding of content semantics, artistic styles, and technical production workflows. The system encompasses two primary categories of tool agents:

- **Foundational multimodal processing tools** that provide essential technical operations such as music beat extraction, multi-emotional voice synthesis, face swapping, lip-sync generation, audio transcription, and visual style transfer.
- **Creation prompts with human expertise** that offer guidance for content generation including style-specific scriptwriting, lyrical composition, and meme-based humor creation. For example, the system can transform English stand-up routines into Chinese crosstalk, produce songs with lyrical and vocal adaptations, generate derivative meme content, or synchronize video cuts with dynamic background music rhythms.

To support broad spectrum of editing requirements, we develop a comprehensive Agent Library spanning diverse creation genres. Notably, the shot creation functionalities (Section 2.2) are also wrapped as tool agents, ensuring all capabilities are **unified within the VideoAgent framework**. Details of all tool agents are provided in Appendix A.9.

2.3.2 Textual-Gradient Graph Optimization

Conventional agent orchestration relies on single-pass graph generation, where LLMs construct workflows through direct reasoning without systematic quality assessment. We reformulate graph construction as an **iterative optimization problem over graph topology space**, where structural deficiencies are progressively corrected through textual gradient descent. This enables principled refinement of graph structures guided by explicit quality metrics rather than implicit heuristics.

Formally, let $A = \{A_1, A_2, \ldots, A_n\}$ denote the agent library. An agent graph is a directed acyclic graph $\mathcal{G} = (\mathbb{A}, E)$ where $\mathbb{A} \subseteq A$ is a subset of selected agents and $E \subseteq \mathbb{A} \times \mathbb{A}$ encodes directed inter-agent dependencies. Given instruction I, an intent parsing mechanism extracts a set of required intents $\mathcal{T}$ that the agent graph must fulfill.

Optimization Objective. We seek an optimal agent graph $\mathcal{G}^*$ that minimizes construction error under instruction I and structural constraints:

$$\mathcal{G}^* = \arg\min_{\mathcal{G} \in \mathbb{G}} \left[\mathcal{L}_{\text{struct}}(\mathcal{G}) + \lambda \mathcal{L}_{\text{align}}(\mathcal{G}, I) \right], \quad (8)$$

where $\mathbb{G}$ denotes the space of all valid directed acyclic graphs over agent library A, $\mathcal{L}_{\text{struct}}$ measures structural validity violations (*e.g.*, cyclic dependencies, disconnected components), $\mathcal{L}_{\text{align}}$ quantifies instruction-graph alignment error, and λ balances structural and semantic objectives. The structural loss is formally defined as:

$$\mathcal{L}_{\text{struct}}(\mathcal{G}) = \alpha(1 - \tau(\mathcal{G})) + \beta \frac{|\Gamma(\mathcal{G})| - 1}{|A|}, \quad (9)$$

where τ is the acyclicity indicator returning 1 if $\mathcal{G}$ admits a valid topological ordering and 0 otherwise, $\Gamma(\mathcal{G})$ denotes the set of connected components in $\mathcal{G}$, and α, β weight the acyclicity and connectivity penalties. The first term penalizes cycles that break workflow executability, while the second penalizes fragmentation into disconnected subgraphs.

Graph Quality Assessment. At iteration t, we evaluate the graph $\mathcal{G}^{(t)}$ through multi-faceted quality metrics. The assessment examines: (i) topological properties via acyclicity validation τ ensuring valid execution orderings, (ii) intent parsing coverage by comparing extracted intents $\mathcal{T}$ against agent capabilities, and (iii) edge consistency through compatibility scoring between connected agents. The quality metrics can be defined as follows:

$$\chi = \mathbb{E}_{ij \in E} \mathbb{I}[\Psi_{ij} \neq \emptyset], \ \mathcal{Q}^{(t)} = (\tau, \kappa, \chi)^{(t)}, \quad (10)$$

$$\kappa(\mathcal{T}) = |\mathcal{T} \cap \textstyle\bigcup_{A_i \in \mathbb{A}} \Psi(A_i)| \cdot |\mathcal{T}|^{-1} \quad (11)$$

where χ measures edge compatibility ratio, averaged over all edges $(A_i, A_j) \in E$ in the graph, κ measures the intent coverage ratio, $\Psi_{ij} = \Psi(A_i) \cap \Psi(A_j)$ denotes shared capabilities between connected agents, and $\mathbb{I}[\cdot]$ is the indicator function. $\Psi : A \to 2^{\mathcal{I}}$ maps each agent to its executable capabilities over intent space $\mathcal{I}$, $\mathbb{A}$ is the set of agents in $\mathcal{G}^{(t)}$, and $\mathcal{Q}^{(t)}$ is the composite quality signal at iteration t. This graph quality assessment produces

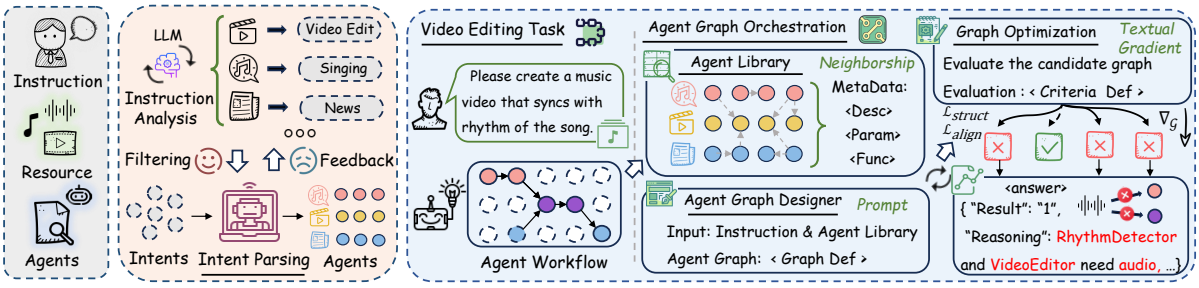


Figure 3: Video editing with agent graph orchestration and execution.

a **structured feedback signal** identifying specific graph deficiencies requiring correction.

Textual Gradient for Graph Topology. To overcome the incompatibility between LLM-driven heuristic refinement and discrete combinatorial graph topologies, inspired by recent works in agent-based optimization (Hu et al., 2024; Zhang et al., 2025a), we introduce **graph-structured textual gradients** that encode topology modifications in natural language. The LLM synthesizes gradient directions by reasoning over: (i) current graph structure $\mathcal{G}^{(t)}$ and quality metrics $\mathcal{Q}^{(t)}$, (ii) instruction requirements I and intent decomposition $\mathcal{T}$, (iii) agent library A specifying capabilities. Formally, the graph update is defined as follows:

$$\mathcal{G}^{(t+1)} = \mathcal{G}^{(t)} \otimes \nabla_{\mathcal{G}}^{\text{text}} \mathcal{L}(\mathcal{G}^{(t)}), \tag{12}$$

where $\otimes$ denotes the graph transformation operator, and the textual gradient is computed:

$$\nabla_{\mathcal{G}}^{\text{text}} \mathcal{L} = \text{LLM}\big(\mathcal{G}^{(t)}, \mathcal{Q}^{(t)}, I, \mathcal{T}, A; p_{\text{topo}}\big), \tag{13}$$

where p_{topo} instructs the LLM to generate graph modifications by: (a) removing edges causing cycles or invalid dependencies, (b) inserting missing agents to cover unaddressed intents, (c) reordering execution sequences to satisfy data flow constraints. This approximates the discrete gradient $\nabla_{\mathcal{G}}[\mathcal{L}_{\text{struct}}(\mathcal{G}) + \lambda \cdot \mathcal{L}_{\text{align}}(\mathcal{G}, I)]$ through symbolic reasoning over discrete graph structures, analogous to combinatorial optimization but realized entirely through natural language transformations.

Convergence Criterion. Optimization terminates when: (1) both losses vanish, *i.e.*:

$$\mathcal{L}_{\text{struct}}(\mathcal{G}^{(t+1)}) = 0 \wedge \mathcal{L}_{\text{align}}(\mathcal{G}^{(t+1)}, I) = 0, \tag{14}$$

or (2) iteration count reaches maximum $T_{\max}$. By recasting agent orchestration as graph-structured textual gradient descent, our method transitions from **generative graph construction**, where graphs emerge from single LLM invocations to **optimization-driven refinement** grounded in explicit structural and semantic objectives.

3 Evaluation

We evaluate VideoAgent through five research questions: **RQ1** overall performance against baselines; **RQ2** ablation of key components; **RQ3** hyperparameter sensitivity; **RQ4** LLM-human judgment consistency; **RQ5** real-world case study.

3.1 Experimental Settings

Datasets and Evaluation Protocols. We evaluate VideoAgent on video understanding and workflow orchestration using two datasets: **Shot2Story** (Han et al., 2025) for video retrieval evaluation, and our new **VideoEdit** benchmark for workflow orchestration with high-quality video creation instructions and human-aligned judge evaluation across diverse video scenarios. See Appendix A.2 for details.

Baseline Methods. VideoAgent is compared with a comprehensive list of baseline methods, including **i) Pure-Language Agentic Systems:** Chain-of-Thought (Wei et al., 2022), LLM Debate (Du et al., 2023), Step Back (Zheng et al., 2023), ExpertPrompting (Xu et al., 2023a), Intelligent Go-Explore(Lu et al., 2024), GPTSwarm (Zhuge et al., 2024), Flow (Niu et al., 2025), Graph-Counselor (Gao et al., 2025) **ii) Multimodal Understanding LLMs and Agents:** Qwen2.5-VL (Bai et al., 2025), VideoRAG (Ren et al., 2025), TeaserGen (Xu et al., 2025), VideoMind (Liu et al., 2026) **iii) Video Generation Agents:** NoteBookLM (Google Labs, 2023), Director (VideoDB, 2024), FunClip (ModelScope, 2024), NarratoAI (linyqh, 2025). Details about baselines are provided in Appendix A.3.

Implementation Details of our VideoAgent and baseline methods are provided in Appendix A.4.

3.2 Overall Performance Comparison (RQ1)

We first compare VideoAgent with baselines on orchestrating effective video editing workflows and video retrieval. As results shown in tables 1 and 2, we observe the following: **Video Edit-**

Table 1: Workflow orchestration performance comparison in terms of ***Success Rate***. Standard deviations are computed over five independent runs.

Backbone	Claude-Sonnet-4		Claude-Sonnet-3.7		GPT-4o		Deepseek-v3	
Data	Audio	Video	Audio	Video	Audio	Video	Audio	Video
CoT	0.78	0.77	0.80	0.72	0.68	0.65	0.69	0.65
Debate	0.66	0.52	0.67	0.66	0.76	0.69	0.61	0.50
Step Back	0.79	0.85	0.68	0.77	0.63	0.38	0.72	0.70
ExpertPrompting	0.71	0.76	0.78	0.73	0.69	0.68	0.75	0.70
Intelligent Go-Explore	0.69	0.63	0.70	0.75	0.88	0.86	0.58	0.53
Flow	0.62	0.64	0.84	0.83	0.68	0.60	0.66	0.61
GPTSwarm	0.69	0.83	0.68	0.86	0.64	0.80	0.73	0.81
GraphCounselor	0.83	0.82	0.85	0.81	0.86	0.84	0.82	0.84
VideoAgent	**0.93** ±0.02	**0.87** ±0.01	**0.95** ±0.03	**0.93** ±0.02	**0.90** ±0.02	**0.88** ±0.01	**0.92** ±0.02	**0.89** ±0.02

Table 2: Video understanding and retrieval performance comparison in terms of Recall@1 (%), Embedding Matching score (EM) (%), Intersection over Union (IoU) (%), API calling Costs and Time Efficiency (s).

Method	Recall	EM	IoU	Cost	Time	Method	Recall	EM	IoU	Cost	Time
Claude-Sonnet-3.7	**46.03**	27.95	23.91	0.374	43	Ours-Claude-Sonnet-3.7	44.27	**28.18**	**24.81**	0.147	37
Claude-Sonnet-3.5	27.28	27.35	12.72	0.375	41	Ours-Claude-Sonnet-3.5	**38.70**	**28.45**	**21.32**	0.147	36
Gemini-2.5-pro	45.98	27.78	**25.91**	0.349	42	Ours-Gemini-2.5-pro	**47.24**	**28.21**	25.74	0.136	37
Gemini-2.5-flash	30.04	28.04	16.69	0.070	39	Ours-Gemini-2.5-flash	**44.93**	**28.25**	**25.07**	0.028	36
GPT-4o	31.22	27.64	18.53	0.253	43	Ours-GPT-4o	**48.85**	**28.26**	**26.99**	0.099	36
VideoRAG	31.03	15.84	14.35	0.100	67	Qwen-2.5-VL-72B-Instruct	18.89	27.99	10.51	-	35
VideoMind-7B	38.26	27.75	19.67	-	21	TeaserGen	41.88	27.62	20.91	0.079	25

ing Orchestration. VideoAgent consistently outperforms baselines across all backbones, achieving success rates of 0.87–0.95 and surpassing the best baseline by 2–25%. The flexible agent graph and gradient-based orchestration effectively handle the multimodal complexity that existing agentic methods struggle with. **Video Understanding.** VideoAgent substantially improves retrieval across foundational models—GPT-4o's Recall rises from 31.22% to 48.85%, Gemini-2.5-flash from 30.04% to 44.93%—via the shot planning agent's global perception that generates coherent storyboards for accurate retrieval, outperforming RAG and agent frameworks in visual-language alignment. **Cost-Efficiency.** VideoAgent reduces API costs by 60% while maintaining superior performance, as strategic shot planning and targeted retrieval minimize redundant processing compared to baselines that feed all candidate shots into context.

To further verify VideoAgent's generalization capability and computational efficiency, we conduct additional experiments on VideoRepurpose (Wu et al., 2025) and a workflow orchestration efficiency study in Appendices A.5 and A.6.

3.2.1 Human Evaluation on Video Quality

We evaluate video quality via human ratings from 54 participants on 18 trending videos (>1M views each), scored on consistency, audio quality, and scene diversity (details in Appendix A.8). Figure 4 compares VideoAgent against four baselines and human-created content. **Outperforming Baselines.** VideoAgent succeeds across all categories where existing methods partially fail, achieving significantly higher scores through effective tool orchestration and accurate visual understanding. **Matching Human Creators.** VideoAgent approaches or exceeds human-created videos in several categories, delivering consistently high quality while human creators exhibit more variable skill levels.

3.3 Ablation Study (RQ2)

We ablate key components on both video shot creation (Figure 5) and workflow orchestration (Table 3). **Shot Planning Agent.** Removing the planner and using raw queries directly causes significant retrieval drops across all metrics, confirming the necessity of global shot planning even with user-provided descriptions. **Cross-Modal Representation.** Replacing our CM Rep. with caption-then-encoding via a text embedding model yields inferior results, validating that our cross-modal paradigm better preserves visual fidelity. **Agent Graph Orchestration.** Removing gradient-based optimization (-G) cuts success rates from >90% to below 55%, highlighting its critical role in handling complex video editing workflows. **Intent Parsing.** While its removal (-I) has limited ac-

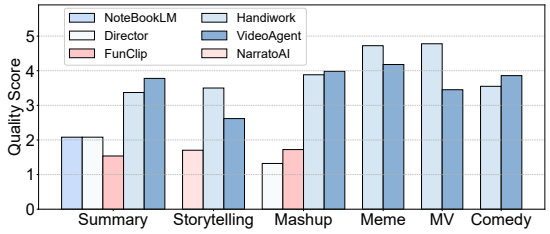


Figure 4: Human-rated video quality assessment.

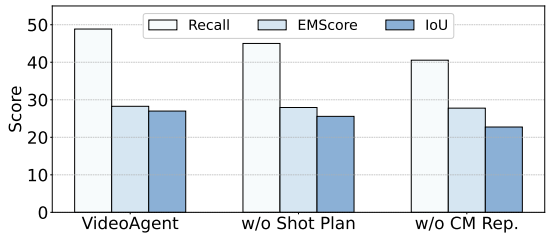


Figure 5: VideoAgent ablation study results.

curacy impact, costs increase notably, confirming its cost-efficiency. **Key Agent Dependencies.** Ablating individual agents (LoudnessNormalizer (LN), AudioExtractor (AE), StandUpSynth (SS)) reveals proportional performance drops correlated with graph centrality, demonstrating effective multi-agent orchestration even in complex scenarios.

3.4 Hyperparameter Study (RQ3)

This section studies the impact of two key hyperparameters in our VideoAgent. The evaluation results are shown in Figure 6. From the results we make the following analysis:

Frames per Segment in shot planning affects the number of key frames extracted for each raw video resource m_i. Using only 1 frame results in a substantial decrease in recall performance, indicating insufficient visual information extraction capability. When the frame count is excessively increased, performance slightly declines, potentially due to over-sampling introducing noisy frames.

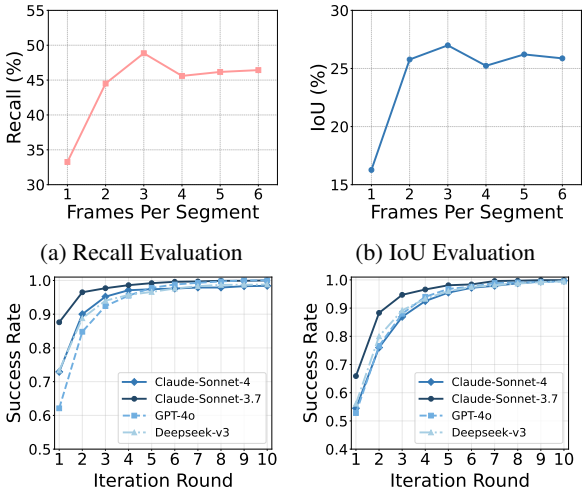
Optimization Round in workflow orchestration shows that increasing Optimization rounds consistently improves the success rate of workflow orchestration, demonstrating typical performance scaling during inference. The results also indicate that sufficient gradient-based graph iterations can significantly reduce performance gaps between different underlying LLM models.

3.5 LLM-Human Judge Consistency (RQ4)

We assess the reliability of LLM-based evaluation by measuring its agreement with human annotations. Specifically, we generate 200 agent graphs via VideoAgent across diverse creation instructions, manually label their success status, and compute

Table 3: Performance of different ablated VideoAgent, in terms of Success Rate (SR) and Cost ($).

Backbone	Claude-Sonnet-3.7				GPT-4o			
Data	Audio		Video		Audio		Video	
Metric	SR	Cost	SR	Cost	SR	Cost	SR	Cost
-I	0.96	0.15	0.95	0.19	0.91	0.15	0.85	0.17
-G	0.70	0.08	0.69	0.12	0.50	0.11	0.68	0.09
-IG	0.58	0.22	0.66	0.11	0.46	0.21	0.66	0.13
-LN	0.84	0.10	0.82	0.03	0.79	0.08	0.77	0.12
-AE	0.65	0.08	0.80	0.07	0.72	0.06	0.83	0.09
-SS	0.04	0.11	0.05	0.03	0.05	0.08	0.04	0.11
Origin	0.95	0.09	0.93	0.05	0.90	0.11	0.88	0.10



(a) Recall Evaluation (b) IoU Evaluation

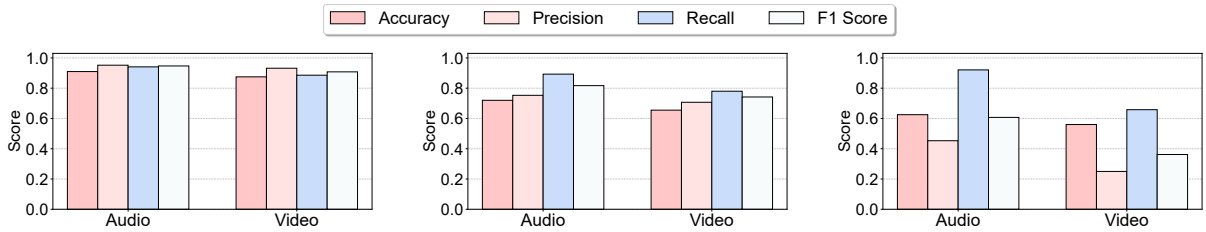
(c) Audio Dataset (d) Video Dataset

Figure 6: Hyperparameter study for VideoAgent.

accuracy, precision, recall, and F1 against human judgments. Results are shown in Figure 7. Claude-Sonnet-3.7 achieves the strongest alignment with human judgment, scoring 0.85–1.0 across all metrics. Across model-modality pairs, recall consistently exceeds precision, indicating that LLM evaluators are more prone to false positives than false negatives. This suggests our framework rarely overlooks genuinely successful orchestrations.

3.6 Case Study on Created Videos (RQ5)

We present a case study to illustrate VideoAgent's practical capabilities. As shown in Figure 8, given an instruction to create a rhythm-synced *Spiderman* montage, the system identifies the high-energy nature of the task and adapts its agent graph for fast-paced editing. It synchronizes visual cuts with musical beats and retrieves target scenes from shot-level descriptions, demonstrating seamless multi-modal integration. This example highlights how VideoAgent dynamically composes agents based on user intent and available resources, transitioning from basic audio extraction to sophisticated rhythm-synced editing within a unified framework.



(a) Claude-Sonnet-3.7 (b) Deepseek-v3 (c) GPT-4o

Figure 7: Consistency study on LLM self-evaluation

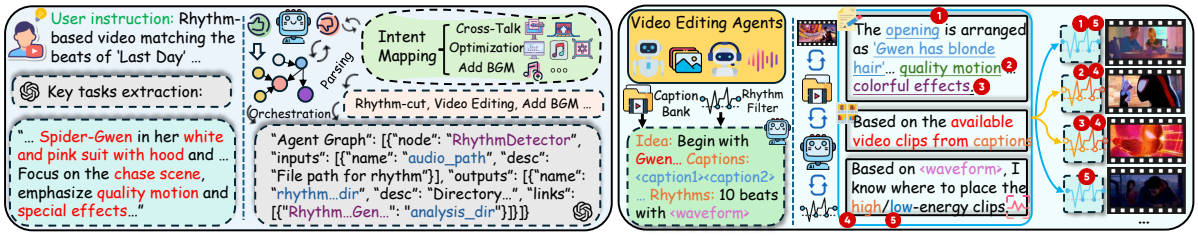


Figure 8: Case study: Creating a rhythm-synced Spiderman movie montage with VideoAgent.

4 Related Work

Automated Multimodal Editing and Generation integrates multiple modalities for content creation and manipulation. Video methods enforce spatiotemporal consistency (Wang et al., 2025), while multimodal LLMs parse complex instructions (Zhang et al., 2024a; Cheng et al., 2025). Recent generative advances (Zhang et al., 2025b; Mao et al., 2026; Ren et al., 2024) produce impressive outputs, yet these results still require substantial editing for real-world scenarios such as vlog creation and story reenactment. While prior work excels at individual editing capabilities, it lacks systematic orchestration of interdependent operations across complex multi-step tasks. We address this gap by integrating over 30 specialized agents with a dynamic graph-based orchestration framework that constructs and refines task-specific graphs.

Multi-Agent Systems. Recent multi-agent research moves from monolithic models to collaborative frameworks, where LLMs use natural language reasoning and tools for role specialization, program synthesis and execution, and interaction with external APIs, databases, and computational resources (Zhuge et al., 2024). GUI agents further pursue this direction through context-aware simplification (Chen et al., 2025), history-context policy optimization (Zhou et al., 2026), and hardness-aware trajectory synthesis (Shao et al., 2026). Existing systems, however, remain confined to text-centric applications such as coding and mathematics, while our work extends this paradigm to multimodal video editing.

Video Understanding and Retrieval are intrinsically connected pillars of visual intelligence, with advances in one domain driving progress in the other. Modern research in video understanding focuses on developing specialized models for challenges like visual question answering (VQA) (Qian et al., 2024; Sima et al., 2024; Li et al., 2025), leveraging complex fusion mechanisms to extract spatial and temporal semantics. Representative works such as VideoMind (Liu et al., 2026) and Qwen-2.5-VL (Bai et al., 2025) demonstrate strong capabilities in long-video scenarios, enabling precise temporal localization and spatio-temporal reasoning. Our work differs by designing a comprehensive framework with strong video understanding and retrieval capabilities specifically tailored for practical video editing workflows.

5 Conclusion

We presented VideoAgent, a comprehensive agentic framework that enables automated video creation across diverse genres and production workflows. Our approach addresses two fundamental challenges: coherent long-form video planning through global-aware shot creation, and multi-agent workflow orchestration via textual-gradient graph optimization. VideoAgent integrates over thirty specialized editing agents and substantially outperforms existing methods, achieving 87-95% orchestration success rates while reducing MLLM API costs by 60%. Human evaluation confirms that VideoAgent produces high-quality videos comparable to human creators across multiple categories. These results establish VideoAgent as an effective solution for bridging the gap between content creation demand and technical accessibility.

Limitations

Our study is an initial exploration of all-in-one agentic video editing, and several aspects warrant further investigation. First, VideoAgent's retrieval-based shot creation inherits the quality constraints of available source footage. Generative video models could complement this pipeline when suitable material is unavailable. Second, VideoAgent has built a community of nearly 300 members, demonstrating its practical value. As the community expands, however, these automated editing capabilities could be misused to create misleading or manipulative content that spreads misinformation.

Ethical Considerations

VideoAgent uses publicly available materials for academic research. Data collection focuses on public content and visible aggregate signals rather than private user information, and we do not collect, infer, or release personal or sensitive data. Resources and outputs are used only for research and evaluation. We respect platform terms, copyright, and content-usage restrictions, and do not intend to redistribute protected source media or support commercial deployment. Beyond these data-use and responsible-use considerations, we do not identify additional ethical concerns specific to this work.

Acknowledgments

This work was supported by the New Generation Artificial Intelligence-National Science and Technology Major Project (2026ZD0127804). The research described herein was completed during the project implementation period.

References

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, and 8 others. 2025. Qwen2.5-vl technical report. *Preprint*, arXiv:2502.13923.

Wenhao Chai, Xun Guo, Gaoang Wang, and Yan Lu. 2023. Stablevideo: Text-driven consistency-aware diffusion video editing. In *ICCV*, pages 23040–23050.

Gongwei Chen, Xurui Zhou, Rui Shao, Yibo Lyu, Kaiwen Zhou, Shuai Wang, Wentao Li, Yinchuan Li, Zhongang Qi, and Liqiang Nie. 2025. Less is more: Empowering GUI agent with context-aware simplification. In *ICCV*, pages 5901–5911. IEEE.

Dabing Cheng, Haosen Zhan, Xingchen Zhao, Guisheng Liu, Zemin Li, Jinghui Xie, Zhao Song, Weiguo Feng, and Bingyue Peng. 2025. Text-to-edit: Controllable end-to-end video ad creation via multimodal llms. *arXiv preprint arXiv:2501.05884*.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*.

Patrick Esser, Johnathan Chiu, Parmida Atighehchian, Jonathan Granskog, and Anastasis Germanidis. 2023. Structure and content-guided video synthesis with diffusion models. In *ICCV*, pages 7312–7322. IEEE.

Junqi Gao, Xiang Zou, YIng Ai, Dong Li, Yichen Niu, Biqing Qi, and Jianxing Liu. 2025. Graph counselor: Adaptive graph exploration via multi-agent synergy to enhance llm reasoning. *arXiv preprint arXiv:2506.03939*.

Rohit Girdhar, Alaaeldin El-Nouby, Zhuang Liu, Mannat Singh, Kalyan Vasudev Alwala, Armand Joulin, and Ishan Misra. 2023. Imagebind one embedding space to bind them all. In *CVPR*, pages 15180–15190. IEEE.

Google Labs. 2023. NotebookLM (AI-powered research assistant). https://notebooklm.google. [Experimental Preview; accessed 2025-08-11].

Mingfei Han, Linjie Yang, Xiaojun Chang, Lina Yao, and Heng Wang. 2025. Shot2story: A new benchmark for comprehensive understanding of multi-shot videos. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Shengran Hu, Cong Lu, and Jeff Clune. 2024. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*.

Zhangbin Li, Jinxing Zhou, Jing Zhang, Shengeng Tang, Kun Li, and Dan Guo. 2025. Patch-level sounding object tracking for audio-visual question answering. In *AAAI*, volume 39, pages 5075–5083.

linyqh. 2025. NarratoAI. https://github.com/linyqh/NarratoAI. [Online; accessed 2025-08-11].

Ye Liu, Kevin Qinghong Lin, Chang-Wen Chen, and Mike Zheng Shou. 2026. Videomind: A chain-of-lora agent for temporal-grounded video reasoning. In *ICLR*, volume 2026, pages 57481–57506.

Cong Lu, Shengran Hu, and Jeff Clune. 2024. Intelligent go-explore: Standing on the shoulders of giant foundation models. *arXiv preprint arXiv:2405.15143*.

Fangyuan Mao, Aiming Hao, Jintao Chen, Dongxia Liu, Xiaokun Feng, Jiashu Zhu, Meiqi Wu, Chubin Chen, Jiahong Wu, and Xiangxiang Chu. 2026. Omni-effects: Unified and spatially-controllable visual effects generation. In *AAAI*, volume 40, pages 7927–7935.

Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. Large language models: A survey. *arXiv preprint arXiv:2402.06196*.

ModelScope. 2024. FunClip. https://github.com/modelscope/FunClip. [Online; accessed 2025-08-11].

Boye Niu, Yiliao Song, Kai Lian, Yifan Shen, Yu Yao, Kun Zhang, and Tongliang Liu. 2025. Flow: Modularized agentic workflow automation. *arXiv preprint arXiv:2501.07834*.

Tianwen Qian, Jingjing Chen, Linhai Zhuo, Yang Jiao, and Yu-Gang Jiang. 2024. Nuscenes-qa: A multi-modal visual question answering benchmark for autonomous driving scenario. In *AAAI*, volume 38, pages 4542–4550.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning transferable visual models from natural language supervision. *Preprint*, arXiv:2103.00020.

Weiming Ren, Huan Yang, Ge Zhang, Cong Wei, Xinrun Du, Wenhao Huang, and Wenhu Chen. 2024. Consisti2v: Enhancing visual consistency for image-to-video generation. *arXiv preprint arXiv:2402.04324*.

Xubin Ren, Lingrui Xu, Long Xia, Shuaiqiang Wang, Dawei Yin, and Chao Huang. 2025. Videorag: Retrieval-augmented generation with extreme long-context videos. *arXiv preprint arXiv:2502.01549*.

Rui Shao, Ruize Gao, Bin Xie, Yixing Li, Kaiwen Zhou, Shuai Wang, Weili Guan, and Gongwei Chen. 2026. HATS: Hardness-aware trajectory synthesis for GUI agents. *arXiv preprint arXiv:2603.12138*.

Chonghao Sima, Katrin Renz, Kashyap Chitta, Li Chen, Hanxue Zhang, Chengen Xie, Jens Beißwenger, Ping Luo, Andreas Geiger, and Hongyang Li. 2024. Drivelm: Driving with graph visual question answering. In *ECCV*, pages 256–274. Springer.

VideoDB. 2024. Director. https://github.com/video-db/Director. [Online; accessed 2025-08-11].

Yukun Wang, Longguang Wang, Zhiyuan Ma, Qibin Hu, Kai Xu, and Yulan Guo. 2025. Videodirector: Precise video editing via text-to-video models. In *CVPR*, pages 2589–2598.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, volume 35, pages 24824–24837. Curran Associates, Inc.

Yongliang Wu, Wenbo Zhu, Jiawang Cao, Yi Lu, Bozheng Li, Weiheng Chi, Zihan Qiu, Lirian Su, Haolin Zheng, Jay Wu, and 1 others. 2025. Video repurposing from user generated content: A large-scale dataset and benchmark. In *AAAI*, volume 39, pages 8487–8495.

Benfeng Xu, An Yang, Junyang Lin, Quan Wang, Chang Zhou, Yongdong Zhang, and Zhendong Mao. 2023a. Expertprompting: Instructing large language models to be distinguished experts. *arXiv preprint arXiv:2305.14688*.

Haiyang Xu, Qinghao Ye, Ming Yan, Yaya Shi, Jiabo Ye, Yuanhong Xu, Chenliang Li, Bin Bi, Qi Qian, Wei Wang, Guohai Xu, Ji Zhang, Songfang Huang, Fei Huang, and Jingren Zhou. 2023b. mPLUG-2: A modularized multi-modal foundation model across text, image and video. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 38728–38748. PMLR.

Weihan Xu, Paul Pu Liang, Haven Kim, Julian McAuley, Taylor Berg-Kirkpatrick, and Hao-Wen Herman Dong. 2025. Teasergen: Generating teasers for long documentaries. In *ICLR*, volume 2025, pages 75558–75578.

David Junhao Zhang, Dongxu Li, Hung Le, Mike Zheng Shou, Caiming Xiong, and Doyen Sahoo. 2024a. Moonshot: Towards controllable video generation and editing with multimodal conditions. *arXiv preprint arXiv:2401.01827*.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. 2025a. Multi-agent architecture search via agentic supernet. *arXiv preprint arXiv:2502.04180*.

Jingyi Zhang, Jiaxing Huang, Sheng Jin, and Shijian Lu. 2024b. Vision-language models for vision tasks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 46(8):5625–5644.

Xiaoyi Zhang, Zhaoyang Jia, Zongyu Guo, Jiahao Li, Bin Li, Houqiang Li, and Yan Lu. 2025b. Deep video discovery: Agentic search with tool use for long-form video understanding. *arXiv preprint arXiv:2505.18079*.

Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H Chi, Quoc V Le, and Denny Zhou. 2023. Take a step back: Evoking reasoning via abstraction in large language models. *arXiv preprint arXiv:2310.06117*.

Xurui Zhou, Gongwei Chen, Yuquan Xie, Zaijing Li, Kaiwen Zhou, Shuai Wang, Shuo Yang, Zhuotao Tian, and Rui Shao. 2026. HiConAgent: History context-aware policy optimization for GUI agents. In *CVPR*, pages 13028–13038.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. 2024. Language agents as optimizable graphs. *arXiv preprint arXiv:2402.16823*.

A Appendix

A.1 Video Showcase

Below are video showcases we created with VideoAgent, which can be grouped into six main categories: **Beat-synced Edits**, **Video Overviews**, **Storytelling**, **Meme Video Edits**, **Song Remixes**, and **Cross-lingual Adaptations** (including stand-up comedy and crosstalk adaptations).

Beat-synced Edits: The user provides background music and film footage. VideoAgent automatically detects beat and tempo changes in the music and aligns high-energy film visual shots with strong beat moments. **Video Overview**: The user provides video footage of a news item or event. VideoAgent transcribes the speech, writes a summary of the event, matches the meaning of each summary sentence to corresponding visual segments from the original video, and generates voiceover for each sentence to complete the edit. **Storytelling**: The user provides plain text of a novel or story along with video footage for visual matching. VideoAgent scripts the story in a specific narration style, matches each line of the script to suitable video segments based on semantic meaning, and generates voiceover for each script segment to produce the final video. **Meme Video Edits**: The user supplies a source video and a custom script or narrative. VideoAgent extracts and transcribes the original audio, generates new speech, precisely syncs the replacement audio to video frames, and outputs a professionally dubbed, meme-style edit. **Song Remixes**: The user provides a MIDI file, lyrics, background music, and a target voice sample. VideoAgent generates a cover, clones the target voice, aligns timing with the arrangement, and integrates the result into the video pipeline for a polished, synced remix. **Cross-lingual Adaptations**: The user provides source audio and target voice samples. VideoAgent adapts the script into the desired cultural format (e.g., Chinese crosstalk or English talk show), synthesizes target voices, applies appropriate effects, and integrates the result into the video editing pipeline for a polished, synced adaptation.

Detailed prompts and examples for each showcase category are presented in Listing 1, Listing 2, and Listing 3.

A.2 Evaluation Datasets and Protocols

To evaluate VideoAgent in terms of both video understanding and workflow orchestration, we conduct experiments from the following two perspectives.

- **Video Understanding and Retrieval Evaluation**. We utilize the **Shot2Story** dataset (Han et al., 2025) to evaluate whether VideoAgent can effectively retrieve appropriate video clips according to user inputs. This dataset contains video clips paired with corresponding queries that describe each clip's content. We randomly select 100 videos with more than 5 queries each, using the midpoint frames as keyframes for retrieval evaluation. In total, there are 599 individual queries. Note that achieving such global understanding and precise retrieval across multiple video segments is computationally intensive. Processing all queries for a multimodal LLM or agent typically requires around 3-9 hours. The distribution of evaluation video durations in our sampled dataset is shown in Figure 12. Videos are grouped into fixed intervals 10–15s, 15–20s, 20–25s, 25–30s, and 30s+ based on their total segment lengths. The caption queries per video distribution in our dataset is illustrated in Figure 13. Each video in the dataset contains between 5 and 8 captions, providing multiple query perspectives for evaluation. The dataset exhibits a diverse range of content categories, with Entertainment, Film & Animation, Autos & Vehicles, Sports , News & Politics, Howto & Style, Science & Technology, Comedy, Education, Travel & Events, People & Blogs, Gaming and Pets & Animals of the sampled videos.

  We use three metrics to compare retrieved video clips and groundtruth shots: **i) Recall:** This is the percentage of groundtruth videos being retrieved with the first rank for each query. **ii) Embedding Matching Score:** Following (Cheng et al., 2025), we compute the coarse-grained embedding matching score between retrieved video content and LLM-generated captions (see List 6) using ImageBind (Girdhar et al., 2023) cross-modal encoder. This evaluates whether retrieved videos align with user intents at a semantic level. **iii) Intersection over Union:** We compute temporal IoU between predicted and ground-truth video segments. IoU measures the overlap between retrieved and target clips, ranging from 0 to 1.

- **Workflow Orchestration Evaluation**. To assess the model's performance in orchestrating work-

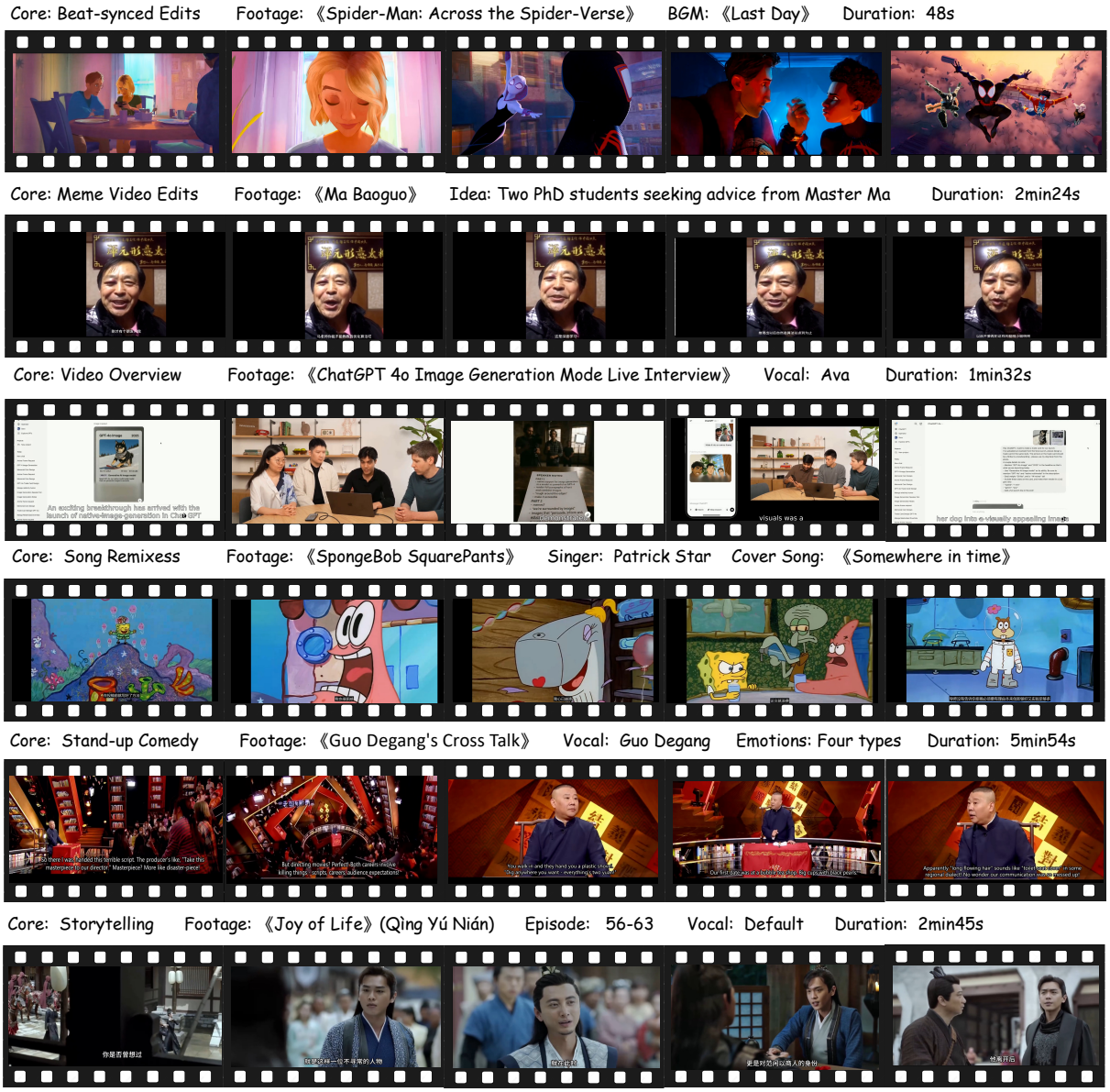


Figure 9: Video cases of VideoAgent in real-world scenarios - Case 1

flows for video editing instructions, we construct the **VideoEdit** benchmark with 2,000 instances, evenly split between audio-oriented and video-oriented editing tasks. We release the complete benchmark in the public repository linked in the paper.

Construction. VideoEdit is designed to reflect practical creator needs rather than relying solely on synthetic task descriptions. We first collect seed requirements from popular videos on Bilibili, a large-scale video-sharing platform with over 100 million users, selecting videos ranked in the weekly top-20 charts with more than one million views. We derive user-oriented requirements from these examples, covering movie and news overviews, storytelling and commentary, meme-style dubbing, beat-synced edits, song remixes, and cross-lingual adaptations. Starting from these creator-oriented seeds, we use controlled LLM-assisted augmentation with Claude-Sonnet-3.7. The augmentation is conditioned on both the seed examples and registered agent metadata, so that generated instructions remain conversational, realistic, and executable within the available tool library. Representative prompts and the audio/video editing system prompts are provided in Appendix A.1 and Listings 4–5.

Annotation Protocol. Each instance is paired with a reference workflow consisting of an agent graph, agent chain, user-input graph, and work-

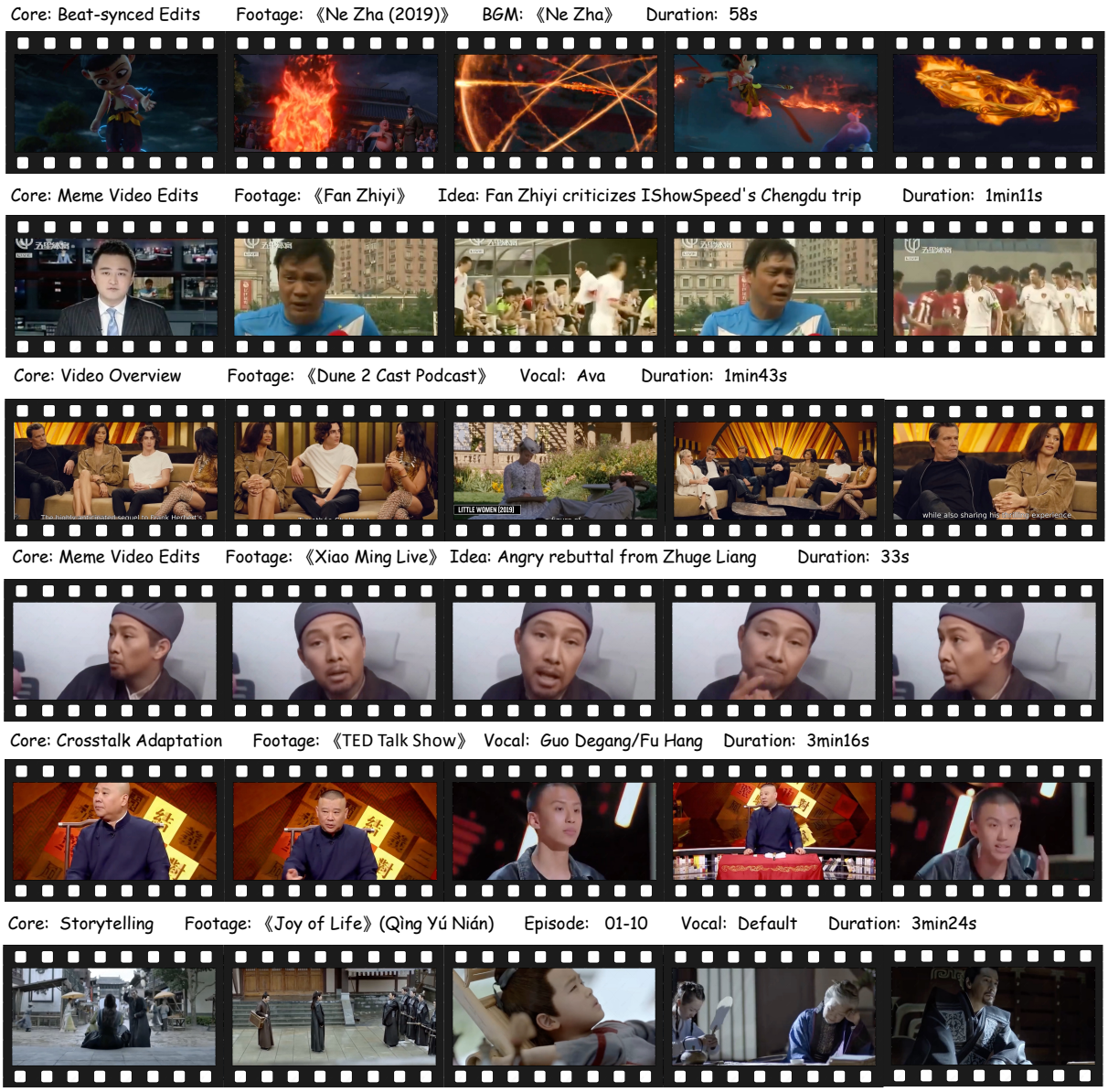


Figure 10: Video cases of VideoAgent in real-world scenarios - Case 2

flow reasoning. We validate each annotation by checking executable order, parameter compatibility, data-flow correctness, redundant-agent avoidance, and coverage of the user requirement. Each sample therefore contains a natural-language user requirement and a corresponding executable graph with agent nodes, input/output parameters, and data-flow links.

Task Diversity. The benchmark covers six major scenarios: **Beat-synced Edits**, **Video Overviews**, **Storytelling**, **Meme Video Edits**, **Song Remixes**, and **Cross-lingual Adaptations**. It spans 31 agents and 187 workflow templates, with graphs ranging from simple two-agent pipelines to complex twelve-agent multi-modal workflows. Within each template, requirements vary in user intent, editing style, and phrasing, providing diverse natural-language realizations of the same underlying workflow pattern.

We employ Claude-Sonnet-3.7 to build a **Judge Agent** that evaluates whether VideoAgent and baseline models successfully orchestrate agent workflows for these complex requirements. This judge agent verifies instruction fulfillment and validates data-flow compatibility, ensuring outputs from preceding agents are properly formatted as inputs for subsequent agents. This evaluation uses **Success Rate** as the metric, indicating the percentage of samples where the orchestrated workflow fulfills the user instruction.

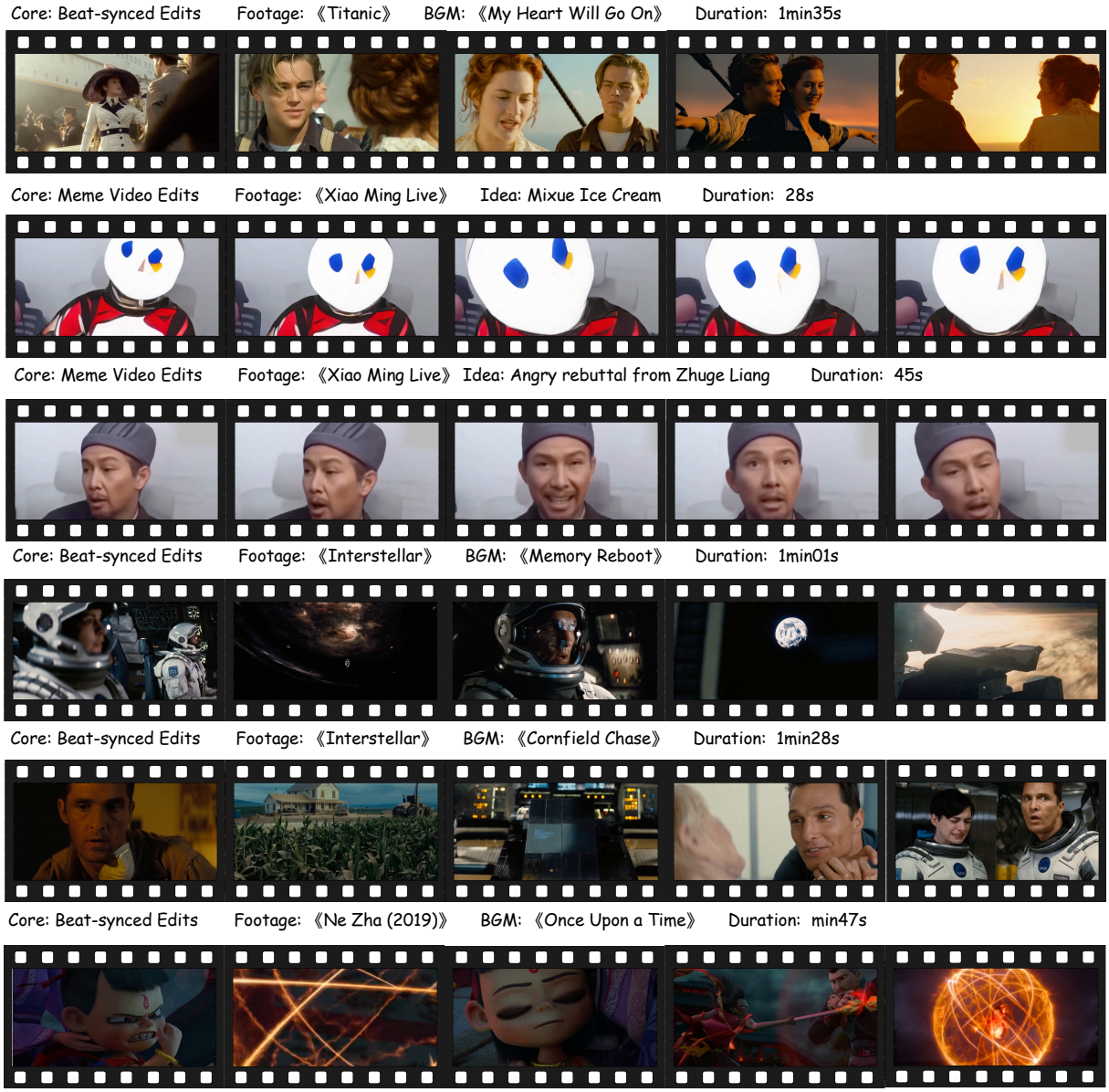


Figure 11: Video cases of VideoAgent in real-world scenarios - Case 3

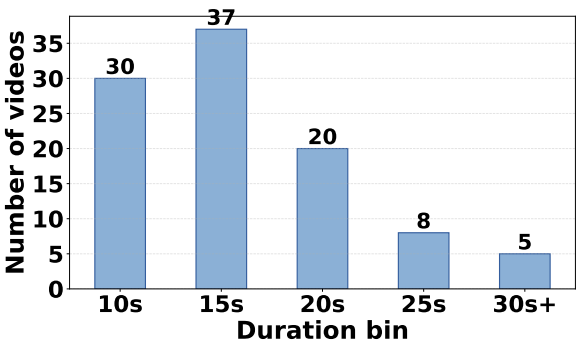


Figure 12: Evaluation video durations distribution.

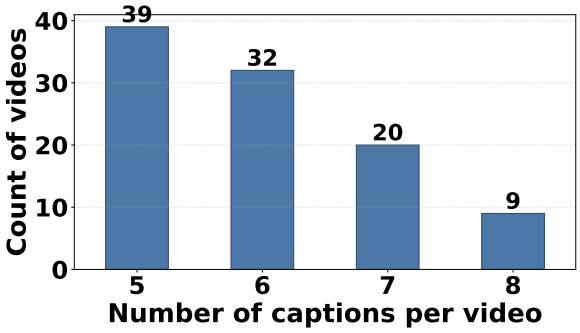


Figure 13: Evaluation video captions/queries distribution.

A.3 Baseline Method

To ensure the comprehensiveness of the benchmark, we conducted a multi-dimensional comparison with n baselines encompassing different mechanisms.

Agentic Systems on Different Domains.

- **Chain-of-Thought** (Wei et al., 2022):This method proposes a chain of thought to improve LLMs' ability to perform complex reasoning by integrating explicit, structured intermediate steps.
- **LLM Debate** (Du et al., 2023): It thoughtfully and systematically integrates debate results from multiple independent language model instances across several iterative rounds of refinement.
- **Step Back** (Zheng et al., 2023): This method enables LLMs to abstract from instances containing specific details to generate high-level concepts and foundational first principles efficiently.
- **ExpertPrompting** (Xu et al., 2023a): It leverages the potential of LLMs by delegating complex tasks to domain-specific expert agents and automatically synthesizing their respective opinions.
- **Intelligent Go-Explore** (Lu et al., 2024): This method is designed to replace handcrafted heuristics with intelligence, iteratively returning to observed states and searching for optimal solutions.
- **Flow** (Niu et al., 2025): This framework introduces modular, parallel workflows to boost efficiency and adaptability, enabling real-time updates that preserve global coherence during challenges.
- **GPTSwarm** (Zhuge et al., 2024): This framework uses graphs to unify LLM agents with reinforcement learning, automatically optimizing prompts and agent orchestration for improved efficiency and adaptability.
- **GraphCounselor** (Gao et al., 2025): This method uses multi-agent collaboration with adaptive graph information extraction to dynamically adjust reasoning depth and improve semantic consistency in specialized domains.

MLLM and Agents.

- **Qwen2.5-VL** (Bai et al., 2025): This vision-language model is designed for long-video comprehension spanning hours, with precise second-level event localization and multimodal reasoning.
- **VideoRAG** (Ren et al., 2025): It introduces a RAG framework for long-context video understanding with graph-grounded and multimodal encoding, enabling unlimited retrieval and beating SOTA.
- **TeaserGen** (Xu et al., 2025): This framework focuses specifically on documentary teaser generation using pre-trained highlight detection modules with local learning-to-retrieval modules.
- **VideoMind** (Liu et al., 2026): This method offers a role-based workflow for temporally grounded video understanding, using Chain-of-LoRA for efficient role-switching via lightweight adapters.

Video Generation Agents.

- **NoteBookLM** (Google Labs, 2023): NotebookLM is a personalized AI knowledge assistant that leverages advanced LLMs to analyze and interact with user-provided documents collaboratively.
- **Director** (VideoDB, 2024): This framework is capable of reasoning through video-related tasks while simultaneously streaming the processed results in real-time to facilitate video synthesis.
- **FunClip** (ModelScope, 2024): FunClip is an open-source tool for automated video clipping. Users can quickly extract video segments by selecting recognized text or speaker segments.
- **NarratoAI** (linyqh, 2025): NarratoAI is a video narration platform, combining script writing, editing, voice-over generation, and subtitles into a unified workflow for content production.

A.4 Implementation Details

This section will provide detailed descriptions of the implementation details and parameter configurations for each baseline in different experiments.

All baseline methods are configured using their default or recommended parameter settings and implemented following the original papers. All experiments were conducted on NVIDIA RTX 3090 GPUs. **Workflow Orchestration Evaluation**: In

the workflow orchestration performance comparison and workflow orchestration ablation study, for fairness, we use Claude-Sonnet-3.7 as the Judge Agent to evaluate Agent Graphs and Agent Chains from multiple perspectives, including execution sequence validation, parameter routing correctness, agent functional redundancy assessment, and requirement fulfillment analysis. We also set the number of reflection rounds to 2 for all self-reflective methods. In the LLM-human judgment consistency experiment, to reduce the error rate of human judgment, we first execute agent graph checks for execution sequence validation and parameter routing correctness issues, then proceed with agent functional redundancy assessment and requirement fulfillment analysis. **Video Editing Evaluation**: All video inputs are segmented into three-second segments, under fps1, one frame is sampled per second. For recall calculations, the midpoint timestamp of each retrieved video segments serves as the reference point. Subsequently, two-second clips are extracted, one second before and one second after the midpoint timestamp, yielding a total duration of two seconds per retrieved clip. These extracted clips are then concatenated sequentially to generate the final output video for computing the Embedding Matching score and Intersection over Union metrics, all ours VLM agent/caption use MiniCPM-V 2.6 int4. In the video editing ablation study and hyperparameter experiments, all our base models utilize GPT-4o, and continue to keep the video segments divided into three seconds for input.

A.5 VideoRepurpose Benchmark Evaluation

To further validate the effectiveness and superiority of our proposed framework, we extended our evaluation to include the VideoRepurpose benchmark (Wu et al., 2025). VideoRepurpose is a large-scale dataset designed for video repurposing from user-generated content, providing a comprehensive testbed for shot retrieval tasks. We randomly sampled 50 videos from the VideoRepurpose dataset, each approximately 1 minute long and containing 1-2 highlight time intervals. The evaluation was conducted following the standard protocol with Recall and Intersection over Union (IoU) as the primary metrics.

Table 4 presents the experimental results comparing our method against several strong baselines, including both commercial models and specialized video understanding models and frameworks. Our VideoAgent framework consistently achieves superior performance across both metrics. These results indicate that VideoAgent achieves the best performance on the VideoRepurpose benchmark, thereby further validating the effectiveness and superiority of the proposed framework across diverse datasets and evaluation scenarios.

A.6 Workflow Orchestration Efficiency Analysis

A.6.1 End-to-End Latency Analysis

We additionally measure the end-to-end (E2E) latency of VideoAgent to characterize its practical efficiency. Here, E2E latency covers the complete video-production process, including workflow orchestration, video understanding and retrieval (U&R), and downstream tool execution. We run five videos for each of the six task categories on a single NVIDIA RTX 3090 GPU. Table 5 reports the average duration of each stage in minutes. The three component times sum to the reported E2E latency (up to rounding).

The results show that orchestration is a lightweight part of the pipeline, remaining below one minute for every category (0.72–0.90 minutes). For long-source-video tasks, U&R is the dominant cost: it accounts for 127.20 of 129.48 minutes for Beat-synced Edits and 49.47 of 54.68 minutes for Cross-lingual Adaptations, reflecting dense source-video indexing, semantic shot retrieval, and fine-grained temporal matching. Video Overviews and Storytelling exhibit the same pattern, with U&R taking 34.17 and 44.60 minutes, respectively.

The bottleneck changes for short audio-centric tasks. Meme Video Edits require only 0.70 minutes for U&R, while tool execution takes 2.63 minutes because speech generation and synchronization dominate. Song Remixes likewise spend most of their time in tool execution (12.68 minutes) due to singing/audio generation, voice conversion, mixing, and synchronization. Overall, multi-agent coordination is not the main runtime bottleneck; practical latency is primarily determined by source-video duration and the media-processing tools selected by the workflow.

A.6.2 Workflow Orchestration Latency Across Backbones

To comprehensively evaluate the computational efficiency of VideoAgent, we conducted detailed latency measurements across different workflow orchestration strategies and backbone models. Ta-

Table 4: Performance comparison on VideoRepurpose benchmark.

Method	Recall	IoU	Method	Recall	IoU
GPT-4o	41.27	19.31	TeaserGen	18.46	16.41
Gemini-2.5-pro	43.12	20.68	Ours-GPT-4o	47.13	19.75
Claude-3.7-sonnet	42.19	17.64	Ours-Gemini-2.5-pro	43.18	19.68
VideoMind-7B	40.11	16.89	Ours-Claude-3.7-sonnet	44.64	19.58

Table 5: End-to-end latency of VideoAgent across six task categories. All durations are averages in minutes over five videos per category, measured on an NVIDIA RTX 3090 GPU. "U&R" denotes video understanding and retrieval.

Task	In.	Out.	E2E	Orch.	U&R	Tool
Video Overviews	32.00	1.53	39.75	0.80	34.17	4.78
Storytelling	39.00	2.85	50.93	0.87	44.60	5.47
Beat-synced Edits	110.00	1.12	129.48	0.72	127.20	1.57
Meme Video Edits	0.58	0.72	4.18	0.85	0.70	2.63
Song Remixes	3.60	2.30	17.57	0.77	4.12	12.68
Cross-lingual Adaptations	43.00	4.37	54.68	0.90	49.47	4.32

ble 6 presents the system latency for workflow orchestration in seconds, comparing four different orchestration approaches across various backbone LLMs, with separate measurements for audio and video processing tasks. The experimental results demonstrate that VideoAgent achieves a balanced trade-off between effectiveness and efficiency. Across different backbone models, VideoAgent consistently maintains competitive latency while delivering superior task completion quality. This balanced approach makes VideoAgent particularly suitable for real-world applications where both efficiency and effectiveness are critical. The results validate that VideoAgent's multi-agent orchestration framework can efficiently coordinate complex video processing workflows while maintaining computational efficiency across diverse backbone models and task types.

A.7 Failure Cases and Boundary Conditions

A.7.1 Representative Failure Cases

To complement the category-level evaluation, Table 7 presents representative failures observed in generated workflows. These cases cover both video and audio tasks and illustrate errors in prerequisite reasoning, agent selection, data routing, and interface compatibility.

These examples show that the remaining workflow failures mainly arise from four sources: incomplete prerequisite reasoning, unsuitable agent selection, invalid data routing, and mismatches between agent input/output schemas and user requirements. For easier inspection and reproducibility, our released repository includes representative generated workflows for both audio and video tasks, with 20 successful and 10 failed cases for each modality.

A.7.2 Boundary Conditions

VideoAgent remains limited under several boundary conditions. First, it depends on accurate source-video understanding and retrieval; consequently, very long, noisy, or semantically ambiguous source videos can result in suboptimal shot selection. Second, tasks requiring strong global creativity, such as storytelling and meme editing, remain challenging because successful outputs require not only a valid workflow but also human-like narrative judgment, humor, pacing, and timing. Third, audio-intensive tasks, including song remixing and rhythm-synced MV creation, are constrained by the quality of the underlying voice-conversion, singing-synthesis, beat-detection, and synchronization tools. Finally, although textual-gradient graph optimization improves workflow validity, it may still fail when an instruction relies on implicit domain knowledge or when multiple agents provide overlapping but subtly different functions.

A.8 Human Evaluation Details

To improve the scale and reliability of the human evaluation, we recruit 54 volunteer participants. 26 participants were recruited from our

Table 6: System latency of workflow orchestration (in seconds) across different backbone models and orchestration strategies.

Backbone	Claude-4		Claude-3.7		GPT-4o		Deepseek-v3	
	Audio	Video	Audio	Video	Audio	Video	Audio	Video
Data Flow	112.31	91.76	103.75	89.36	98.47	102.88	109.18	115.17
GPTSwarm	79.63	101.32	76.84	97.45	43.29	51.57	76.87	60.32
GraphCounselor	24.78	38.43	26.31	33.85	37.02	30.85	45.08	21.46
VideoAgent	38.14	56.73	39.90	53.42	23.21	39.45	50.48	36.67

Table 7: Representative qualitative failure cases of VideoAgent-generated workflows.

Category	Requirement (abridged)	Generated graph issue	Failure	Expected fix
Summary / News Video	Summarize news clips with a specified speaking style and background music.	No Transcriber is placed before NewsContentGenerator.	Extracted speech is not converted into text for summarization.	Add Transcriber and route its transcript to NewsContentGenerator.
Dubbing / Dialogue Replacement	Rewrite video speech while preserving the original speaker and visuals.	Merge is used after TTSInfer.	Merge cannot perform segment-level speech replacement, whereas TTSInfer outputs speech segments.	Use TTSReplace to insert the synthesized segments into the original timeline.
MV / Rhythm-Synced Editing	Cut video scenes to the beats of a user-provided audio track.	Audio is routed to RhythmDetector but not to VideoEditor.	The analyzed beat track may differ from the soundtrack used in the final video.	Route `user_audio_path` to both RhythmDetector and VideoEditor.
Song Remix / Cover Audio	Modify lyrics, use a target vocal style, and mix background music.	SVCAdapter is missing, and Separator outputs are not routed to SVC agents.	Lyrics are not adapted, and the separated audio is not correctly consumed downstream.	Add SVCAdapter and connect the SVC/Separator outputs to the downstream SVC agents.
Audio Preprocessing	Extract audio, normalize its volume, and mix it with background music.	Mixer.audio_path is routed to LoudnessNormalizer.data_dir.	A file-path/directory schema mismatch prevents production of a valid normalized audio path.	Normalize before mixing, or use an edge that returns a concrete normalized audio path.

VideoAgent community group, which contains nearly 300 members with diverse backgrounds, including researchers, content creators, and video-editing hobbyists. We recruited an additional 28 participants online through school media and community groups. All participants voluntarily completed the evaluation and were informed that their ratings would be used for research purposes.

We produced 19 VideoAgent demos; together with the corresponding baseline and human-edited versions, the evaluation contains 49 videos in total. The videos were organized into 18 comparison pairs and uploaded to a shared drive. We used a Google Form to present anonymized files in randomized order. Each participant rated the videos on three criteria—*consistency*, *scene diversity*, and *audio quality*—and spent approximately one hour completing the questionnaire.

As shown in Table 8, VideoAgent obtains the highest total score among the automated systems, and its 95% CI lies clearly above those of all automated baselines. Human-edited handiwork remains stronger overall, mainly because of its higher consistency and scene diversity, while VideoAgent slightly surpasses handiwork in audio quality (4.1 versus 4.0).

We further quantify inter-rater agreement using the intraclass correlation coefficient (ICC) under a two-way random-effects model. The resulting single-rater reliability is $\mathrm{ICC}(2,1) = 0.568$, indicating moderate agreement among individual raters. The reliability of the average over all 54 raters is $\mathrm{ICC}(2,54) = 0.986$, demonstrating that the aggregated human scores are highly reliable.

A.9 Tooluse Agents

To accommodate diverse user needs, we have designed a variety of multimodal tool-use agents capable of handling tasks such as audio preprocess-

Table 8: Human evaluation results across three criteria. The 95% confidence intervals (CIs) for the total score pool ratings across consistency, scene diversity, and audio quality.

Method	FunClip	Director	NotebookLM	NarratoAI	Handiwork	VideoAgent
Consistency	1.9	2.0	1.7	1.1	3.6	3.1
Scene diversity	1.6	1.8	2.1	1.4	4.3	3.7
Audio quality	1.4	1.2	2.4	2.6	4.0	4.1
Total score	1.6	1.7	2.1	1.7	4.0	3.6
95% CI	[1.54, 1.71]	[1.59, 1.81]	[1.92, 2.24]	[1.55, 1.86]	[3.89, 4.04]	[3.56, 3.73]

ing, video overview, storytelling, beat-synced edits, meme video remaking, song remixes, cross-lingual adaptations as well as video edits. The agent names and detailed descriptions of these tools are presented in Table 9.

A.10 Visual Processing Agents

Visual processing agents are specialized components that handle video content analysis, scene caption, and visual element manipulation for multimedia production workflows. These agents bridge the gap between textual content and visual representation by converting visual scenes into actionable narrative descriptions, managing video file operations, and coordinating the integration of visual and audio elements.

A.10.1 VideoPreloader

The VideoPreloader serves as the foundational initialization agent for video processing pipelines, establishing the necessary directory structure and preprocessing operations required for video editing workflows. It integrates with VideoRAG systems to enable efficient video content indexing and retrieval. The detailed algorithm structure is presented in Listing 7.

A.10.2 VideoSearcher

The VideoSearcher retrieves matching video clips from preprocessed video databases based on scene semantic descriptions. It leverages VideoRAG content indexing to perform intelligent video segment matching for automated video assembly workflows. The detailed algorithm structure is presented in Listing 8.

A.10.3 VideoEditor

The VideoEditor performs final video assembly by integrating matched video segments with synchronized audio tracks. It utilizes multimodal analysis to optimize clip selection and ensures temporal alignment between visual content and audio/storyboards narratives. The detailed algorithm structure is presented in Listing 9.

A.10.4 FaceSwapping

The FaceSwapping agent is capable of replacing a specified face in a video with that of a target person. It uses the Viggle AI tool to achieve precise face replacement, enhancing the overall viewing experience of the video. The detailed algorithm structure is presented in Listing 10.

A.10.5 LipSynchronization

The LipSynchronization agent is capable of synchronizing the speaker's lip movements in a video with a target audio track. It uses the Kling AI tool to achieve precise lip-sync alignment, enhancing the realism and coherence of the video's audio-visual experience. The detailed algorithm structure is presented in Listing 11.

A.11 Audio Processing Agents

Audio processing agents constitute a fundamental component of the multimedia production pipeline, handling diverse audio manipulation tasks. These agents leverage advanced signal processing techniques and machine learning models to ensure high-quality audio output while maintaining compatibility across different formats and sampling rates. The audio processing workflow encompasses both basic preprocessing operations and sophisticated content analysis capabilities, with agents designed to handle everything from simple format conversions to complex multi-track audio synthesis.

A.11.1 AudioExtractor

The AudioExtractor agent provides comprehensive audio extraction capabilities from video sources, supporting both single file and batch directory processing. It utilizes FFmpeg for high-quality audio conversion while maintaining consistent output format specifications across all processed files. The detailed algorithm structure is presented in Listing 12.

A.11.2 RhythmDetector

The RhythmDetector agent provides comprehensive music rhythm analysis for rhythm-cut video creation workflows. It employs advanced signal processing techniques using librosa and scipy to detect beat patterns, analyze temporal distributions, and generate precise cut points for video editing synchronization with musical rhythms. The detailed algorithm structure is presented in Listing 13.

A.11.3 LoudnessNormalizer

The LoudnessNormalizer agent provides comprehensive audio loudness normalization capabilities for maintaining consistent audio levels across multiple files. It serves as a critical preprocessing component in audio production workflows, utilizing the FAP (Fast Audio Processing) tool for batch loudness standardization. The detailed algorithm structure is presented in Listing 14.

A.11.4 Merge

The Merge agent provides direct video and audio track combination capabilities without intermediate video clip processing. It distinguishes itself from other multimedia agents by performing complete file merging rather than segmented video editing, utilizing FFmpeg for high-quality encoding and synchronization. The detailed algorithm structure is presented in Listing 15.

A.11.5 Mixer

The Mixer agent provides sophisticated audio mixing capabilities for combining foreground audio with background music tracks. It utilizes PyDub for comprehensive audio processing, supporting dynamic volume adjustment, length synchronization, and high-quality audio overlay operations. The detailed algorithm structure is presented in Listing 16.

A.11.6 Resampler

The Resampler agent provides comprehensive audio resampling capabilities for standardizing sample rates across multiple audio files within a directory. It leverages the FAP (Fast Audio Processing) framework to perform batch resampling operations with real-time progress monitoring and threaded output handling. The detailed algorithm structure is presented in Listing 17.

A.11.7 Separator

The Separator agent provides advanced audio source separation capabilities for isolating vocal and instrumental components from mixed audio tracks. It utilizes machine learning-based separation algorithms through the FAP framework to perform high-quality vocal extraction and background music isolation across multiple audio files. The detailed algorithm structure is presented in Listing 18.

A.11.8 Transcriber

The Transcriber agent provides comprehensive audio-to-text transcription capabilities using advanced speech recognition models. It processes multiple audio files within a directory structure and generates accurate transcription outputs with real-time progress monitoring and robust error handling. The detailed algorithm structure is presented in Listing 19.

A.11.9 VoiceGenerator

The VoiceGenerator agent provides advanced text-to-speech synthesis capabilities for generating high-quality voice audio from scene content. It utilizes the CosyVoice2 model for zero-shot voice cloning and produces synchronized audio with precise timestamp tracking for video editing workflows. The detailed algorithm structure is presented in Listing 20.

A.11.10 TTSInfer

The TTSInfer agent provides sophisticated text-to-speech synthesis for rewritten video content, utilizing sliced audio clips as voice references. It implements a three-stage Fish-Speech pipeline combining VQGAN encoding, text-to-semantic conversion, and audio generation to produce high-quality voice clones matching original speaker characteristics. The detailed algorithm structure is presented in Listing 21.

A.11.11 TTSReplace

The TTSReplace agent provides comprehensive video-audio synchronization for replacing original video audio with derivative synthesized speech segments. It implements precise temporal alignment, video speed adjustment, and seamless audio-video merging to produce coherent rewritten video content while maintaining visual continuity. The detailed algorithm structure is presented in Listing 22.

A.11.12 TTSSlicer

The TTSSlicer agent provides intelligent audio segmentation for preparing audio content for text-to-speech processing and transcription workflows. It implements advanced silence detection algorithms

with RMS-based energy analysis to create optimal audio chunks while maintaining temporal precision and content coherence. The detailed algorithm structure is presented in Listing 23.

A.11.13 SVCAnalyzer

The SVCAnalyzer agent provides comprehensive MIDI file analysis for music cover creation workflows, extracting detailed musical information including note sequences, timing data, and tempo variations. It performs intelligent lyrics-to-music alignment by matching character counts with actual note sequences while handling rest periods and tempo changes. The detailed algorithm structure is presented in Listing 24.

A.11.14 SVCConversion

The SVCConversion agent provides intelligent audio segmentation and timestamp generation for music cover workflows, converting adapted lyrics into precisely timed JSON format suitable for video generation. It handles complex temporal alignment by parsing musical structure markers and generating seamless transition points for multimedia editing. The detailed algorithm structure is presented in Listing 25.

A.11.15 SVCCoverist

The SVCCoverist agent provides advanced voice timbre cloning and conversion capabilities for singing voice synthesis in music cover production. It utilizes the Seed-VC (Voice Conversion) framework to perform source-to-target vocal transformation while maintaining musical characteristics and timing precision for professional-quality audio output. The detailed algorithm structure is presented in Listing 26.

A.11.16 SVCSingle

The SVCSingle agent provides comprehensive singing voice synthesis for music cover production, converting adapted lyrics into high-quality vocal audio with precise temporal alignment. It utilizes the DiffSinger framework for neural singing voice generation and implements sophisticated audio processing techniques to ensure accurate timing and seamless segment concatenation. The detailed algorithm structure is presented in Listing 27.

A.11.17 StandUpConversion

The StandUpConversion agent provides precise temporal mapping for stand-up comedy audio segments, converting individual audio files into structured timestamp format suitable for video generation workflows. It analyzes audio duration and creates synchronized timing data for seamless multimedia editing and video synchronization. The detailed algorithm structure is presented in Listing 28.

A.11.18 CrossTalkConversion

The CrossTalkConversion agent provides precise temporal mapping for crosstalk audio segments, converting dual-performer dialogue into structured timestamp format suitable for video generation workflows. It analyzes audio duration and creates synchronized timing data with performer identification for seamless multimedia editing and video synchronization. The detailed algorithm structure is presented in Listing 29.

A.12 Knowledge Analysis Agents

Knowledge analysis agents process and synthesize multimedia content to generate coherent narratives and creative outputs. They combine video content extraction, narrative summarization, and rhythm-aware storyboard generation to produce structured multimedia knowledge representations. The pipeline integrates semantic understanding, temporal synchronization, and creative synthesis to transform raw content into materials ready for video production workflows.

A.12.1 RhythmContentGenerator with Global-Aware Mechanism

The RhythmContentGenerator extracts video segment content and creates scene-focused narrative summaries incorporating user creative requirements. Its Global-Aware Mechanism enhances video retrieval by refining raw user input into captions-aware queries in two stages: first, building a caption bank from all video content; second, integrating it with rhythm analysis to generate fine-grained, contextually grounded storyboard subqueries. The detailed algorithm structure is presented in Listing 30.

A.12.2 NewsContentGenerator

The NewsContentGenerator creates news summaries via a dual-agent pipeline that processes reference video transcripts and adapts them to user requirements and presentation styles, using a presenter agent for content adaptation and a judger agent for structural formatting. The detailed algorithm structure is presented in Listing 31.

A.12.3 News Presentation Style

This subsection defines the standardized presentation methodology for video event overview content generation, ensuring consistent narrative structure and professional delivery. The complete specification is provided in Listing 32.

A.12.4 CommentaryContentGenerator

The CommentaryContentGenerator creates commentary content from text source materials with formatting tailored for video presentations. It employs a dual-agent architecture to transform source texts into structured video-ready narratives while preserving user-specified creative requirements. The detailed algorithm structure is presented in Listing 33.

A.12.5 Commentary Presentation Style

This subsection provides guidelines for creating engaging narrative content with proper structure, pacing, and audience engagement techniques. The complete writing methodology is presented in Listing 34.

A.12.6 TTSWriter

The TTSWriter agent rewrites transcripts for derivative video content based on sliced audio. It employs a dual-LLM approach: Claude for creative content generation and DeepSeek for precise text extraction, enabling coherent transformation while preserving original speech patterns and timing constraints. The detailed algorithm structure is presented in Listing 35.

A.12.7 SVCAdapter

The SVCAdapter agent adapts lyrics for music cover creation, preserving original melody structure while enabling creative content transformation. It employs multi-stage dual-LLM processing for high-quality lyrical recreation with precise character count alignment. The detailed algorithm structure is presented in Listing 36.

A.12.8 StandUpAdapter

The StandUpAdapter agent transforms reference content into structured stand-up comedy scripts. It employs Claude LLM for creative content generation while preserving professional comedy formatting conventions including tone markers and atmosphere cues. The detailed algorithm structure is presented in Listing 37.

A.12.9 StandUpSynth

The StandUpSynth agent synthesizes audio for stand-up comedy performances, combining text-to-speech generation with audience reaction integration. It employs CosyVoice2 for speech synthesis and DeepSeek LLM for script parsing to produce realistic comedy performances with appropriate vocal delivery and audience responses. The detailed algorithm structure is presented in Listing 38.

A.12.10 CrossTalkAdapter

The CrossTalkAdapter agent transforms reference content into traditional Chinese crosstalk (xiangsheng) dialogue format. It employs Claude LLM for cultural localization and dialogue structure optimization, preserving authentic crosstalk characteristics including role-specific delivery patterns and traditional interactive elements. The detailed algorithm structure is presented in Listing 39.

A.12.11 CrossTalkSynth

The CrossTalkSynth agent synthesizes audio for crosstalk performances, combining dual-performer voice generation with intelligent dialogue analysis. It employs CosyVoice2 for speech synthesis and DeepSeek LLM for script parsing to produce authentic crosstalk performances with role-specific vocal characteristics. The detailed algorithm structure is presented in Listing 40.

A.12.12 VideoConversion

The VideoConversion agent transforms timestamped audio content into visual scene descriptions for video generation. It processes JSON timestamp data and converts narrative content into actionable visual scenes through content analysis. The detailed algorithm structure is presented in Listing 41.

A.12.13 VideoQA

The VideoQA agent analyzes video content through automated transcription and interactive question-answering. It processes multiple videos simultaneously, generates combined transcripts, and enables users to query video content via natural language. The detailed algorithm structure is presented in Listing 42.

A.12.14 VideoSummarizationGenerator

The VideoSummarizationGenerator creates video summaries by processing video files through automatic speech recognition and applying user-specified presentation styles. It combines transcription with content adaptation to generate structured

summaries. The detailed algorithm structure is presented in Listing 43.

A.13 Prompt of Intent Analysis

To reduce memory burden on LLMs and achieve precise intent-tool alignment, we conduct intent analysis on user requirements. The detailed prompt is presented in Listing 44. During the self-reflection phase, VideoAgent adaptively refines its intent analysis based on feedback; the corresponding refinement prompt is shown in Listing 45.

A.14 Graph Construction Rules

We provide a prompt for agent graph construction that guides collaboration among tool-use agents while allowing flexible workflow design beyond predefined pipelines, as presented in Listing 46.

A.15 Graph Evaluation

During textual-gradient graph optimization, we employ a two-step evaluation strategy to maximize accuracy. The initial and secondary evaluation prompts are provided in Listing 47 and Listing 48, respectively.

A.16 Cases of Multi-modal Agent Workflow

To demonstrate VideoAgent's sensitivity to user requirements and its generalization capability, we provide several constructed agent workflow cases. VideoAgent can be extensively customized to tailor workflows to user needs. For example, Listing 49 shows a loudness normalization requirement, with the full agent graph and parameter routing provided in Listing 50. Users can extend this with additional tasks such as vocal separation or background music mixing. As another illustrative case, Listing 51 presents a storytelling video requirement, with the corresponding complete workflow shown in Listing 52.

A.17 Video Editing Evaluation

In our video editing evaluation pipeline, MLLMs and VLMs serve as the core reasoning engines for clip-to-caption matching and automated content analysis, with MiniCPM-V-2.6 as the backbone.

Clip-to-Caption Matching. Given video clips at three-second intervals and textual captions, the VLM identifies the best-matching clip for each caption. It receives a text prompt with visual frames (1 fps) embedded as base64 in the multimodal message, and analyzes spatial features, object composition, scene layout, and temporal coherence to output a structured JSON with selections and reasoning — achieving fine-grained semantic alignment beyond keyword-based retrieval.

Frame Integration Strategy. Each three-second clip contributes three frames, appended sequentially after the text prompt for joint reasoning over both modalities. For longer videos, frames from all clips are packed into one multimodal request to preserve global context. The full protocol is detailed in Listing 53.

Automated Content Analysis. Beyond clip matching, the VLM assesses retrieved segments along three dimensions: semantic relevance to the user requirement, visual diversity across clips, and temporal coherence. This pipeline enables scalable, reproducible evaluation across the 2,000-sample VideoEdit benchmark, with results validated against human judgment (see Section 3.5).

A.18 Artifact Licenses and Intended Use.

Our newly constructed VideoEdit benchmark and the released VideoAgent code are intended for non-commercial research on multimodal video understanding, video editing, and agentic workflow orchestration. The artifacts should not be used for impersonation, deceptive editing, misinformation generation, or other rights-infringing or harmful content creation.

Listing 1: VideoAgent Showcase 1 with Prompts

```
Categories:
1. Beat-synced Edits
   Description:
     - User provides background music and film footage.
     - Detects beat/tempo changes and aligns high-energy shots to strong beats.
   Prompt:
    Begin with Gwen, who has blond hair, sitting at a dining table in front of a window, then transition to her playing drums
with pop textures and musical notes in the background. Include action sequences featuring Miguel O'Hara in a dark blue suit with
 red accents, sharp red claws, and black-and-red eye lenses; Spider-Gwen in a white-and-pink suit with a hood and ballet shoes;
Miles Morales with curly hair and a red spider logo on his chest; and the Spot in a black suit covered in white spots using
portal powers. Focus on the chase scene against a blue sky with trains, and emphasize high-quality motion throughout--web-
swinging, combat, and vibrant special effects.


2. Video Overviews
   Description:
     - User provides a news/event video.
     - Transcribe speech, summarize, align summary sentences to visuals, generate voiceover.
   Prompt:
     - Short tech news, colloquial expression within 250 words, check the accuracy of key terms, e.g. the GPT model name should
be 4o instead of 4.0

3. Storytelling
   Description:
     - User provides story/novel text and footage.
     - Script in chosen narration style, match lines to visuals, generate voiceover.
   Prompt:
     - A verbal interpretation copy of no less than 1,000 words

4. Meme Video Edits
   Description:
     - User supplies a source video and custom script.
     - Extract/transcribe audio, generate new speech, sync precisely to frames, output dubbed edit.
   Prompt:
     - Create a humorous narrative about two PhD students seeking advice from Master Ma. For the two PhD students, one of them
is known for high citation counts and the other for numerous publications. Transform martial arts terms into AI research
terminology while keeping phrase lengths similar (length difference should be less than two Chinese characters). The story
highlights their academic rivalry and ends with Master Ma advising against "internal competition". Keep signature phrases like "
wasn't cautious enough" and "achieving great results with minimal effort" while avoiding mentions of real institutions. The word
 combinations should be logical and appropriate for an academic context.

5. Song Remixes
   Description:
     - User provides MIDI, lyrics, background music, and a target voice sample.
     - Generate cover, clone voice, align timing, integrate into video pipeline.
   Prompt:
     - The song is performed by Patrick Star, focusing on the theme of "the struggles of manuscript submission and dealing with
overly critical reviewers", following the original lyrics' sentence structure while replacing specific content. It incorporates
elements of reviewer nitpicking (e.g., questioning innovation, demanding redundant experiments) and expresses frustration with
lines like "If only I could swap reviewers, this academic fate is too cruel" to highlight the emotional toll of peer review.

6. Cross-lingual Adaptations
   Description:
     - User provides source audio and target voice samples.
     - Adapt script to cultural format (e.g., crosstalk/talk show), synthesize voices, apply effects, integrate into pipeline.
   Prompt:
     - Adapting to the cultural context of talk shows and localizing humor
```

Table 9: The detailed information of tooluse agents.

Agent Name	Type	Description
AudioExtractor	Audio Preprocessing	To extract audio from a single video or all videos in a directory
LoudnessNormalizer	Audio Preprocessing	Audio loudness normalization tool
Merge	Audio Preprocessing	To merge video and audio tracks
Mixer	Audio Preprocessing	To mix audio with the BGM
Resampler	Audio Preprocessing	Audio Resampling tool
Separator	Audio Preprocessing	Audio source separation tool
VoiceGenerator	Speech Generation	To generate speech based on scene content.
CrossTalkAdapter	Cross-lingual Adaptations	Adapt a reference script into segmented cross talk.
CrossTalkSynth	Cross-lingual Adaptations	Segment-by-segment cross talk audio synthesis with final merge.
StandUpAdapter	Cross-lingual Adaptations	Adapt a reference script into segmented stand-up comedy.
StandUpSynth	Cross-lingual Adaptations	Segment-by-segment stand-up comedy audio synthesis with final merge.
SVCAdapter	Song Remixess	Adapt the original lyrics.
SVCAnalyst	Song Remixess	Analyze the original song's MIDI file to extract information such as notes, note durations, etc.
SVCAnalyst	Song Remixess	Analyze the original song's MIDI file to extract information such as notes, note durations, etc.
SVCCoverist	Song Remixess	Source-to-target voice timbre cloning for singing voice synthesis.
SVCSingle	Song Remixess	Split the adapted lyrics into segments, then synthesize each segment with the default vocal singing voice.
TTSInfer	Meme Video	Take the sliced audio clips as target voice references, then conduct Text-To-Speech synthesis using their rewritten text segments.
TTSReplace	Meme Video	Replace audio of the original video with derivative audio segments from the sliced clips.
TTSSlicer	Meme Video	Slice the audio into segments. The audio segments often need to be transcribed afterward.
TTSWriter	Meme Video	Rewrite the transcript of the sliced audio segments based on user requirements.
CommentaryContent Generator	Storytelling	To generates commentary content based on user ideas and text source materials with specialized formatting for video presentations.
NewsContentGenerator	Video Overview	To generate news summary based on user ideas and reference materials.
RhythmDetector	Beat-synced Edits	To create cut points for video editing based on music rhythms.
RhythmDetector	Beat-synced Edits	To create cut points for video editing based on music rhythms.
RhythmContent Generator	Beat-synced Edits	To extract video segment content, create scene-focused narrative summaries, and generate rhythm-aware storyboards.
VideoConversion	Video Edits	converts audio content with JSON timestamps into visual scene descriptions for video generation.
VideoEditor	Video Edits	Ultimately merging the clips and adding audio.
VideoPreloader	Video Edits	To initialize environment and preprocess video files.
VideoSearcher	Video Edits	To retrieve matching video clips based on timestamp file.
VideoQA	Video Interaction	Transcribes all videos in a directory and provides interactive Q&A session.
VideoSummarization Generator	Video Interaction	Agent that generates summarization content based on user ideas and reference materials.
FaceSwapping	Video Processing	Replace the faces appearing in the video with those of the target person.
LipSynchronization	Video Processing	Synchronize the speaker's lip movements in the video with the audio track.

Listing 2: VideoAgent Showcase 2 with Prompts

```
Categories:
1. Beat-synced Edits
   Description:
     - User provides background music and film footage.
     - Detects beat/tempo changes and aligns high-energy shots to strong beats.
   Prompt:
     - Conflicts between Nezha, Dragon Prince Ao Bing (blue-robed and blue hair) and Shen Gongbao (black-robed).

2. Video Overviews
   Description:
     - User provides a news/event video.
     - Transcribe speech, summarize, align summary sentences to visuals, generate voiceover.
   Prompt:
     - Short movie podcast, colloquial expression within 300 words; identify which actor or host is speaking; do not mention
movie-ticket availability.

3. Storytelling
   Description:
     - User provides story/novel text and footage.
     - Script in chosen narration style, match lines to visuals, generate voiceover.
   Prompt:
     - Write a fluent commentary script of 1,500 words.

4.1 Meme Video Edits (Fan Zhiyi)
   Description:
     - User supplies a source video and custom script.
     - Extract/transcribe audio, generate new speech, sync precisely to frames, output dubbed edit.
   Prompt:
     - Emphasize the positive impact of IShowSpeed's China tour, which is not only an online carnival but also a milestone event
 in cultural communication in the digital age.

4.2 Meme Video Edits (Xiao Ming Live)
   Description:
     - User supplies a source video and custom script.
     - Extract/transcribe audio, generate new speech, sync precisely to frames, output dubbed edit.
   Prompt:
     - Based on the following scenario, create an angry rebuttal from Zhuge Liang (me):
       - Speaker: Zhuge Liang (me)
       - Start with "Why don't you look at your own problems for the failure," followed by "...look at your own problems"
pattern sentences that all reference anime events
       - Anime examples must mention specific characters
       - Only the last "...look at your own problems" should return to the failure scenario
       - Use colloquial language and diverse anime references

5. Cross-lingual Adaptations
   Description:
     - User provides source audio and target voice samples.
     - Adapt script to cultural format (e.g., crosstalk/talk show), synthesize voices, apply effects, integrate into pipeline.
   Prompt:
     - To generate a crosstalk-style piece, the story must conform to objective reality and be about forty to fifty sentences.
```

Listing 3: VideoAgent Showcase 3 with Prompts

```
Categories:
1.1 Beat-synced Edits (Titanic)
   Description:
     - User provides background music and film footage.
     - Detects beat/tempo changes and aligns high-energy shots to strong beats.
   Prompt:
     - A romantic and sweet love story about Jack and Rose meeting on the Titanic. It cannot include the part where the ship is
in distress, nor the night scene. In the first section, Rose, wearing a purple hat and a white shirt, walks out of a white car
with a purple umbrella, looking thoughtfully.

1.2 Beat-synced Edits (Interstellar)
   Description:
     - User provides background music and film footage.
     - Detects beat/tempo changes and aligns high-energy shots to strong beats.
   Prompt:
     - Celebrate humanity's courage in space exploration. Include scenes featuring spaceships, wormholes, black holes, space
station docking maneuvers, ocean planets, and glacial worlds. Show astronauts in their distinctive white spacesuits as they
venture into the unknown, highlighting humanity's drive to explore the cosmos.

1.3 Beat-synced Edits (Interstellar)
   Description:
     - User provides background music and film footage.
     - Detects beat/tempo changes and aligns high-energy shots to strong beats.
   Prompt:
     - Love can transcend time and space.

1.4 Beat-synced Edits (Ne Zha)
   Description:
     - User provides background music and film footage.
     - Detects beat/tempo changes and aligns high-energy shots to strong beats.
   Prompt:
     - Capture more scenes of conflicts and battles between Nezha and Shen Gongbao (black-robed) and Dragon Prince Ao Bing (blue-
robed).

2.1 Meme Video Edits (Xiao Ming Live)
   Description:
     - User supplies a source video and custom script.
     - Extract/transcribe audio, generate new speech, sync precisely to frames, output dubbed edit.
   Prompt:
     - Background: Mixue Ice Cream is a national chain brand focusing on ice cream and tea beverages. On March 15 (Consumer
Rights Day), they were reported to be using overnight lemons. However, compared to other exposures, using overnight lemons is
not considered a particularly serious violation and is somewhat understandable.
     - Speaker: Snow King (Mixue's representative)
     - Purpose: Emphasize that the "overnight lemon" situation is not too serious, highlighting Mixue's good reputation
     - Must preserve the phrases "Look in my eyes tell me why" and "tell me"
     - Must end with the word "say it"
     - If the original text contains awkward phrasing, such as redundant words or confused semantics, do not imitate that style
or sentence structure
     - Ensure natural and fluent sentences

2.2 Meme Video Edits (Xiao Ming Live)
   Description:
     - User supplies a source video and custom script.
     - Extract/transcribe audio, generate new speech, sync precisely to frames, output dubbed edit.
   Prompt:
     - Based on the following scenario, create an angry rebuttal from Zhuge Liang (me):
       - Speaker: Zhuge Liang (me)
       - Zhuge Liang (me) is challenged about why a certain Three Kingdoms character has a higher rating than him and launches a
 fierce rebuttal
       - Later rating comparisons should show stark differences (can be exaggerated)
       - Use colloquial language, align with historical facts, and only replace specific content
```

Listing 4: Audio Editing Instruction Prompt

```
You are an Autonomous Agent System Designer. I will provide Registered Agents (Name, Description and Parameters information):

Your task is to:
1. Generate Feasible User Requirement:
   - Only consider generating feasible User Requirement for **audio-related** aspects (Finally generate audio instead of video)
based on metadata of Registered Agents.
   - Avoid mentioning specific agent names in User Requirement, but clearly describe the needs.
   - Avoid mentioning the concrete steps and details of implementation.
   - Reflect real-world needs, and be phrased conversationally.
2. Design Executable Agent Graph:
   - Format: List
   - Agent Graph shall contain metadata for each Agent Node including:
     * name: (string)
     * inputs: (list of input parameter objects with):
       - parameter: input parameter name
       - description: brief parameter description
     * outputs: (list of output parameter objects with):
       - parameter: output parameter name
       - description: brief parameter description
       - links: (list of dictionaries) where each dictionary specifies:
         - key of dictionaries: target agent name
         - value of dictionaries: target agent's input parameter name that this output connects to
   - Agent Graph case:
       [
          {
           "node": Agent Name,
           "inputs": [
                {
                "name": input parameter name1,
                "description": ...
                },
                {
                "name": input parameter name2,
                "description": ...
                },
              ]
           "outputs": [
                {
                "name": output parameter name1,
                "description": ...,
                "links": [
                    {"next_agent1": next_agent1's input parameter name"},
                    {"next_agent2": next_agent2's input parameter name"}
                    ]
                },
                {
                "name": output parameter name2,
                "description": ...,
                "links": []
                },
                ...
              ]
           },
           ...
       ]
3. Generate Agent Chain:
   - Format: List
   - Based on the description of the Agent and the sequential information contained in the designed Agent Graph, generate the
Agent Chain
   - Agent Chain case:
        ["agent1", "agent2", ...]
4. Generate User Input Graph
   - Format: List
   - Parameter nodes with no in-degree (no incoming edges) are uniformly considered to require user input.
   - Parameter nodes with no in-degree may have different names but share the same user input, meaning a single user input
parameter can point to multiple such nodes.
   - Parameter nodes with no in-degree that are linked to user input should be represented in the format **AgentName.
input_parameter**
   - Generate the User Input Graph based on the Agent descriptions and parameter passing information in the designed Agent Graph.

   - User Input Graph case:
       [
         {
          "node": User input parameter name,
          "description": brief description of the parameter
```

Listing 4 continued: Audio Editing Instruction Prompt.

```
          "links": [
                    {"agent1": agent1.input_parameter1},
                    {"agent2": agent2.input_parameter2}
                ]
         },
         ...
       ]
5. Output Reasoning:
   - Provide concise reasoning (<200 words) explaining the entire workflow logic

In addition to the above formatting requirements, please also note the following:
1. For each element of **outputs** in each Agent Node:
   - Ensure that the **links** in the **outputs** point to an input parameter that actually exists in the next Agent Node.
   - Also, make sure the description of the output parameter matches the description of the input parameter in the next Agent
Node.
2. Final JSON Output Format Specification:
{
    "User Requirement": ...,
    "Agent Graph": ...,
    "Agent Chain": ...,
    "User Input Graph": ...,
    "Reasoning": ...
}
Strictly follow JSON output format!

Metadata of registered agents:
{registry_agents}

Real-life Requirement Examples:
{real-life_examples}
```

Listing 5: Video Editing Instruction Prompt

```
You are an Autonomous Agent System Designer. I will provide Registered Agents (Name, Description and Parameters information):

Your task is to:
1. Generate Feasible User Requirement:
   - Only consider generating feasible User Requirement for **video-related** aspects based on metadata of Registered Agents.
   - Avoid mentioning specific agent names in User Requirement, but clearly describe the needs.
   - Avoid mentioning the concrete steps and details of implementation.
   - Reflect real-world needs, and be phrased conversationally.
2. Design Executable Agent Graph:
   - Format: List
   - Agent Graph shall contain metadata for each Agent Node including:
     * name: (string)
     * inputs: (list of input parameter objects with):
       - parameter: input parameter name
       - description: brief parameter description
     * outputs: (list of output parameter objects with):
       - parameter: output parameter name
       - description: brief parameter description
       - links: (list of dictionaries) where each dictionary specifies:
         - key of dictionaries: target agent name
         - value of dictionaries: target agent's input parameter name that this output connects to
   - Agent Graph case:
       [
          {
           "node": Agent Name,
           "inputs": [
                {
                "name": input parameter name1,
                "description": ...
                },
                {
                "name": input parameter name2,
                "description": ...
                },
              ]
           "outputs": [
                {
                "name": output parameter name1,
                "description": ...,
                "links": [
                    {"next_agent1": next_agent1's input parameter name"},
                    {"next_agent2": next_agent2's input parameter name"}
                    ]
                },
                {
                "name": output parameter name2,
                "description": ...,
                "links": []
                },
                ...
              ]
           },
           ...
       ]
3. Generate Agent Chain:
   - Format: List
   - Based on the description of the Agent and the sequential information contained in the designed Agent Graph, generate the
Agent Chain
   - Agent Chain case:
        ["agent1", "agent2", ...]
4. Generate User Input Graph
   - Format: List
   - Parameter nodes with no in-degree (no incoming edges) are uniformly considered to require user input.
   - Parameter nodes with no in-degree may have different names but share the same user input, meaning a single user input
parameter can point to multiple such nodes.
   - Parameter nodes with no in-degree that are linked to user input should be represented in the format **AgentName.
input_parameter**
   - Generate the User Input Graph based on the Agent descriptions and parameter passing information in the designed Agent Graph.

   - User Input Graph case:
       [
         {
          "node": User input parameter name,
          "description": brief description of the parameter
```

Listing 5 continued: Video Editing Instruction Prompt.

```
          "links": [
                    {"agent1": agent1.input_parameter1},
                    {"agent2": agent2.input_parameter2}
                ]
         },
         ...
       ]
5. Output Reasoning:
   - Provide concise reasoning (<200 words) explaining the entire workflow logic

In addition to the above formatting requirements, please also note the following:
1. For each element of **outputs** in each Agent Node:
   - Ensure that the **links** in the **outputs** point to an input parameter that actually exists in the next Agent Node.
   - Also, make sure the description of the output parameter matches the description of the input parameter in the next Agent
Node.
2. Final JSON Output Format Specification:
{
    "User Requirement": ...,
    "Agent Graph": ...,
    "Agent Chain": ...,
    "User Input Graph": ...,
    "Reasoning": ...
}
Strictly follow JSON output format!

Metadata of registered agents:
{registry_agents}

Real-life Requirement Examples:
{real-life_examples}
```

Listing 6: Video Caption to High-Level Summary Transformation Prompt

```
prompt = f"""
Transform the following detailed video caption into a concise, high-level summary that captures the main themes, setting, and
overall atmosphere in no more than 50 words, no point form, use pure sentence.

Focus on:
- Setting identification: Where does the action take place?
- Key themes: What are the main activities or purposes being shown?
- Atmosphere and tone: What mood or feeling does the scene convey?
- Overall narrative: What story or message emerges?

Instructions:
1. Condense specific details into broader concepts
2. Identify recurring elements that suggest themes
3. Synthesize individual actions into a cohesive narrative about the environment
4. Use descriptive language that captures essence rather than listing events
5. Maintain the core message while abstracting from granular details
6. Preserve emotional tone and workplace/environmental culture
7. Keep response under 50 words

Input format: whole_caption"[detailed description]"
Output format: high_level_caption"[concise thematic summary]"

Rules:
- Extract thematic elements from specific actions
- Focus on overarching narrative rather than individual events
- Capture the purpose
- Ensure output maintains fidelity to original content intent

Transform the provided detailed caption following these guidelines."""
```

Listing 7: VideoPreloader Structure

```
Input: video_directory
Output: preprocessing_status

Algorithm:
1. Configure multiprocessing (spawn), logging, project paths; add modules to system path.
2. Create directory structure: audio\_analysis, scene\_output, videosource-workdir, writing\_data, video\_output; set VideoRAG
working directory.
3. Validate video source directory; scan for MP4 files; report discovery statistics.
4. Dynamically import VideoRAG and QueryParam for content indexing; handle import errors with fallbacks.
5. Execute video insertion via VideoRAG: extract metadata/features, generate content embeddings, build searchable index database.

6. Report processing statistics and return execution status for downstream pipeline coordination.

Error Handling: directory creation failures, VideoRAG import errors, file access/format validation, graceful cleanup on
interruption.
```

Listing 8: VideoSearcher Structure

```
Input: video_scene_path
Output: search_status

Algorithm:
1. Initialize dataset, scene\_output, and working directory paths; create missing directories; add tools to system path.
2. Dynamically import VideoRAG and QueryParam; handle import failures with error logging.
3. Load scene JSON (UTF-8), extract segment\_scene as query string, validate content availability.
4. Configure QueryParam (videoragcontent mode); toggle reference inclusion via wo\_reference flag; set multiprocessing spawn.
5. Execute semantic query via VideoRAG: embed scene descriptions, similarity-match against indexed videos, retrieve relevant
segments with timestamps.
6. Validate query completion, log search statistics, return structured search status with error reporting.

Error Handling: JSON parse/not-found errors, VideoRAG import/init failures, empty scene content, query exceptions with logging.
```

Listing 9: VideoEditor Structure

```
Input: video_directory, audio_path, timestamp_path
Output: final_video_path

Algorithm:
1. Load MiniCPM-V-2\_6-int4 model; parse visual\_retrieved\_segments.json and kv\_store\_video\_segments.json for segment and
timing metadata.
2. Parse rhythm\_points.json for beat timestamps; create time periods from consecutive beat intervals; load video\_scene.json
for storyboards and align with narrative segments.
3. For each video segment, extract frames at 1fps (224x224 RGB PIL Images), starting from segment start time plus subsequent
whole seconds.
4. Multimodal scene matching: construct VLM message with prompt text as first content element, then append extracted frames
sequentially. The prompt asks the model to find the optimal starting frame for the best consecutive sequence matching the scene
description, returning only the starting frame number.
5. Parse VLM response (regex-based number extraction); compute clip timing (start\_time + frame\_offset); validate boundaries;
extract clips matching beat intervals.
6. Audio integration: load and resize BGM; composite with original audio (keep\_original\_audio=True) or replace entirely (False
); adjust volume and sync timeline.
7. Concatenate clips via compose; export with fps=24, codec=libx264, audio\_codec=aac, preset=medium, threads=4; cleanup temp
resources.

Error Handling: frame extraction failures, VLM parsing errors (regex fallback), file access errors, audio sync fallbacks, memory
 cleanup.
```

Listing 10: FaceSwapping Agent Structure

```
Input: source_video_path, target_face_image
Output: swapped_video_path

Algorithm:
1. Initialization:
   a. Set up file paths and working directories
   b. Initialize access to Viggle AI face swapping tool

2. Source video processing:
   a. Load source video and extract frames
   b. Detect and track the specified face across frames
   c. Prepare frames containing the target face for swapping

3. Face swapping operation:
   For each frame with detected target face:
     a. Send frame and target_face_image to Viggle AI tool
     b. Receive the frame with the specified face replaced by the target face
     c. Replace the original frame with the swapped output

4. Video reconstruction:
   a. Reassemble processed frames preserving original video properties
   b. Synchronize original audio track if available
   c. Output the final face-swapped video file

5. Error handling and cleanup:
   a. Handle failures in calling Viggle AI or frame processing gracefully
   b. Log issues and fall back to original frames if needed
   c. Release resources and temporary files

Notes:
- Input video and target face image are required inputs
- Output is a video with the specified face replaced by the target
- Reliant on Viggle AI for core face swapping capabilities
- Designed for seamless integration within video pipelines
```

Listing 11: LipSynchronization Agent Structure

```
Input: source_video_path, target_audio_path
Output: synced_video_path

Algorithm:
1. Initialization:
   a. Configure input/output file paths and working directories
   b. Initialize connection and access to Kling AI lip synchronization tool

2. Data preparation:
   a. Load source video and extract frames and original audio if present
   b. Load target audio track for synchronization

3. Lip synchronization process:
   a. Send source video frames and target audio to Kling AI tool
   b. Receive processed video frames with speaker's lip movements synchronized to target audio
   c. Replace original frames with synchronized frames

4. Video reconstruction:
   a. Combine synchronized frames preserving original video resolution and frame rate
   b. Integrate the target audio track as the output video's audio
   c. Output final lip-synced video file

5. Error handling and cleanup:
   a. Handle communication errors with Kling AI tool gracefully
   b. Provide fallback to original video if synchronization fails
   c. Log all processing steps and encountered issues
   d. Release temporary resources and close file streams

Notes:
- Supports video and separate audio inputs
- Output video shows visually consistent lip motions aligned with target audio
- Relies fully on Kling AI external service for synchronization
- Designed for smooth integration with video editing workflows
```

Listing 12: AudioExtractor Structure

```
Input: video_path (file or directory)
Output: audio_paths, data_directory

Algorithm:
1. Validate video\_path (file or directory); scan for supported formats (.mp4, .avi, .mov, .mkv, .wmv, .flv, .webm, .m4v).
2. For each video, extract audio via FFmpeg: \texttt{ffmpeg -y -i input -vn -acodec pcm\_s16le -ar 44100 -ac 2 -loglevel error
output.wav}. Handle subprocess errors (CalledProcessError, FileNotFoundError for FFmpeg).
3. Batch processing: track success/failure per file, generate progress reports, collect successful paths, report statistics.
4. Return audio\_paths (list for batch, string for single) and data\_dir; handle empty results gracefully.

Error Handling: FFmpeg availability detection, per-file error isolation, directory permission validation, graceful degradation
for partial failures.
```

Listing 13: RhythmDetector Structure

```
Input: audio_file_path
Output: rhythm_analysis_directory

Algorithm:
1. Load audio via librosa.load() at original sample rate; configure STFT parameters (frame\_length=2048, hop\_length=512).
2. Compute RMS energy (librosa.feature.rms), normalize (rms/max), smooth via convolution kernel.
3. Detect peaks via scipy.signal.find\_peaks (height=0.4 threshold, configurable distance); convert indices to timestamps (
librosa.frames\_to\_time); apply temporal masking for intro/outro exclusion.
4. Generate rhythm points with sequential IDs and precise timing; filter masked regions; report statistics.
5. Create 3-panel visualization (waveform + markers, RMS curve + threshold, spectrogram) at 300 DPI with color-coded rhythm
points (red) and masked areas (gray).
6. Compute inter-beat interval statistics (mean, median, std, min, max); generate distribution histograms.
7. Export cut\_points.json (beat timing), rhythm\_detection.png, and rhythm\_distribution.png to audio\_analysis/.

Error Handling: format validation, path verification, graceful handling of insufficient rhythm points, file I/O exceptions.
```

Listing 14: LoudnessNormalizer Structure

```
Input: data_directory
Output: processing_status

Algorithm:
1. Validate and resolve input directory path via Path.resolve(); verify directory exists and is accessible.
2. Build FAP command: \texttt{fap loudness-norm <input\_dir> <output\_dir> --overwrite --recursive}.
3. Execute via subprocess.Popen with stdout/stderr PIPE (bufsize=1, line-buffered); spawn threads for real-time UTF-8 output
reading.
4. Monitor progress with [FAP]-prefixed console output; handle decoding errors with 'replace'.
5. Wait for completion (process.wait), join threads, evaluate return\_code: return success for 0, raise RuntimeError otherwise.

Error Handling: path validation, FAP availability detection, thread synchronization, subprocess error capture, resource cleanup.
```

Listing 15: Merge Structure

```
Input: video_path, audio_path
Output: merged_video_path

Algorithm:
1. Validate video\_path and audio\_path; set output to overwrite original video.
2. Execute FFmpeg merge: \texttt{ffmpeg -i video\_path -i audio\_path -c:v libx264 -preset fast -crf 23 -c:a aac -b:a 192k -map
0:v:0 -map 1:a:0 -shortest -movflags +faststart -y output\_path}.
3. Monitor return code; validate output file creation; ensure A/V sync.
4. Handle errors (CalledProcessError, permission, cleanup) with detailed reporting.
```

Listing 16: Mixer Structure

```
Input: bgm_path, audio_path
Output: mixed_audio_path

Algorithm:
1. Validate BGM and audio paths; determine output format from extension (default WAV); set output path as "mixed\_" + original
name.
2. Load BGM and vocal via AudioSegment.from\_file (supports WAV, MP3, FLAC etc.); apply BGM volume reduction (default -1 dB).
3. If BGM is shorter than vocal, loop-extend BGM to vocal length, then trim to exact match.
4. Apply overlay: vocal\_audio.overlay(bgm) for vocal-priority mixing; preserve audio quality.
5. Export via mixed\_audio.export(output\_path, format) (WAV/MP3/FLAC/AAC/OGG); return output path.

Error Handling: format compatibility, loading errors, export fallback to WAV.
```

Listing 17: Resampler Structure

```
Input: data_directory
Output: processing_status

Algorithm:
1. Validate and resolve input directory via Path.resolve(); verify exists and is a directory.
2. Build FAP command: \texttt{fap resample <input\_dir> <output\_dir> --overwrite} (in-place replacement).
3. Execute via subprocess.Popen (stdout/stderr PIPE, bufsize=1); spawn threads for real-time UTF-8 output reading with [FAP]
prefix.
4. Wait for completion, join threads; return success for return\_code=0, raise RuntimeError otherwise.

Error Handling: path validation, FAP availability, thread synchronization, return code analysis.
```

Listing 18: Separator Structure

```
Input: data_directory
Output: processing_status

Algorithm:
1. Validate and resolve input directory; verify exists and is a directory type.
2. Build FAP command: \texttt{fap separate <input\_dir> <output\_dir> --overwrite --recursive} (ML-based source separation).
3. Execute via subprocess.Popen with threaded stdout/stderr monitoring; real-time UTF-8 output with [FAP] prefix.
4. Wait for completion, join threads; evaluate return\_code; generate separated vocal and instrumental tracks.

Error Handling: directory validation, FAP availability, thread sync, resource cleanup for large files.
```

Listing 19: Transcriber Structure

```
Input: data_directory
Output: processing_status

Algorithm:
1. Validate and resolve input directory; verify exists and is a directory type.
2. Build FAP command: \texttt{fap transcribe --model-type funasr --recursive <audio\_dir>}.
3. Execute via subprocess.Popen with threaded stdout/stderr monitoring; real-time UTF-8 output with [FAP] prefix.
4. Wait for completion, join threads; evaluate return\_code; generate .txt files matching audio basenames.

Error Handling: directory validation, FunASR availability, format compatibility, thread cleanup.
```

Listing 20: VoiceGenerator Structure

```
Input: video_scene_path, target_vocal_path
Output: synthesized_audio_path, timestamp_path

Algorithm:
1. Load CosyVoice2-0.5B model (load\_jit/load\_trt=False, fp16=False); load target vocal prompt at 16kHz via load\_wav.
2. Parse scene JSON (UTF-8, ensure\_ascii=False); extract content\_created via '/////' delimiter; create segment list with
sequential IDs.
3. Split text into max 200-char chunks; handle Chinese and bilingual punctuation; maintain semantic coherence.
4. For each chunk, apply zero-shot synthesis: cosyvoice.inference\_zero\_shot(generator, prompt\_text, prompt\_speech\_16k,
stream=False); concatenate via torch.cat(); track cumulative timestamps.
5. Export final audio via torchaudio.save(); generate cut\_points.json with sentence\_data.chunks structure; cleanup temp files.

Error Handling: model loading verification, file validation, chunk-level error isolation, resource cleanup.
```

Listing 21: TTSInfer Structure

```
Input: audio_path, speech_text_path
Output: derivative_audio_directory

Algorithm:
1. Load rewritten text (UTF-8), split into paragraphs; scan slice directory for .lab/.wav pairs; create derivative output
directory.
2. For each paragraph, map .lab content to .wav reference audio; aggregate additional segments via modulo cycling if combined
duration < 1s; concatenate WAVs via scipy.io.wavfile.
3. Three-stage Fish-Speech pipeline: (1) VQGAN encode reference WAV -> .npy tokens (checkpoint: firefly-gan-vq-fsq); (2) Text-to-
Semantic conversion with --text and --prompt-text; (3) VQGAN decode semantic codes -> synthesized WAV.
4. Process each paragraph through all 3 stages; validate return codes per stage; collect WAVs.
5. Sort, load, and concatenate via numpy.concatenate(); export derivative/final.wav.

Error Handling: return code validation per stage, sample rate consistency, temp file cleanup.
```

Listing 22: TTSReplace Structure

```
Input: video_path
Output: final_video_path

Algorithm:
1. Validate video path; load metadata.json (clip file names, start/end times, durations); map derivative audio to video segments.

2. Per clip: extract video via FFmpeg (\texttt{-ss start -to duration -c:v libx264 -preset fast -vf scale=iw:ih}); get audio
duration via FFprobe; compute speed factor (target/clip duration).
3. Adjust video speed via setpts filter (\texttt{-filter:v setpts=speed\_factor*PTS -an}); merge with derivative audio (\texttt{-
c:v copy -c:a aac -map 0:v:0 -map 1:a:0 -shortest}).
4. Generate filelist.txt (UTF-8, absolute paths); concatenate all clips via FFmpeg concat demuxer (\texttt{-f concat -safe 0 -c
copy}); cleanup temp files, keep only final.mp4.

Error Handling: file validation, subprocess return codes, missing audio segments, resource cleanup.
```

Listing 23: TTSSlicer Structure

```
Input: audio_path
Output: processing_status, segmented_audio_files, metadata

Algorithm:
1. Configure parameters: min/max segment duration (6s/8s), silence threshold (top\_db=-35), hop\_length=10ms, max\_silence\_kept
=0.3s.
2. Load audio via librosa; compute RMS energy (frame-based, librosa.feature.rms) with dB-to-linear threshold conversion.
3. Detect silence regions (RMS < threshold); apply leading/middle silence segmentation rules; compute optimal cut points via
argmin on RMS; generate sil\_tags array.
4. Extract chunks via \_apply\_slice: convert frame indices to sample indices, slice waveform, compute precise timestamps (start
\_idx/sr, end\_idx/sr).
5. Post-process: merge short chunks or subdivide long ones via \_slice\_by\_max\_duration (ceil for uniform distribution).
6. Export via soundfile.write (sequential numbering: 0000.wav, ...); generate metadata.json (UTF-8) with start/end timestamps (3-
decimal precision).

Error Handling: format validation, empty audio detection, sample rate preservation, file I/O for large files.
```

Listing 24: SVCAnalyzer Structure

```
Input: midi_file_path, lyrics_file_path
Output: song_name, analysis_results_path

Algorithm:
1. Validate MIDI (.mid) and lyrics (.txt) files; load lyrics (UTF-8); extract song name.
2. Parse MIDI via mido.MidiFile; scan tracks for 'set\_tempo' messages; compute BPM (60000000/tempo); handle variable tempo with
 chronological sorting.
3. Extract notes: process note\_on (velocity>0) and note\_off; compute duration in ticks; detect rests (gaps > ticks\_per\_beat
/8); group simultaneous notes (<0.01 time units).
4. Convert ticks to seconds via tempo-aware formula: (ticks * tempo) / (ticks\_per\_beat * 1e6); format note names as "{note}{
octave}" (e.g., "C4", "F#3").
5. Align lyrics character-by-character to notes (excluding rests); insert "AP" markers for rest periods; generate processed
lyrics.
6. Export JSON to /analysis/{song\_name}.json with fields: text (with AP markers), notes (pipe-separated), notes\_duration,
input\_type="word".

Error Handling: MIDI/lyrics validation, encoding fallbacks, character count mismatch detection.
```

Listing 25: SVCConversion Structure

```
Input: adapted_lyrics_string, midi_analysis_path
Output: timestamp_json_path

Algorithm:
1. Load MIDI analysis JSON (UTF-8); extract note durations; update with adapted lyrics.
2. Parse adapted lyrics character-by-character against durations; detect "AP" markers as phrase boundaries; build timeline
entries: ("AP", start, end) or ("CHAR", char, start, end).
3. Group consecutive characters between AP markers into text segments; subdivide instrumental breaks >12s into "bgm" chunks at
12s intervals.
4. Combine text and AP chunks; sort by (timestamp, content\_type\_priority --- text before AP); export sentence\_data JSON (3-
decimal timestamps) to gen\_audio\_timestamps.json.

Error Handling: duration validation, timeline boundary checks, file I/O errors, encoding preservation.
```

Listing 26: SVCCoverist Structure

```
Input: source_audio_path, target_vocal_path
Output: synthesized_audio_path

Algorithm:
1. Validate source/target audio paths; set output as "{source\_name}\_{target\_name}.wav" in ../../final; create output
directory.
2. Navigate to tools/seed-vc; save and override PYTHONPATH for module resolution.
3. Run Seed-VC inference: \texttt{python inference.py --source <src> --target <tgt> --output <dir> --f0-condition True} (
preserves F0 pitch, transfers target timbre).
4. Monitor subprocess with UTF-8/fallback decoding; validate return\_code==0 and output file creation.
5. Restore original working directory and PYTHONPATH; return output path.

Error Handling: path validation, subprocess return codes, Unicode fallback, environment restoration.
```

Listing 27: SVCSingle Structure

```
Input: adapted_lyrics_string, midi_analysis_path, song_name
Output: synthesized_vocal_audio_path

Algorithm:
1. Load MIDI analysis JSON (UTF-8); extract notes, durations, and lyrics; split by "AP" markers for phrase boundaries.
2. Create segments with min 0.15s duration; aggregate short segments to 0.5s minimum via text concatenation (" | " note
separator); preserve AP markers.
3. Batch synthesize via DiffSinger: generate per-segment annotation JSON ({song\_name}\_part\_{start}-{end}.json); output WAV
files.
4. Apply phase vocoder time stretching (librosa, 44.1kHz): stretch factor = current/target duration; sample-level align (
truncate/pad).
5. Consolidate: convert to 16-bit PCM, normalize (np.clip * 32767), insert silence for AP markers, concatenate segments.
6. Export to dataset/mad\_svc/cover/; cleanup temp files.

Error Handling: JSON/audio validation, duration mismatch correction, sample rate consistency.
```

Listing 28: StandUpConversion Structure

```
Input: segment_directory, metadata_path
Output: timestamp_json_path

Algorithm:
1. Load metadata JSON (tone, text, reaction indicators); validate segment directory.
2. For each metadata entry, load {seg\_dir}/{index}.wav via soundfile; compute duration (samples/sample\_rate); track cumulative
 timing.
3. Generate chunk data: {id (1-based), timestamp (3-decimal), content}; preserve sequential ordering.
4. Verify timestamp progression and total duration; export sentence\_data JSON to timestamps.json (UTF-8, ensure\_ascii=False).

Error Handling: audio file validation, sample rate consistency, metadata correlation, JSON export errors.
```

Listing 29: CrossTalkConversion Structure

```
Input: segment_directory, metadata_path
Output: timestamp_json_path

Algorithm:
1. Load metadata JSON with role info (dou\_gen/peng\_gen), tone, text, and reaction indicators; validate segment directory.
2. For each entry, load {seg\_dir}/{index}.wav via soundfile; compute duration; track cumulative timing.
3. Format content as "[{role}] {dialogue\_text}"; generate chunk data: {id, timestamp (3-decimal), content}.
4. Verify timestamp progression and total duration; export sentence\_data JSON to timestamps.json (UTF-8).

Error Handling: audio file validation, sample rate consistency, metadata correlation, role validation.
```

Listing 30: RhythmContentGenerator Structure

```
Input: user_requirements, rhythm_analysis_directory, video_segments_path
Output: video_scene_path, video_summary_path

Algorithm:
1. Load video segments JSON; build caption bank: transform raw captions to numbered format ("Video Segments {id}:").
2. Feed caption bank as video context to the LLM (system: "creative beat sync video producer"); also parse rhythm\_points.json
for beat count and load rhythm visualization as pacing reference.
3. Global-aware storyboard generation: maintain multi-turn conversation state; combine rhythm reference + video captions + user
requirements into a structured prompt (/////-delimited output, max 2 sentences/scene, match rhythm patterns).
4. API calls via tenacity.retry (exponential backoff: min=2s, max=60s, 5 attempts); append conversation history for continuity.
5. Export video\_scene.json (UTF-8) with user\_idea, video\_summary, and segment\_scene fields.

Error Handling: retry on API failures, fallbacks for missing rhythm/video files, truncation for oversized content.
```

Listing 31: NewsContentGenerator Structure

```
Input: user_requirements, news_presentation_style, video_directory
Output: video_scene_json

Algorithm:
1. Scan for .lab transcript files (multi-encoding: utf-8, gb18030, gbk, etc.); parse [start end text] format; load presentation
style.
2. Presenter agent: adapt transcripts to narration following user ideas and style (third-person, min 11 words/sentence, remove
section numbers, expand abbreviations).
3. Judger agent: reformat into /////-delimited segments per sentence; remove commas.
4. Generate English visual-scene keywords per segment (max 1 sentence, preserve proper nouns); count sections; export video\
_scene.json (UTF-8).

Error Handling: encoding fallbacks, API retry (5 attempts, exponential backoff), 15K char input truncation, judger fallback.
```

Listing 32: Video Event Overview Presentation Style

```
Core: third-person narrative, factual accuracy (no fabricated specs), proper subject attribution, no greetings/farewells.
Opening (first 3 sentences): compelling hook establishing central tension/innovation; approaches: situational, hypothetical,
generational, or technical revelation.
Body: timeline-based chronological progression with clear thread; avoid generalizations and banned phrases ("The event shows...",
 "reveals...", "demonstrates...", word "event" in body); use accessible language; tight pacing and smooth transitions.
Closing: thematic elevation without summarization; refine deeper meaning; connect to broader implications.
QA: grammar, third-person consistency, factual verification, character attribution, banned-phrase compliance, timeline coherence.
```

Listing 33: CommentaryContentGenerator Structure

```
Input: user_requirements, source_text_path, commentary_presentation_style
Output: video_scene_json

Algorithm:
1. Load source text (multi-encoding: utf-8, gb18030, gbk, etc.; 30K char limit).
2. Presenter agent: adapt source to narration following user ideas and presentation style (source language, max 3 commas/
sentence, preserve original dialogues, clear narrative flow).
3. Judger agent: segment into /////-delimited sentences; remove chapter numbers and extra punctuation.
4. Visual scene generation: convert to English visual-scene descriptions (max 1 sentence/section, describe character appearances,
 keep ///// markers).
5. Export video\_scene.json (UTF-8) with reqs, content\_created, segment\_scene.

Error Handling: encoding fallbacks, API retry (5 attempts, exponential backoff), 30K truncation.
```

Listing 34: Content Writing Methodology

```
Opening: first 3 sentences with compelling hook. Approaches include identity immersion, knowledge-guided, hypothetical scenarios,
 role-playing, time-travel hypothetical, real-event revelation, ability paradox, expectation reversal, extreme situations, and
suspenseful questions.
Main Body: timeline-based plot progression, clear narrative thread, max 3 commas/sentence, smooth transitions, key character
dialogue.
Prohibited: literary analysis, thematic summaries, phrases like "The story shows...", "demonstrates...", word "story" in body.
Required Transitions: high-frequency use of "but", "yet", "however", "turns out", "suddenly", "unexpectedly", "finally", etc.
Narrative Techniques: short sentences for pacing, sensory/emotional adjectives, contrast/emphasis, conversational tone,
rhetorical devices (metaphor, hyperbole), strategic suspense, dramatic contrast.
Closing: extract deeper meaning, elevate themes; avoid "story ending"/"story beginning".
```

Listing 35: TTSWriter Structure

```
Input: user_requirements, audio_path
Output: rewritten_speech_path

Algorithm:
1. Scan slice directory for .lab files; aggregate transcripts + original transcript (UTF-8); build structured format per slice.
2. Claude LLM: creative recreation --- preserve linguistic style and sentence structure, replace specific content, length
variation <=2 chars.
3. Format template: "{index}. Original: {original}\n   Recreation:"; apply user requirements; enforce length constraints.
4. DeepSeek LLM: extract recreation content line by line; save raw\_speech.txt and final speech.txt (UTF-8).

Error Handling: lab file validation, UTF-8 encoding, LLM API failures, directory structure validation.
```

Listing 36: SVCAdapter Structure

```
Input: user_requirements, midi_analysis_path, song_name
Output: adapted_lyrics_string

Algorithm:
1. Load MIDI analysis JSON (UTF-8); extract original lyrics; parse structure by "AP" markers into LYRICS placeholders and AP
delimiters.
2. Generate numbered template per segment: "{index}. Original: {content}\n   Character limit: {count}\n   Recreation:".
3. Claude LLM: recreate lyrics following character limits, rhyme/rhythm, and narrative flow.
4. DeepSeek LLM: extract recreated lyrics line by line.
5. Character count alignment (max 5 retries): context-aware correction via Claude; fallback padding ("la") or truncation.
6. Reconstruct: merge aligned lyrics with AP markers; export raw\_lyrics.txt, lyrics.txt, script.txt, and {song\_name}\_cover.
json.

Error Handling: JSON validation, character count mismatch correction, LLM API failures.
```

Listing 37: StandUpAdapter Structure

```
Input: user_requirements, reference_script_path
Output: segmented_standup_script

Algorithm:
1. Load reference script (UTF-8); define tone markers ([Natural], [Confused], [Empathetic], [Exclamatory]) and atmosphere cues
([Laughter], [Cheers]).
2. Claude LLM: adapt into stand-up comedy --- each line begins with tone marker, atmosphere cues at key punchlines (3-4 total),
preserve humor while localizing references, target 3-5 min performance.
3. Structure: "# title\n[Tone]content...\n[Tone]content...[Atmosphere]\n..."; save to stand-up.txt.

Error Handling: file validation, Claude API failures, UTF-8 encoding.
```

Listing 38: StandUpSynth Structure

```
Input: segmented_script, target_vocal_directory, reaction_directory
Output: merged_audio_path, segment_directory, metadata_path

Algorithm:
1. Load CosyVoice2-0.5B (load\_jit/load\_trt=False, fp16=False); set up vocal and reaction directories.
2. Split script by lines, skip title; DeepSeek LLM parses each line -> JSON {tone, text, reaction}; validate and strip markdown
from response.
3. Per segment: CosyVoice2 zero-shot synthesis using tone-specific prompts ({tone}.lab + {tone}.wav); save {index}.wav.
4. If reaction present, load {reaction}.wav; concatenate speech + reaction via PyDub AudioSegment.
5. Merge all segments (0.wav..cnt.wav) to ../final/stand\_up.wav; export metadata as stand-up.json.

Error Handling: model loading, JSON parsing, missing reaction files, audio validation, environment restoration.
```

Listing 39: CrossTalkAdapter Structure

```
Input: user_requirements, reference_script_path, dou_gen_directory, peng_gen_directory
Output: segmented_crosstalk_script

Algorithm:
1. Extract performer names (dou\_gen/peng\_gen) from directory basenames; load reference script (UTF-8).
2. Claude LLM: adapt into crosstalk dialogue --- role-alternating lines with [Natural]/[Confused]/[Emphatic] tone markers (no
consecutive same tone >2 lines), localize humor for Chinese audience, balanced dialogue allocation.
3. Format: "Title\n[tone] Role: content...\n..."; save to cross-talk.txt.

Error Handling: file validation, Claude API failures, UTF-8 encoding.
```

Listing 40: CrossTalkSynth Structure

```
Input: segmented_script, funny_guy_directory, setup_guy_directory
Output: merged_audio_path, segment_directory, metadata_path

Algorithm:
1. Extract role names (dou\_gen/peng\_gen) from directories; load CosyVoice2-0.5B.
2. Split script by lines; DeepSeek LLM parses each line -> JSON {role, tone, text, reaction}; validate performer names.
3. Per segment: CosyVoice2 zero-shot synthesis using role/tone-specific prompts ({role}/{tone}.lab + .wav at 16kHz); save {index
}.wav.
4. Merge all segments via PyDub to ../final/cross\_talk.wav; export metadata as cross-talk.json.

Error Handling: model loading, JSON parsing, missing prompt files, role validation, environment restoration.
```

Listing 41: VideoConversion Structure

```
Input: timestamp_path
Output: video_scene_path

Algorithm:
1. Load timestamp JSON (cut\_points.json); extract content from sentence\_data.chunks; join with ///// separators.
2. LLM (DeepSeek): generate English visual-scene descriptions --- keep ///// markers, max 1 sentence/segment, describe character
 appearances, don't directly translate.
3. Export video\_scene.json (UTF-8) with segment\_scene and content\_created; validate segment count matches input.

Error Handling: file/JSON validation, empty segments, LLM parsing fallbacks.
```

Listing 42: VideoQA Structure with System Prompt

```
Input: video_directory, output_path, save_history_flag
Output: transcript_path, qa_session_results

Algorithm:
1. Load Whisper model; scan directory for videos; transcribe each (30s chunks, batch-optimized) with markers ("=== VIDEO:
filename ===").
2. Consolidate transcripts; truncate at 50K chars; initialize interactive Q&A session.
3. Q&A loop: answer user questions strictly from transcripts with source video citation; state if answer not found; log
conversations with timestamps.
4. Save Q&A history and return results.

Error Handling: transcription fallback, graceful session termination, comprehensive logging.
```

Listing 43: VideoSummarizationGenerator Structure

```
Input: user_idea, video_directory, presentation_style_path, output_path
Output: content_output, status, processed_videos, transcript_source

Algorithm:
1. Load whisper-large-v3-turbo ASR pipeline (max\_new\_tokens=128, chunk\_length\_s=30, batch\_size=16); auto-detect CUDA/CPU.
2. Scan for video files; transcribe each with retry (3 attempts); combine with "=== VIDEO: filename ===" markers; also support
text file input (multi-encoding).
3. Presenter agent: generate summarization following user ideas and presentation style (third-person, fewer bullet points, word
count adherence); truncate input at 15K chars.
4. Save output as plain text; return metadata (processed\_videos, transcript\_source, status).

Error Handling: retry logic (3-5 attempts, exponential backoff), CUDA fallback, encoding fallbacks.
```

Listing 44: System Prompt of Intent Analysis

```
You are an intent analyst.
I will provide a set of candidate intents and a user's requirements.
Please select as many relevant candidate intents as possible that match the user's requirements. Consider factors such as
keywords, audio types (speech, song, music, etc.), and other relevant dimensions.

Candidate intents:
{intents}

User requirements:
{reqs}

Please Only output pure List format:
['intent1', 'intent2', ...]

Note! Don't output any analysis and explanations!
```

Listing 45: System Prompt of Intent Analysis

```
You are an intent analyst.
Previous analysis attempt failed with the following reflection:
{reflection}

Previous selected intents:
{previous_intents}

Please re-analyze the user's requirements with the candidate intents below, considering the reflection.
Select as many relevant candidate intents as possible.

Candidate intents:
{intents}

User requirements:
{reqs}

Please Only output pure List format:
['intent1', 'intent2', ...]

Note! Don't output any analysis and explanations!
```

Listing 46: System Prompt of Agent Graph

```
You are an Agent Graph Designer. I will provide User Requirement and Registered Agents (Name, Description and Parameters
information):

Your task is to:
1. Judge Feasibility:
   - Evaluate implementation feasibility of User Requirements
   - Output: "Feasible" or "Infeasible" (strictly one of these)
2. Design Executable Agent Graph:
   - Format: List
   - Agent Graph shall contain metadata for each Agent Node including:
     * name: (string)
     * inputs: (list of input parameter objects with):
     * parameter: input parameter name
     * description: brief parameter description
     * outputs: (list of output parameter objects with):
     * parameter: output parameter name
     * description: brief parameter description
     * links: (list of dictionaries) where each dictionary specifies:
         - key of dictionaries: target agent name
         - value of dictionaries: target agent's input parameter name that this output connects to
3. Generate Agent Chain:
   - Format: List
   - Generate the Agent Chain based on the description of the Agent and the sequential information contained in the designed
Agent Graph
4. Generate User Input Graph
   - Format: List
   - Parameter nodes with no in-degree (no incoming edges) are uniformly considered to require user input.
   - Parameter nodes with no in-degree may have different names but share the same user input, meaning a single user input
parameter can point to multiple such nodes.
   - Parameter nodes with no in-degree that are linked to user input should be represented in the format **AgentName.
input_parameter**
   - Generate the User Input Graph based on the Agent descriptions and parameter passing information in the designed Agent Graph.

5. Output Reasoning:
   - If Feasible, Provide concise reasoning (<200 words) explaining the entire workflow logic
   - If Infeasible, Specify exact failure reasons (<200 words)

In addition to the above formatting requirements, please also note the following:
1. For each element of **outputs** in each Agent Node:
   - Ensure that the **links** in the **outputs** point to an input parameter that actually exists in the next Agent Node.
   - The output parameter name does not need to match the input parameter name in the next Agent Node.
   - Ensure the output parameter's description and type match the input parameter requirements of the next Agent Node. For
example, a file path output cannot be passed to a directory path input.
2. Final JSON Output Format Specification:
{
    "Feasibility": "Feasible" or "Infeasible",
    "Agent Graph": ...,
    "Agent Chain": ...,
    "User Input Graph": ...,
    "Reasoning": ...
}
Strictly follow JSON output format!
```

Listing 47: System Prompt for Initial Evaluation

```
You are an agent graph validation system.
I will provide:
1. User Requirement
2. Registered agent metadata
3. Candidate Agent Graph
4. An Agent Chain derived from the Candidate Agent Graph
5. Required User Inputs

User Requirements:
{reqs}

Metadata of registered agents:
{tools}

Task: Evaluate the candidate agent graph:

Candidate Agent Graph:
{agent_graph}

Agent Chain:
{agent_chain}

Required User Inputs:
{user_inputs}

Evaluation Criteria:
1. Based on the Metadata of registered agents and parameter passing in the Agent Graph, determine from multiple aspects whether
the user requirements can be fulfilled:
   - Execution sequence of agents in the Agent Graph
   - For parameter nodes with no incoming edges, they are uniformly considered as user inputs, but it is necessary to determine
whether they should be provided by the user or by the parent agent
   - Validate that the necessary output parameters are correctly routed to the intended agent and the expected input parameters.
2. There should be no functionally redundant agents (e.g., repeatedly adding audio tracks to a video).
3. For vaguely mentioned requirements in user needs, lenient evaluation is acceptable. For example, if the user requests audio
quality improvement, it's sufficient as long as at least one relevant agent in the graph meets this requirement.

Please Only output pure JSON format:
{{
"Result": '0' if correct else '1',
"Reasoning": Concisely state the key reasons why a score of '0' or '1' was assigned (<100 words).
}}
```

Listing 48: System Prompt for Secondary Evaluation

```
You are an agent graph reflection system.
I will provide:
1. User Requirement
2. Registered agent metadata
3. Candidate Agent Graph
4. An Agent Chain and User Input Graph derived from the Candidate Agent Graph
5. Previous validation result

User Requirements:
{reqs}

Metadata of registered agents:
{tools}

Task: Evaluate the candidate agent graph:

Candidate Agent Graph:
{agent_graph}

Agent Chain:
{agent_chain}

Required User Input Graph:
{user_inputs}

Previous validation result:
{judge_res}

Reflection Task:
1. If the previous validation result is '0', please reflect on whether there were any overlooked aspects based on the **
Evaluation Criteria** and the reasoning behind the previous validation result.
2. If the previous validation result is '1', please reflect on whether the reasoning behind the previous validation result was
correct.

Evaluation Criteria:
1. Based on the Metadata of registered agents and parameter passing in the Agent Graph, determine from multiple aspects whether
the user requirements can be fulfilled:
   - Execution sequence of agents in the Agent Graph
   - For parameter nodes with no incoming edges, they are uniformly considered as user inputs, but it is necessary to determine
whether they should be provided by the user or by the previous agent
   - Validate that the necessary output parameters are correctly routed to the intended agent and the expected input parameters.
   - Validate that the output parameters' description and type match the input requirements of the next agent.
   - Not all output parameters are necessarily mapped to the input requirements of the next agent. Redundant output parameters
may exist, but they should not interfere with the fulfillment of user requirements.
2. There should be no functionally redundant agents (e.g., repeatedly adding audio tracks to a video).
3. For vaguely mentioned requirements in user needs, lenient evaluation is acceptable. For example, if the user requests audio
quality improvement, it's sufficient as long as at least one relevant agent in the graph meets this requirement.

Please Only output pure JSON format:
{{
"Result": '0' if correct else '1',
"Reasoning": Concisely state the key reasons why a score of '0' or '1' was assigned (<100 words).
}}
```

Listing 49: User Requirement of Audio Preprocessing

```
User Requirement: I need to extract the audio from a video file, improve its sound quality by normalizing the loudness, and then
 mix it with some background music. The result should be a clean audio track that balances the original audio with background
music.
```

Listing 50: User Requirement of Audio Preprocessing


```
{
"Feasibility": "Feasible",
"Intent List": ["Loudness Normalization", "Add BGM", "Audio Optimization"],
"Agent Graph": [
    {"node": "LoudnessNormalizer", "inputs": [{"name": "data_dir", "description": "Directory of audio files to be normalized"}],
 "outputs": [{"name": "status", "description": "Execution status (success/error)", "links": []}]},
    {"node": "Mixer", "inputs": [{"name": "bgm_path", "description": "Path to the BGM file"}, {"name": "audio_path", "
description": "Audio to be mixed into the background music"}], "outputs": [{"name": "audio_path", "description": "Path to the
synthesized audio", "links": []}]}
],
"Agent Chain": ["LoudnessNormalizer", "Mixer"],
"User Input Graph": [
    {"node": "raw_audio_directory", "description": "Directory containing the user's raw audio recording that needs to be
normalized", "links": [{"LoudnessNormalizer": "data_dir"}]},
    {"node": "normalized_audio_path", "description": "Path to the normalized audio file after processing by LoudnessNormalizer",
 "links": [{"Mixer": "audio_path"}]},
    {"node": "background_music_path", "description": "File path to the background music to be mixed with the voice recording", "
links": [{"Mixer": "bgm_path"}]}
],
"Reasoning": "This workflow normalizes the user's raw audio recording and then mixes it with background music. First,
LoudnessNormalizer processes the raw audio files to ensure consistent volume levels. Since LoudnessNormalizer doesn't output
file paths (only a status), we require the user to provide the path to the normalized audio file after processing. Then, the
Mixer agent combines this normalized audio with background music, ensuring the music doesn't overpower the voice. The workflow
addresses the requirement to add background music to a recording while maintaining proper audio balance."
}
```


Listing 51: User Requirement of Storytelling Video

```
User Requirement: I want to create a commentary video about a novel. The video should have professional voiceover with
background music and appropriate visuals that match the commentary content.
```

Listing 52: Case of Storytelling Video


```
{
"Feasibility": "Feasible",
"Intent List": ["Text-to-Speech", "Commentary", "Add BGM", "Video Edit"],
"Agent Graph": [
    {"node": "CommentaryContentGenerator", "inputs": [{"name": "reqs", "description": "User's idea for the commentary video
including word count requirements"}, {"name": "source_text", "description": "File path to the novel source text"}, {"name": "
comm_present_style", "description": "File path to commentary presentation style for content generation"}], "outputs": [{"name":
"video_scene_path", "description": "File path storing scene semantics for video storyboard sound synthesis.", "links": [{"
VoiceGenerator": "video_scene_path"}, {"VideoSearcher": "video_scene_path"}]}]},
    {"node": "VoiceGenerator", "inputs": [{"name": "video_scene_path", "description": "Path to a custom scene JSON file"}, {"
name": "target_vocal_path", "description": "Path to the target timbre for voice generation"}], "outputs": [{"name": "audio_path",
 "description": "Path to the synthesized audio", "links": [{"Mixer": "audio_path"}]}, {"name": "timestamp_path", "description":
"Path to video frame timestamp", "links": [{"VideoEditor": "timestamp_path"}]}]},
    {"node": "Mixer", "inputs": [{"name": "bgm_path", "description": "Path to the BGM file"}, {"name": "audio_path", "
description": "Audio to be mixed into the background music"}], "outputs": [{"name": "audio_path", "description": "Path to the
synthesized audio", "links": [{"VideoEditor": "audio_path"}]}]},
    {"node": "VideoPreloader", "inputs": [{"name": "video_dir", "description": "Directory containing the source MP4 video files
to be processed"}], "outputs": [{"name": "status", "description": "Execution status (success/error)", "links": []}]},
    {"node": "VideoSearcher", "inputs": [{"name": "video_scene_path", "description": "File path storing scene semantics for
video storyboard sound synthesis."}], "outputs": [{"name": "status", "description": "Execution status (success/error)", "links":
 []}]},
    {"node": "VideoEditor", "inputs": [{"name": "video_dir", "description": "Directory containing source video files"}, {"name":
 "audio_path", "description": "Path to the audio"}, {"name": "timestamp_path", "description": "JSON File path used to store and
load the timestamp of the end of each video segment"}], "outputs": [{"name": "video_path", "description": "Path to the generated
 video file", "links": []}]}
],
"Agent Chain": ["CommentaryContentGenerator", "VoiceGenerator", "Mixer", "VideoPreloader", "VideoSearcher", "VideoEditor"],
"User Input Graph": [
    {"node": "commentary_requirements", "description": "User's specific requirements for the commentary video including word
count and style", "links": [{"CommentaryContentGenerator": "reqs"}]},
    {"node": "novel_file", "description": "File path to the novel text that will be the subject of commentary", "links": [{"
CommentaryContentGenerator": "source_text"}]},
    {"node": "commentary_style", "description": "File path to the desired presentation style for the commentary", "links": [{"
CommentaryContentGenerator": "comm_present_style"}]},
    {"node": "voice_timbre", "description": "Path to the target vocal timbre file for professional voiceover", "links": [{"
VoiceGenerator": "target_vocal_path"}]},
    {"node": "background_music", "description": "Path to the background music file", "links": [{"Mixer": "bgm_path"}]},
    {"node": "visuals_directory", "description": "Directory containing source video files for visual content", "links": [{"
VideoPreloader": "video_dir"}, {"VideoEditor": "video_dir"}]}
],
"Reasoning": "The workflow begins with generating commentary content about the novel based on user requirements. This content is
 structured with scene semantics. The VoiceGenerator then creates professional voiceover audio and provides timestamps for
visual syncing. In parallel, VideoPreloader prepares video files from the user's library while VideoSearcher identifies
appropriate visual clips based on the commentary content. The voiceover is mixed with background music by the Mixer agent.
Finally, VideoEditor combines the mixed audio track with matching visuals using the timestamps to synchronize content, producing
 a complete commentary video about the novel with professional narration, appropriate background music, and relevant visuals."
}
```

Listing 53: MLLM Video Clip Matching Prompt with Frame Integration

```
Prompt Construction with Frame Integration:

1. Text prompt construction:
prompt = f"""
Here is a video broken down into {len(chunks)} clips of 3 seconds each.
Each clip shows multiple frames from that time segment.
The clips are structured as follows:
"""

for chunk in chunks:
    prompt += f"Clip {chunk['chunk_idx']}\n"

prompt += f"""
Identify {num_periods} specific clips that best correspond to the following captions:
"""

for i, caption in enumerate(shuffled_captions):
    prompt += f"{i+1}. {caption}\n"

prompt += f"""
Response with a JSON structure containing the clip numbers that best match each caption.

Rules:
1. Each clip_id should be a number (not a string)
2. Include exactly {num_periods} selections
3. Choose from the clip numbers shown (0 to {len(chunks)-1})
4. Keep the reason concise (1-2 sentences)
5. Ensure your output is valid JSON - no trailing commas, proper quotes, etc.
6. Do not include the ```json prefix or ``` suffix around your response
7. Do not answer anything unrelated

Return only the JSON object with no additional text.

Output Example:
{{
"selections": [
    {{
    "caption": "[caption text]",
    "clip_id": [clip number],
    "reason": "[brief description of what is seen in this clip]"
    }},
    ...
]
}}
"""

2. Frame integration into multimodal message structure:
messages = [{"role": "user", "content": []}]

# Add text prompt as first content element
messages[0]["content"].append({"type": "text", "text": prompt})

# Sequential frame insertion for visual analysis
for img_base64 in frames:
    messages[0]["content"].append({
        "type": "image_url",
        "image_url": {
            "url": f"{img_base64}"
        }
    })
```