國立臺灣大學電機資訊學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master's Thesis

透過電影中的隱喻探討大型語言模型的影片推理能力

Investigating Video Reasoning Capability of Large Language Models with Tropes in Movies

趙雋同

Chun-Tong Chao

指導教授: 徐宏民 博士

Advisor: Winston H. Hsu, Ph.D.

中華民國 113 年 6 月

June, 2024

doi:10.6342/NTU202401530

國立臺灣大學碩士學位論文
口試委員會審定書
MASTER'S THESIS ACCEPTANCE CERTIFICATE
NATIONAL TAIWAN UNIVERSITY

透過電影中的隱喻探討大型語言模型的影片推理能力

Investigating Video Reasoning Capability of Large Language Models with Tropes in Movies

本論文係趙雋同君(學號R11922109)在國立臺灣大學資訊工程學系完成之碩士學位論文，於民國113年6月28日承下列考試委員審查通過及口試及格，特此證明。

The undersigned, appointed by the Department of Computer Science and Information Engineering on 28 June 2024 have examined a Master's thesis entitled above presented by CHUN-TONG CHAO (student ID: R11922109) candidate and hereby certify that it is worthy of acceptance.

口試委員Oral examination committee:

(指導教授Advisor)	陳文進	莫尚元
		葉梅珍

系主任/所長Director: 陳祝嵩

doi:10.6342/NTU202401530

Acknowledgements

在兩年碩士生涯裡，由衷感謝徐宏民教授的指導以及建議，無論是從找題目的掙扎又或者實驗時遇到的瓶頸，教授都能以犀利的角度拆解問題並點出盲點。同時也感謝蘇弘庭學長在整篇論文中的幫助，從題目發想、實驗設計甚至中間的挫敗，都是學長從旁提攜和提供寶貴意見。從學長身上我學會了如何創意思考、嚴謹實驗和解決問題。

感謝伴侶、家人和朋友，讓我在遇到瓶頸焦慮時，能夠為我打氣鼓勵、給予我身心靈上的支持，

doi:10.6342/NTU202401530

doi:10.6342/NTU202401530

摘要

大型語言模型不僅在語言任務中表現出色，還在影片推理方面展示了其有效性。本文介紹了一個名為 **Tropes in Movies (TiM)** 的新資料集，旨在作為探索兩個關鍵但此前被忽視的影片推理技能：（1）抽象感知（Abstract Perception）：理解和標記影片中的抽象概念，（2）長時長組合推理（Long-range Compositional Reasoning）：規劃和整合中間推理步驟，以理解包含大量影格的長影片。通過利用電影敘事中的情節，TiM 評估了最先進且基於大型語言模型的方法的推理能力。我們的實驗表明，目前的方法，包括圖片描述-推理器、大型多模態模型指令微調和視覺編程，在應對抽象感知和長時長組合推理的挑戰時，僅比隨機方法稍有優越。為了解決這些不足，我們提出了角色互動增強的 Face-Enhanced Viper of Role Interactions（FEVoRI）和 Context Query Reduction（ConQueR），通過在推理過程中加強關注角色互動和逐步改進影片上下文和情節查詢，顯著提高了性能，提升了 15 個 F1 點。然而，這一性能仍落後於人類水平（40 vs. 65 F1）。此外，我們引入了一種新評估標準，用於評估抽象感知和長時長組合推理在任務解決中的必要性。這是通過使用抽象語法樹分析視覺編程生成的程式碼來完成的，從而確認了 TiM 相較其他資料集複雜。

關鍵字：大型語言模型、大型多模態模型、推理、視覺編程、隱喻

doi:10.6342/NTU202401530

doi:10.6342/NTU202401530

Abstract

Large Language Models (LLMs) have demonstrated effectiveness not only in language tasks but also in video reasoning. This paper introduces a novel dataset, **Tropes in Movies (TiM)**, designed as a testbed for exploring two critical yet previously overlooked video reasoning skills: (1) **Abstract Perception**: understanding and tokenizing abstract concepts in videos, and (2) **Long-range Compositional Reasoning**: planning and integrating intermediate reasoning steps for understanding long-range videos with numerous frames. Utilizing tropes from movie storytelling, TiM evaluates the reasoning capabilities of state-of-the-art LLM-based approaches. Our experiments show that current methods, including Captioner-Reasoner, Large Multimodal Model Instruction Fine-tuning, and Visual Programming, only marginally outperform a random baseline when tackling the challenges of Abstract Perception and Long-range Compositional Reasoning. To address these deficiencies, we propose Face-Enhanced Viper of Role Interactions (FEVoRI) and Context Query Reduction (ConQueR), which enhance Visual Programming by fostering

doi:10.6342/NTU202401530

role interaction awareness and progressively refining movie contexts and trope queries during reasoning processes, significantly improving performance by 15 F1 points. However, this performance still lags behind human levels (40 vs. 65 F1). Additionally, we introduce a new protocol to evaluate the necessity of Abstract Perception and Long-range Compositional Reasoning for task resolution. This is done by analyzing the code generated through Visual Programming using an Abstract Syntax Tree (AST), thereby confirming the increased complexity of TiM

Keywords: LLMs, LMMs, reasoning, visual programming, tropes

doi:10.6342/NTU202401530

Contents

doi:10.6342/NTU202401530

doi:10.6342/NTU202401530

List of Figures

doi:10.6342/NTU202401530

doi:10.6342/NTU202401530

List of Tables

doi:10.6342/NTU202401530

doi:10.6342/NTU202401530

Chapter 1 Introduction

Large Language Models (LLMs)[2, 15, 21, 27] have not only dominated Natural Language Processing but also extended their reach into Computer Vision (CV) reasoning tasks. Leveraging LLMs as their foundation, various video reasoning models have been introduced. *Captioner-Reasoner (C-R)* [5, 17, 18, 28, 36] leverages visual language models (VLMs) to tokenize visual inputs into language tokens to feed into LLMs. While there may be potential information loss during captioning, C-R achieves remarkable performance on various video reasoning tasks such as NexT-QA [29]. *Large Multimodal Model Instruction Fine-tuning (LMM-IF)* [16, 33, 38] aligns visual inputs to LLMs' token space using projection layers, thereby avoiding information loss during captioning. *Visual Programming (VP)* [8, 25] harnesses LLMs to generate programs that call visual perception modules and integrate their outputs. In contrast to the C-R and LMM-IF approaches, VP facilitates "System 2 style" stepwise reasoning [7]. It demonstrates the capability to address complex reasoning tasks that require external knowledge or commonsense such as [10, 20, 29] in a stepwise and interpretable manner. While LLM-based methods demonstrate significant performance on existing benchmarks , several critical aspects remain underexplored in current models and datasets, as shown in Figure 1.1. First, **Abstract Perception**: While most queries in existing datasets target concrete elements like actions, objects, or attributes —easily captured by vision models—abstract concepts such as emotion, motivation, hu-

doi:10.6342/NTU202401530

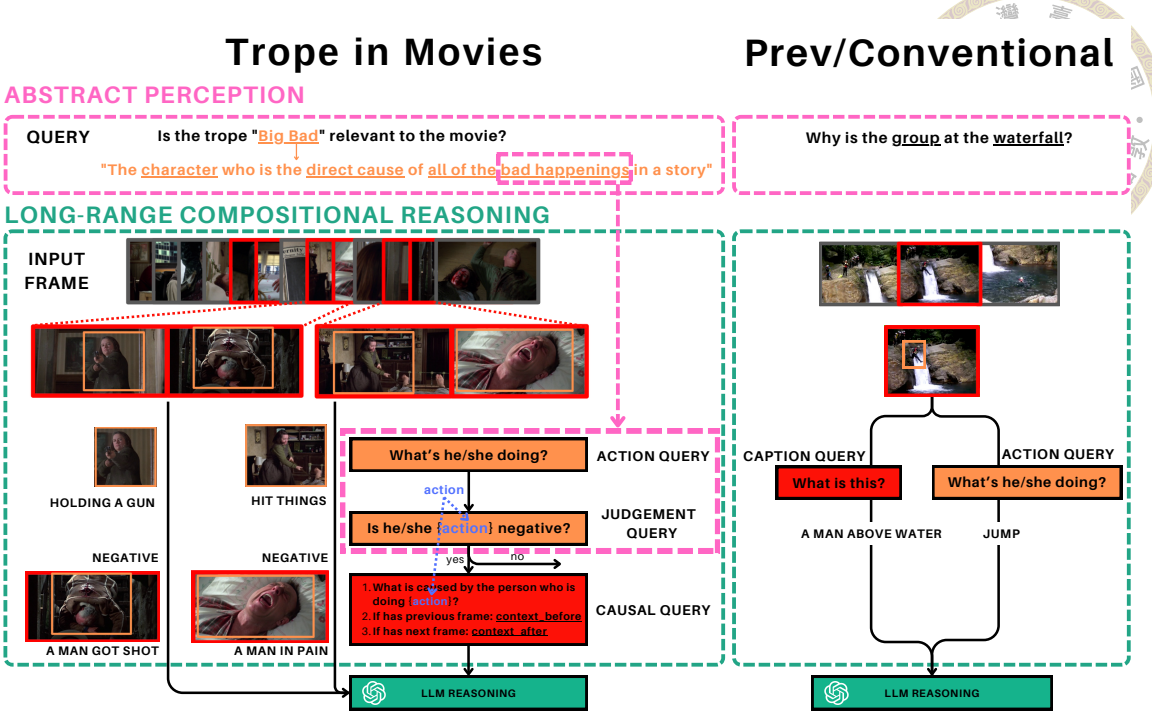


Figure 1.1: Compared to previous datasets like NExT-QA [29], Tropes in Movies (TiM) introduces the challenges of **Abstract Perception** (upper box) and **Long-range Compositional Reasoning** (lower box), offering a robust framework for evaluating and developing LLM-based methods. The blue text (action) indicates that the answer to the action query will affect the input of the judgment query and causal query, which means decomposing these complex elements necessitates multiple, nested queries that are interdependent.

mor, and judgment remain obscure and continue to challenge advanced VLMs. Second, **Long-range Compositional Reasoning**: Traditional datasets often assume that context and queries are straightforward, suitable for sparse sampling and simple decomposition. However, the reality is that contexts can span hour-long videos with thousands of frames, and queries may involve a wide range of complex elements. Decomposing these complex elements necessitates multiple, nested queries that are interdependent. Consequently, these prevailing approaches may overlook the nuanced interplay between visual cues and complex linguistic structures in longer, more dynamic sequences.

To evaluate these capabilities, we introduce a novel dataset, **Tropes in Movies (TiM)**, designed to rigorously test existing and future LLM-based video reasoning models against the challenges identified as Abstract Perception and Long-range Compositional Reason-

doi:10.6342/NTU202401530

ing. On TiM, the model has to determine whether a *trope* is present. *Tropes* are commonly employed narrative devices that enable storytellers to craft situations easily recognizable to audiences [3]. For instance, the trope "Big Bad" refers to an antagonist who is responsible for all the negative events in a story and drives the plot forward. Recognizing such a trope within varied narrative contexts demands Abstract Perception and Long-range Compositional Reasoning from a machine learning model. Abstract Perception allows the model to identify the essential characteristics of the "Big Bad" trope beyond specific instances, encompassing a range of characters, judgments, motivations, and actions that fit the trope's broad definition. Long-range Compositional Reasoning enables the model to process thousands of frames and decompose the concept of "Big Bad" into aspects such as evil characteristics, negative judgments, and the causation of terrible events. It also helps locate the relevant frames from among thousands to determine whether the trope is present.

We conducted comprehensive experiments on TiM using state-of-the-art (SOTA) LLM-based methods. These SOTA methods achieved a maximum F1 score of 25, only marginally surpassing the random baseline and significantly lagging behind human performance (65 F1 [3]). Even Gemini-1.5 [27], which is known for multimodal long-context abilities, only reaches 40 F1. This underscores that advanced LLM-based video reasoning methods, including C-R [36], LMM-IF [16, 33], and VP [25], struggle with the Abstract Perception and Long-range Compositional Reasoning challenges presented by TiM. Consequently, TiM could serve as an effective testbed for further developing and evaluating future LLMs. Additionally, we have enhanced ViperGPT [25] by introducing a Face-Enhanced Viper of Role Interactions (FEVoRI) that fosters role awareness and a Context Query Reduction (ConQueR) that decouples context from query during reasoning, which

doi:10.6342/NTU202401530

improved the F1 score of base ViperGPT by 15 points. However, the performance still lags significantly behind human benchmarks (40 vs. 65 F1), indicating substantial room for improvement.

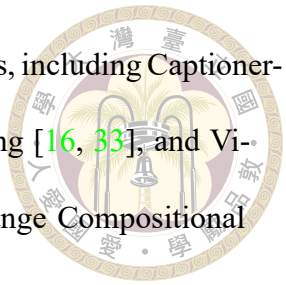
We conducted a comprehensive ablation study on FEVoRI to explore the impact of Abstract Perception and Long-range Compositional Reasoning. Our findings reveal that TiM: (1) requires a higher number of frames to achieve optimal performance, with a noticeable decrease (-2.8 F1) when sparse sampling methods—commonly employed in many models—are used; (2) sees a significant improvement (+4.5 F1) with the adoption of advanced VLM (replace BLIP-2 [13] with Gemini [27]) that bolster abstraction; and (3) shows that GPT-4 [21] performs only marginally better (by 0.17 F1) than GPT-3.5.

To more accurately quantify the challenges of Abstract Perception and Long-range Compositional Reasoning in datasets, we examine the abstract syntax tree (AST) of code generated by (VP). We propose a novel framework, AST Based Code Dignosis (ABCD), which is AST-based, to evaluate the levels of Abstract Perception and Long-range Compositional Reasoning. ABCD quantifies Abstract Perception by counting VLM calls and token lengths, and examines Long-range Compositional Reasoning through the nodes and edges of the AST. ABCD reveals that TiM necessitates code with higher Abstract Perception and Long-range Compositional Reasoning. It also provides a useful tool for quantifying challenges in video reasoning for future tasks.

The contributions of this work are summarized as follows:

- We introduce a novel dataset, **Tropes in Movies (TiM)**, designed to assess the Abstract Perception and Long-range Compositional Reasoning aspects of video reasoning.

doi:10.6342/NTU202401530



- We demonstrate that SOTA LLM-based video reasoning methods, including Captioner-Reasoner [36], Large Multimodal Model Instruction Fine-tuning [16, 33], and Visual Programming [25], face Abstract Perception and Long-range Compositional Reasoning challenges in effectively tackling TiM.

- We enhanced Viper [25] by introducing FEVoRI and ConQueR. These enhancements respectively enable role awareness and the decoupling of context from the query, facilitating progressive reasoning. This approach improved the F1 score by 15 points, marking a significant step toward reaching human-level performance (40 vs. 65 F1).

- We have established a protocol, AST Based Code Dignosis (ABCD), which utilizes the abstract syntax tree (AST) of generated code to evaluate the levels of Abstract Perception and Long-range Compositional Reasoning in datasets. ABCD not only highlights the unique challenges presented by TiM compared to previous models but also provides a valuable tool for future research to analyze datasets.

doi:10.6342/NTU202401530

doi:10.6342/NTU202401530

Chapter 2 Related Work

2.1 Comparison to Existing Tasks

TiM presents a unique challenge in video reasoning, requiring Abstract Perception and Long-range Compositional Reasoning. Most existing benchmarks primarily focus on identifying specific objects, actions, or attributes in short video clips [32, 34, 35]. TVQA [11, 12], which leverages TV series similar to the movies used in our benchmark, creates a dataset centered on temporal relations. More recent datasets have advanced further to include causal relations [1, 14, 29] and incorporate external knowledge [20]. While these tasks pose challenges for conventional end-to-end video QA models, LLM-based models significantly enhance performance in a training-free manner by tokenizing inputs and incorporating commonsense knowledge from LLMs. For instance, training-free LLM-based methods [25, 33, 36] outperform previous supervised models [30, 31] that were specifically trained for Video QA tasks. While several datasets [19, 39] attempt to assess the model's capability to handle long-range videos, they do not incorporate the same levels of Abstract Perception and Long-range Compositional Reasoning. TrUMAn [24] is another dataset that uses tropes in video clips to evaluate machine learning models; however, it utilizes short clips featuring a single trope and does not involve the same depth of Long-range Compositional Reasoning. Therefore, we are optimistic that TiM will further advance the

doi:10.6342/NTU202401530

development of LLM reasoning capabilities.

2.2 Tropes in Movies

Tropes are tools used in creative works and are leveraged for automatic content creation assistance [4, 23], or to serve as a testbed for evaluating the reasoning skills of machine learning models [3, 24]. TiMoS [3] compiles movie synopses from the IMDb dataset and associates these with trope annotations from the TVTropes database. TiMoS serves as a benchmark to test NLP models and demonstrates that supervised models (e.g., BERT [6]) struggle to reason about tropes in movie synopses. Since these models access human-written synopses instead of the movie, simplifying the challenge of understanding visual inputs. In contrast, TrUMAn [24] utilizes video clips annotated with tropes from TVTropes to create a video trope reasoning dataset. However, reasoning from short clips is considerably simpler than from full movies. TiM utilizes a subset of the TiMoS dataset and associates it with movies collected from the MovieNet dataset [9], enabling the evaluation of video reasoning capabilities with long videos.

doi:10.6342/NTU202401530

Chapter 3 TiM Dataset

3.1 Overview

TiM comprises (1) 684 movies, each annotated with per-shot keyframes, subtitles, and trope labels, and (2) 95 trope identification queries accompanied by their definitions. The TiM dataset is specifically designed to pose more demanding and intricate reasoning tasks in video analysis, particularly focusing on extended content such as movies. The homepage of the TiM dataset[1] offers a download link for the TiM data along with detailed explanations of the annotations. Additionally, we have provided a pre-processing script for our baseline models in Section 4 to facilitate reproduction of our experimental results.

3.2 Trope

Considering the broad diversity of tropes, we utilize a set of 95 tropes categorized into four groups as introduced by TiMoS [3], depicted in Figure 3.1. Subsequent research could explore expanding the dataset by incorporating additional tropes. The categories used are Character Traits, Role Interaction, Situation, and Storyline. Character Traits analyze individual strengths and personalities, showing their impact on behavior and interactions

[1] https://ander1119.github.io/TiM/

doi:10.6342/NTU202401530

within the story. Role Interaction explores the dynamics between characters and their influence on the film's development. Situation covers specific scene-level scenarios that drive the plot with abstract concepts and emotional dynamics. Storyline focuses on the overall narrative structure, guiding the flow and thematic elements throughout the film. Together, these categories offer a comprehensive framework for analyzing the complex interplay of tropes in cinematic narratives.

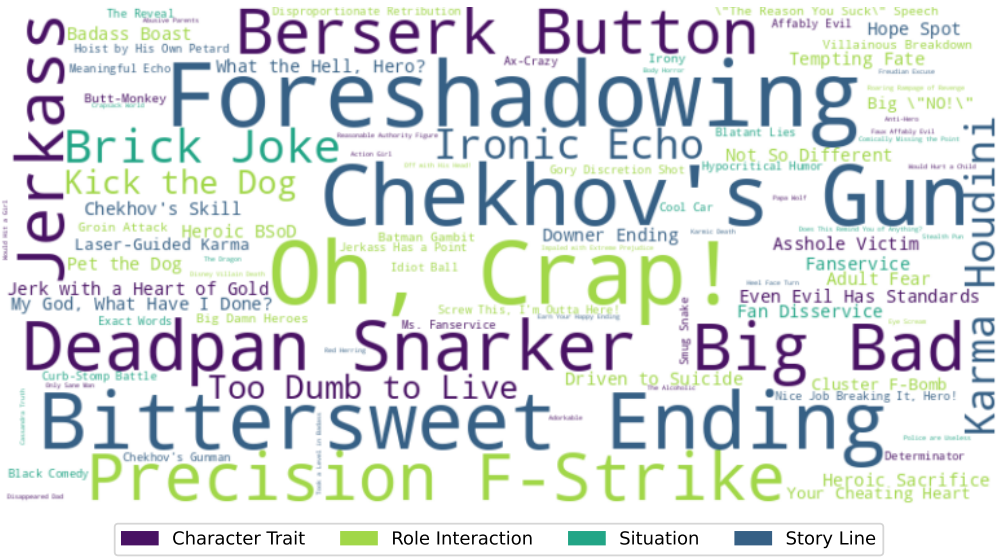


Figure 3.1: Word cloud of trope occurrences in Fullset, size of the tropes in proportion to their frequency in Fullset and color of the tropes correspond to the category they belongs

3.3 Task Definition

We formulate the task considered here as binary classification: $y = f(\text{movie}, \text{trope})$, where $y \in \{\text{True}, \text{False}\}$ indicates whether a given trope is present in the movie. This simplifies the task and enhances the focus on complex reasoning for single tropes in movies. Future research could consider revisiting the more challenging multi-label tasks [3].

doi:10.6342/NTU202401530

3.4 Evaluation

We have selected the micro F1 score as the primary metric for global comparison within the chosen set in TiM.

Table 3.1: Comparison between different experiment setups.

Setting	Movies	Frames	Subtitles		Tropes
			Line	Char	
Fullset	684	1545.7	-	-	11.91
VDset	246	1585.9	1587.4	56k	13.38
Mainset	50	1699.6	1822.2	65k	6.08

3.5 Data Collection

We sourced trope occurrences in movies from the TiMoS dataset [3], originally compiled from the TVTropes database. Movie frames and subtitles were gathered from the MovieNet dataset [9]. We aligned the movies with their corresponding tropes using their IMDb IDs. Future research could extend this dataset by collecting more movies.

3.6 Data Statistics

This benchmark is tailored for LLM-based methods, utilizing the entire dataset as the test set. Supervised learning evaluations are conducted using 5-fold cross-validation. To accommodate the absence of some subtitles in the MovieNet dataset, we offer the *VDset*, which includes subtitles. Additionally, the *Mainset*—a subset of 50 movies—is provided for more detailed analysis as experiments may require additional time or resources. Table 3.1 presents a comparative analysis of different experimental setups.

doi:10.6342/NTU202401530

doi:10.6342/NTU202401530

Chapter 4 Experiments

4.1 Baselines

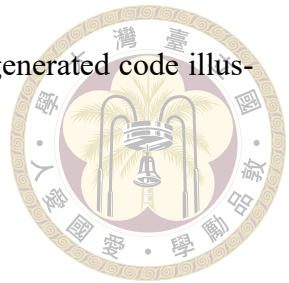
Captioner-Reasoner We tested LLoVi [37], which addresses video reasoning by tokenizing frames using VLMs such as BLIP-2 [13]. This efficient approach allowed LLoVi to achieve an accuracy of 67.7 on NExT-QA [29]. Given its success, LLoVi shows potential for handling more complex, long-range video QA tasks by effectively summarizing captions.

Large Multimodal Model Instruction Fine-tuning SEVILA [33] introduces a two-stage pipeline that utilizes fine-tuned large multimodal models to localize keyframes and apply reasoning to selected frames, achieving an accuracy of 73.8 on NExT-QA with only 4 frames used for sparse sampling as inputs. Considering that TiM might require more input frames, we also incorporate LLaMA-VID [16], which adopts a different strategy by projecting frames into two tokens to efficiently handle long-range video inputs.

Visual Programming ViperGPT [25] leverages LLMs as a code generator that dynamically allocates VLMs and vision models, such as object detection, to progressively derive reasoning results. Although Viper may not always outperform LLoVi and SEVILA in

doi:10.6342/NTU202401530

terms of performance, it offers superior interpretability because the generated code illustrates how LLMs decompose tasks and perform stepwise reasoning.



Gemini 1.5 To assess the limits of machine learning models, we tested Gemini 1.5 [27], a trillion-scale model that significantly surpasses the size of previously mentioned models. This serves as a benchmark for future research.

4.2 Proposed Method

In our initial approach to TiM, we enhanced Viper [25] with two novel features designed to address Abstract Perception and Long-range Compositional Reasoning respectively.

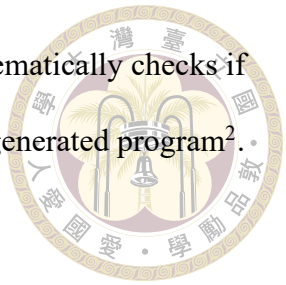
Face-Enhanced Viper of Role Interactions (FEVoRI) Previous datasets have primarily focused on short, simple clips rather than movies featuring numerous characters with rich interactions. Consequently, the original Viper design lacked tools specifically aimed at role identification. FEVoRI augments Viper by providing a face detection tool with examples in the prompts. FEVoRI enhances the fine-grained understanding of the "human" object to address Abstract Perception[1].

Context Query Reduction (ConQueR) Viper [25] processes NExT-QA [29] by temporally locating frames or objects and querying the VLM about them. This approach struggles with TiM due to the intricate narratives of movies and the complex definitions of tropes. ConQueR addresses the Long-range Compositional Reasoning challenge by pro-

[1] Implementation details in Appendix B.1

doi:10.6342/NTU202401530

gressively decomposing the narrative context and trope query. It systematically checks if the extracted context matches each dimension of a trope through the generated program[2].



4.3 Setup

Most models in our experiments are training-free, so the entire TiM dataset is used for testing. Additionally, we fine-tuned SeViLa on TiM in a supervised setting to evaluate its performance. For these experiments, we employed five-fold cross-validation and reported the average performance. For LLoVi [36], we employ the standard prompt with BLIP-2 [13] to generate captions for each frame in every shot of TiM. This is followed by a multi-round summarizing process to create a summary for each movie. These summaries, coupled with binary classification queries, are then inputted into an LLM to generate answers. In the multi-modality version, we enhance the summarization process by integrating captions with subtitles. For SeViLA [33], we use NExT-QA setting for both zero-shot and fine-tuned scenarios, enabling the Localizer to select 16 frames from a set of 120. These selected frames are used to address binary classification queries, with an enhancement in the multi-modality version where visual features are concatenated with subtitles before being processed by the LLM during both the localizer and answerer stages. For LLaMA-VID [15], we use long-video-tuning model, which was tuned with QA pair from MovieNet [9], to inference on binary classification query on TiM. For Viper and our proposed method, we have adapted the NExT-QA prompt on TiM and use GPT-4 as the code generator. For FEVoRI, we have integrated additional face identification tools into the Viper API, specifically employing DeepFace [22] for face recognition.

[2]Implementation details in Appendix B.2

doi:10.6342/NTU202401530

4.4 State-of-the-art Comparison

Table 4.1: State-of-the-art performance on TiM. everyshot: the model takes one frame per shot. SeViLA†: SeViLA that uses the zero-shot localizer. 120→ 16: SeViLA localizer selects 16 keyframes from 120 frames. 16 (SeViLA): Viper uses 16 frames selected by SeViLA localizer. FEVoRI*: evaluate on *Mainset*. Human: human evaluation result from [3]. we select Mainset as multi-modality setting for fair comparison

Modality	Method	# Frames	Pre.	Rec.	F1	Category F1			
						CT	RI	ST	SL
V(Fullset)	Random	-	12.24	48.48	19.54	19.23	19.99	17.37	23.37
	LLoVi [37]	everyshot	20.47	17.67	18.97	13.46	16,67	15.22	25.58
	SeViLA† [33]	120→ 16	12.35	96.71	21.90	25.12	19.02	22.38	20.96
	SeViLA [33]	120→ 16	15.29	51.75	23.61	23.46	23.43	17.81	27.58
	Viper [26]	16 (SeViLA†)	13.26	67.33	22.15	21.58	22.63	19.92	24.60
	Viper [26]	16 (SeViLA)	14.09	68.70	23.39	21.41	24.62	20.90	26.85
	FEVoRI*	120	27.07	32.32	29.42	12.36	22.75	35.62	48.78
	Gemini 1.5 [27]	120	38.37	34.42	40.74	40.45	38.79	38.55	45.11
V+D(Mainset[3])	Random	-	14.14	50.08	22.06	20.26	21.24	19.50	23.92
	LLoVi [37]	everyshot	31.35	17.21	18.78	20.20	24.40	35.95	40.63
	SeViLA† [33]	120→ 16	17.30	89.33	28.98	22.64	24.76	32.83	35.79
	SeViLA [33]	120→ 16	22.98	58.18	28.54	28.92	25.00	37.50	42.86
	LLaMA-VID [15]	240	15.56	90.12	26.53	25.72	24.60	28.31	38.15
	Viper [26]	16 (SeViLA†)	14.58	37.87	21.05	18.15	14.35	20.58	31.56
	Viper [26]	16 (SeViLA)	14.38	38.79	20.98	24.39	15.22	18.02	24.76
	Viper [26]	120	27.78	21.74	24.39	22.91	19.59	40.43	48.78
	FEVoRI	120	27.88	39.80	32.79	30.52	29.55	**42.42**	49.67
	FEVoRI+ConQueR	120	32.11	51.28	**39.64**	**42.80**	**34.48**	39.78	**55.17**
Synopses	Human [3]	-	65.77	63.98	64.87	-	-	-	-

As shown in Table 4.1, all LLM-based baselines struggle with reasoning on TiM, achieving only random-level performance (first row of each block). This underscores that despite their significant achievements on various video reasoning benchmarks, state-of-the-art models are unable to overcome the challenges posed by TiM. Access to dialogues results in an F1 score improvement of 2-4 points. Captioner-Reasoner (LLoVi [36]) records lower F1 scores, indicating that the loss of information or the abstraction gap during video captioning may lead to subpar performance on TiM. LLoVi also achieves relatively better performance in the Storyline (SL) category, which focuses on the overall

[3]Performance difference between Mainset and VDset in Appendix 6

doi:10.6342/NTU202401530

plot rather than on fine-grained details. LMM-IF methods, including SeViLa [33] which achieves significant performance on various benchmarks, and LLaMA-VID [16] designed for long videos, often resort to blindly guessing "yes." This approach typically results in high recall but poor precision. Fine-tuning SeViLa enhances performance through supervised learning. Viper [25] achieves decent performance without resorting to blindly guessing "yes," and maintains superior precision compared to SeViLa and LLaMA-VID. Gemini [27] achieves a 41 F1 score, surpassing all previously mentioned methods due to its larger scale of parameters and training data. However, it still significantly trails human performance [3], scoring 41 compared to 65 F1. Comprehensive experiments show that SOTA LLMs still struggle to address challenges in TiM.

4.5 FEVoRI Analysis

Table 4.2: Ablation study on FEVoRI framework on TiM *Mainset*.

	Modality	# Frames	VLM	Coder	Pre.	Rec.	F1	Category F1			
								CT	RI	ST	SL
FEVoRI	V+D	120	BLIP-2	GPT-4	27.88	39.80	32.79	30.52	29.55	42.42	49.67
	V	120	BLIP-2	GPT-4	27.07	32.23	29.42 ((-3.37↓))	12.36	22.75	35.62	48.00
	V+D	**everyshot**	BLIP-2	GPT-4	27.27	46.15	34.29 ((+1.50↑))	33.30	30.12	44.68	50.00
	V+D	**16**	BLIP-2	GPT-4	25.71	40.72	31.52 ((-1.27↓))	23.74	25.56	38.83	47.54
	V+D	120	**Gemini**	GPT-4	29.37	51.15	37.31 ((+4.52↑))	28.71	29.49	47.17	53.23
	V+D	120	BLIP-2	**GPT-3.5**	30.16	35.52	32.62 ((-0.17↓))	27.18	30.34	39.56	38.65

FEVoRI significantly boosts the F1 score by 8.5. Comparing Viper and FEVoRI in the second block of Table 4.1, our augmentation allows the VP LLM to understand character interactions, leading to substantial performance improvements, particularly in the CT (Character Traits) and RI (Role Interaction) categories, where fine-grained role interactions are crucial. Remarkably, even the visual-only FEVoRI outperforms the supervised SeViLA [33], demonstrating the superior design of our methodology. As the performance

doi:10.6342/NTU202401530

gain is primarily from an 18.00 improvement in recall, while precision improves by only 0.1, we hypothesize that FEVoRI improves by effectively identifying more relevant cases.

ConQueR further increases the F1 score by 6.9. In the second block of Table 4.1 , comparing FEVoRI and FEVoRI+ConQueR, the modified ConQueR demonstrates how progressively decomposing the trope query and movie narrative context enhances understanding. ConQueR also effectively filters key signals to extract crucial information from long-range videos. The performance improvement highlights promising directions for future work in addressing Long-range Compositional Reasoning.

A higher frame rate consistently outperforms sparse sampling. Several tropes depend on fleeting, fine-grained details or a comprehensive understanding of the entire plot. We compared the density of frame sampling by evaluating every shot (approximately 1,000 frames) and 120 frames per video, alongside a sparse sampling method that uses only 16 frames per video, which is commonly used in many approaches. As shown in Table 4.2, a higher frame rate leads to marginal yet consistent improvements, with every-shot sampling boosting the F1 score by 2.8 points across all categories. This indicates that while sparse sampling is efficient, it may compromise performance.

Enhancing VLM Abstract Perception improves performance by 4.5. A core challenge of TiM is Abstract Perception, which involves tokenizing visual signals into coherent concepts. Table 4.2 shows that replacing BLIP-2 [13] with more advanced Gemini [27], the F1 score is boosted by 4.5 as Gemini is capable to tackle more abstract queries.

doi:10.6342/NTU202401530

GPT-4 shows a slight improvement over GPT-3.5 in program generation. When replacing GPT-4 with GPT-3.5, the F1 score drops by 0.2, as shown in Table 4.2, demonstrating that GPT-3.5 is capable to generate programs without ConQueR.

doi:10.6342/NTU202401530

doi:10.6342/NTU202401530

Chapter 5 Code Analysis

5.1 Abstract Syntax Tree (AST) for Visual Programming

While Section 4 effectively highlights the challenges of Abstract Perception and Long-range Compositional Reasoning encountered with TiM, it is challenging to quantify the degree of the challenge. Hence, we propose an evaluation protocol to assess the degree of Abstract Perception and Long-range Compositional Reasoning, leveraging the Abstract Syntax Tree (AST) of VP code. AST is a tree structure that represents the syntactic structure of a code snippet, thereby reflecting the complexity of the reasoning task addressed by VP. By decomposing VP code into an AST, we can assess the level of Abstract Perception by measuring VLM calls and the level of Long-range Compositional Reasoning by analyzing the nodes and edges within the AST. More nodes indicate higher syntactic complexity, while more edges signify intricate relationships between code constructs. This detailed analysis provides insights into the sophistication of the logic used, making AST a valuable tool for evaluating the intricacies of VP tasks. Therefore, we propose a novel framework based on AST to analyze the Abstract Perception and Long-range Compositional Reasoning level of a VP task based on generated code[5].

[5] Implementation details in B.3

doi:10.6342/NTU202401530

Table 5.1: We propose an AST Based Code Dignosis (ABCD) to assess the levels of Abstract Perception and Long-range Compositional Reasoning in a dataset, using code generated by VP. A higher number indicates greater complexity and challenge. (Section 5)

Dataset	Abstract Perception		Long-range Compositional Reasoning	
	VLM Calls	VLM Tokens	AST Nodes	AST Edges
NExT-QA [29]	1.60	11.15	102.09	146.32
GQA [10]	1.34	12.69	42.16	55.63
OKVQA [20]	1.66	13.75	42.50	58.46
TiM (w/o ConQueR)	1.77	14.11	123.19	178.01
TiM (w/ ConQueR)	**1.97**	**20.67**	**141.81**	**205.06**

5.2 AST Based Code Diagnosis (ABCD)

Abstract Perception Level Analysis **VLM calls** serve as the primary interface for connecting visual inputs and transferring them to language representations. The frequency of VLM calls reflects the abstraction requirements for a visual programming task. **VLM Tokens** indicate the complexity of VLM calls, which can vary from simple questions like ”What is it doing?” to more complex and abstract inquiries such as ”What is caused by the person doing action?” Facing the challenge of directly assessing the Abstract Perception level of a VLM query, we have developed a proxy method. This approach estimates the token length of a VLM call, based on the premise that more abstract concepts generally require a greater number of tokens for explanation in VLM models.

Long-range Compositional Reasoning Level Analysis **AST Nodes** represent a construct like statements, expressions, or operators, which when analyzed collectively through the count of nodes, provides a quantitative measure of the code’s structural complexity. Essentially, each node encapsulates a specific element or operation in the code, and more nodes typically indicate more constructs and interactions. Therefore, a higher count of nodes in visual programming often indicates a more complex and intricate codebase, filled

doi:10.6342/NTU202401530

with numerous functional components and logical constructs, necessitated by tasks that require a higher level of Long-range Compositional Reasoning. **AST Edges** denote the relationships between nodes, which are vital for understanding the structural and logical organization of code. Each edge connects nodes in a way that reflects the syntactic dependencies and execution order within the program, effectively mapping out the flow of control and data. A higher number of edges generally indicates a more complex interplay of these dependencies, suggesting more intricate code logic and increased interactions among the program's components. Thus, in VP, a dense network of AST edges usually points to sophisticated program constructs and a higher degree of Long-range Compositional Reasoning, as tasks often necessitate nuanced combinations and sequences of operations to achieve desired functionalities and outcomes.

5.3 Results

As shown in Table 5.1, it is clear that TiM requires a higher level of both Abstract Perception and Long-range Compositional Reasoning, even without ConQueR. Regarding Abstract Perception, TiM requires more VLM calls and a greater number of tokens to effectively process visual inputs from videos. As for Long-range Compositional Reasoning, this results in a higher number of AST nodes and edges. Furthermore, adopting ConQueR not only increases AST nodes and edges but also adds to the number of VLM calls and tokens. This analysis not only measures performance but also examines Abstract Perception and Long-range Compositional Reasoning based on the complexity of the generated programs.

doi:10.6342/NTU202401530

doi:10.6342/NTU202401530

Chapter 6 Conclusion

We introduce a novel task, TiM, accompanied by a new dataset designed to test the challenges of Abstract Perception and Long-range Compositional Reasoning. Our findings reveal that SOTA LLM-based methods such as Captioner-Reasoner, Large Multimodal Model Instruction Fine-tuning, and Visual Programming, lack the capabilities to meet these challenges effectively. To enhance performance, we have augmented the VP model [25] with FEVoRI and ConQueR, achieving a 15-point improvement in F1 score. Additionally, we propose a new protocol, ABCD, to assess the Abstract Perception and Long-range Compositional Reasoning levels of datasets using code generated by VP. We believe that TiM could serve as a valuable testbed for the development and refinement of novel LLM-based video reasoning methods.

doi:10.6342/NTU202401530

doi:10.6342/NTU202401530

References

[1] M. BCS. Star: A benchmark for situated reasoning in real-world videos.

[2] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. Advances in neural information processing systems, 33:1877–1901, 2020.

[3] C.-H. Chang, H.-T. Su, J.-H. Hsu, Y.-S. Wang, Y.-C. Chang, Z. Y. Liu, Y.-L. Chang, W.-F. Cheng, K.-J. Wang, and W. H. Hsu. Situation and behavior understanding by trope detection on films. In Proceedings of the Web Conference 2021, pages 3188–3198, 2021.

[4] J.-P. Chou, A. F. Siu, N. Lipka, R. Rossi, F. Dernoncourt, and M. Agrawala. Talestream: Supporting story ideation with trope knowledge. In Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology, pages 1–12, 2023.

[5] J. Chung and Y. Yu. Long story short: a summarize-then-search method for long video question answering. 2023.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational

doi:10.6342/NTU202401530

Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, 2019. Association for Computational Linguistics.

[7] J. S. B. Evans. In two minds: dual-process accounts of reasoning. Trends in cognitive sciences, 7(10):454–459, 2003.

[8] T. Gupta and A. Kembhavi. Visual programming: Compositional visual reasoning without training. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 14953–14962, 2023.

[9] Q. Huang, Y. Xiong, A. Rao, J. Wang, and D. Lin. Movienet: A holistic dataset for movie understanding. In Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IV 16, pages 709–727. Springer, 2020.

[10] D. A. Hudson and C. D. Manning. Gqa: A new dataset for real-world visual reasoning and compositional question answering. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 6700–6709, 2019.

[11] J. Lei, L. Yu, M. Bansal, and T. Berg. Tvqa: Localized, compositional video question answering. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 1369–1379, 2018.

[12] J. Lei, L. Yu, T. Berg, and M. Bansal. Tvqa+: Spatio-temporal grounding for video question answering. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 8211–8225, 2020.

[13] J. Li, D. Li, S. Savarese, and S. Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In International conference on machine learning, pages 19730–19742. PMLR, 2023.

doi:10.6342/NTU202401530

[14] J. Li, L. Niu, and L. Zhang. From representation to reasoning: Towards both evidence and commonsense reasoning for video question-answering. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 21273–21282, 2022.

[15] Y. Li, C. Wang, and J. Jia. Llama-vid: An image is worth 2 tokens in large language models. arXiv preprint arXiv:2311.17043, 2023.

[16] Y. Li, C. Wang, and J. Jia. Llama-vid: An image is worth 2 tokens in large language models. arXiv preprint arXiv:2311.17043, 2023.

[17] X. Lin, G. Bertasius, J. Wang, S.-F. Chang, D. Parikh, and L. Torresani. Vx2text: End-to-end learning of video-based text generation from multimodal inputs. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 7005–7015, 2021.

[18] X. Lin, S. Tiwari, S. Huang, M. Li, M. Z. Shou, H. Ji, and S.-F. Chang. Towards fast adaptation of pretrained contrastive models for multi-channel video-language retrieval. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 14846–14855, 2023.

[19] K. Mangalam, R. Akshulakov, and J. Malik. Egoschema: A diagnostic benchmark for very long-form video language understanding. Advances in Neural Information Processing Systems, 36, 2024.

[20] K. Marino, M. Rastegari, A. Farhadi, and R. Mottaghi. Ok-vqa: A visual question answering benchmark requiring external knowledge. In Proceedings of the IEEE/cvf conference on computer vision and pattern recognition, pages 3195–3204, 2019.

doi:10.6342/NTU202401530

[21] R. OpenAI. Gpt-4 technical report. arxiv 2303.08774. View in Article, 2:3, 2023.

[22] S. I. Serengil and A. Ozpinar. Hyperextended lightface: A facial attribute analysis framework. In 2021 International Conference on Engineering and Emerging Technologies (ICEET), pages 1–4. IEEE, 2021.

[23] J. R. Smith, D. Joshi, B. Huet, W. Hsu, and J. Cota. Harnessing ai for augmenting creativity: Application to movie trailer creation. In Proceedings of the 25th ACM international conference on Multimedia, pages 1799–1808, 2017.

[24] H.-T. Su, P.-W. Shen, B.-C. Tsai, W.-F. Cheng, K.-J. Wang, and W. H. Hsu. Truman: Trope understanding in movies and animations. In Proceedings of the 30th ACM International Conference on Information & Knowledge Management, pages 4594–4603, 2021.

[25] D. Surís, S. Menon, and C. Vondrick. Vipergpt: Visual inference via python execution for reasoning. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 11888–11898, 2023.

[26] D. Surís, S. Menon, and C. Vondrick. Vipergpt: Visual inference via python execution for reasoning. In ICCV, 2023.

[27] G. Team, R. Anil, S. Borgeaud, Y. Wu, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth, et al. Gemini: a family of highly capable multimodal models. arXiv preprint arXiv:2312.11805, 2023.

[28] Z. Wang, M. Li, R. Xu, L. Zhou, J. Lei, X. Lin, S. Wang, Z. Yang, C. Zhu, D. Hoiem, et al. Language models with image descriptors are strong few-shot video-language learners. Advances in Neural Information Processing Systems, 35:8483–8497, 2022.

doi:10.6342/NTU202401530

[29] J. Xiao, X. Shang, A. Yao, and T.-S. Chua. Next-qa: Next phase of question-answering to explaining temporal actions. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 9777–9786, 2021.

[30] J. Xiao, A. Yao, Z. Liu, Y. Li, W. Ji, and T.-S. Chua. Video as conditional graph hierarchy for multi-granular question answering. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 36, pages 2804–2812, 2022.

[31] J. Xiao, P. Zhou, T.-S. Chua, and S. Yan. Video graph transformer for video question answering. In European Conference on Computer Vision, pages 39–58. Springer, 2022.

[32] D. Xu, Z. Zhao, J. Xiao, F. Wu, H. Zhang, X. He, and Y. Zhuang. Video question answering via gradually refined attention over appearance and motion. In Proceedings of the 25th ACM international conference on Multimedia, pages 1645–1653, 2017.

[33] S. Yu, J. Cho, P. Yadav, and M. Bansal. Self-chained image-language model for video localization and question answering. In NeurIPS, 2023.

[34] Z. Yu, D. Xu, J. Yu, T. Yu, Z. Zhao, Y. Zhuang, and D. Tao. Activitynet-qa: A dataset for understanding complex web videos via question answering. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 33, pages 9127–9134, 2019.

[35] K.-H. Zeng, T.-H. Chen, C.-Y. Chuang, Y.-H. Liao, J. C. Niebles, and M. Sun. Leveraging video descriptions to learn video question answering. In Proceedings of the AAAI conference on artificial intelligence, volume 31, 2017.

[36] C. Zhang, T. Lu, M. M. Islam, Z. Wang, S. Yu, M. Bansal, and G. Bertasius. A simple llm framework for long-range video question-answering. arXiv preprint arXiv:2312.17235, 2023.

doi:10.6342/NTU202401530

[37] C. Zhang, T. Lu, M. M. Islam, Z. Wang, S. Yu, M. Bansal, and G. Bertasius. A simple llm framework for long-range video question-answering. arXiv preprint arXiv:2312.17235, 2023.

[38] H. Zhang, X. Li, and L. Bing. Video-llama: An instruction-tuned audio-visual language model for video understanding. arXiv preprint arXiv:2306.02858, 2023.

[39] H. Zhang, Y. Liu, L. Dong, Y. Huang, Z.-H. Ling, Y. Wang, L. Wang, and Y. Qiao. Movqa: A benchmark of versatile question-answering for long-form movie understanding. arXiv preprint arXiv:2312.04817, 2023.

doi:10.6342/NTU202401530

Appendix A — Dataset Analysis

A.1 Performance Comparison

Table A.1 displays the performance difference between the Mainset and VDset for the baseline models we selected. The gap is relatively small, and for a fair comparison, we have chosen Mainset as the multi-modality setting in Table 4.1.

Table A.1: State-of-the-art performance on multi-modality settings (VDset and Mainset). everyshot: the model takes one frame per shot. SeViLA†: SeViLA that uses the zero-shot localizer. 120→ 16: SeViLA localizer selects 16 keyframes from 120 frames. $16_{(\text{SeViLA})}$: Viper uses 16 frames selected by SeViLA localizer.

Modality	Method	# Frames	Pre.	Rec.	F1	Category F1			
						CT	RI	ST	SL
VDset	LLoVi [37]	everyshot	19.23	21.73	20.40	20.85	24.87	19.49	31.62
	SeViLA† [33]	120→16	14.82	92.97	25.56	24.34	27.50	19.85	29.50
	SeViLA [33]	120→16	16.32	65.21	26.11	26.08	28.89	18.29	28.57
	LLaMA-VID [15]	240	14.47	98.30	25.22	24.74	26.34	19.85	29.27
	Viper [26]	$16_{(\text{SeViLA}^{\dagger})}$	16.08	46.24	23.86	17.78	24.06	19.65	31.21
	Viper [26]	$16_{(\text{SeViLA})}$	16.48	44.41	24.04	21.01	27.30	19.72	27.79
Mainset	LLoVi [37]	everyshot	31.35	17.21	18.78	20.20	24.40	35.95	40.63
	SeViLA† [33]	120→16	17.30	89.33	28.98	22.64	24.76	32.83	35.79
	SeViLA [33]	120→16	22.98	58.18	28.54	28.92	25.00	37.50	42.86
	LLaMA-VID [15]	240	15.56	90.12	26.53	25.72	24.60	28.31	38.15
	Viper [26]	$16_{(\text{SeViLA}^{\dagger})}$	14.58	37.87	21.05	18.15	14.35	20.58	31.56
	Viper [26]	$16_{(\text{SeViLA})}$	14.38	38.79	20.98	24.39	15.22	18.02	24.76

doi:10.6342/NTU202401530

Figure A.1: Character-Trait

Figure A.2: Role-Interaction

Figure A.3: Situation

Figure A.4: Storyline

Figure A.5: Word cloud of trope occurrences in four category, size of the tropes in proportion to their frequency in Fullset

A.2 Trope

In A.5 we present word cloud visualizations to illustrate the distribution of trope occurrences across four distinct categories in our dataset. The categories include Character-Trait, Role-Interaction, Situation, and Storyline. Each word cloud represents the frequency of tropes, with larger words indicating higher occurrence

Character Trait A.1 visualizes the various traits that define character personalities and behaviors. Prominent tropes such as "Big Bad," "Berserk Button," and "Jerkass" are highly frequent, indicating their common use in character development.

Role Interaction A.2 showcases the dynamics between characters and their roles. Notable tropes like "Pet the Dog," "Heroic Sacrifice," and "Oh, Crap!" dominate this category, highlighting key interactions that drive the narrative.

doi:10.6342/NTU202401530

Situation Situational tropes are visualized in A.3, with "Fan Disservice," "Irony," and "Brick Joke" being some of the most frequent. These tropes often set the stage for pivotal moments within the story.

Storyline A.4 depicts the storyline-related tropes, such as "Foreshadowing," "Chekhov's Gun," and "Bittersweet Ending." These elements are crucial for the progression and structure of the narrative. To avoid missing tropes due to low frequency, we also present all tropes in each category in the A.2:

Table A.2: Trope list in each category

Category	Tropes		
CT	Big Bad	Jerkass	Faux Affably Evil
	Smug Snake	Abusive Parents	Would Hurt a Child
	Action Girl	Adorkable	Even Evil Has Standards
	Deadpan Snarker	Determinator	Only Sane Man
	Anti-Hero	Asshole Victim	Jerk with a Heart of Gold
	Papa Wolf	Affably Evil	Too Dumb to Live
	Butt-Monkey	Ax-Crazy	Reasonable Authority Figure
	Berserk Button	Ms. Fanservice	The Alcoholic
	Disappeared Dad	Would Hit a Girl	
RI	Oh, Crap!	Driven to Suicide	Adult Fear
	Not So Different	Heroic BSoD	"The Reason You Suck" Speech
	Eye Scream	Gory Discretion Shot	Impaled with Extreme Prejudice
	Off with His Head!	Disney Villain Death	Your Cheating Heart
	Big "NO!"	Tempting Fate	Disproportionate Retribution
	Badass Boast	Groin Attack	Roaring Rampage of Revenge
	Big Damn Heroes	Heroic Sacrifice	Screw This, I'm Outta Here!
	Kick the Dog	Pet the Dog	Villainous Breakdown
	Precision F-Strike	Cluster F-Bomb	Jerkass Has a Point
	Idiot Ball	Batman Gambit	
ST	Police are Useless	The Dragon	Comically Missing the Point
	Body Horror	The Reveal	Curb-Stomp Battle
	Cassandra Truth	Blatant Lies	Crapsack World
	Cool Car	Fanservice	Fan Disservice
	Brick Joke	Hypocritical Humor	Does This Remind You of Anything?
	Black Comedy	Irony	Exact Words
	Stealth Pun		
SL	Bittersweet Ending	Karma Houdini	Earn Your Happy Ending
	Laser-Guided Karma	Downer Ending	Hoist by His Own Petard
	Ironic Echo	Chekhov's Skill	What the Hell, Hero?
	Hope Spot	Heel Face Turn	Took a Level in Badass
	Chekhov's Gun	Foreshadowing	My God, What Have I Done?
	Chekhov's Gunman	Red Herring	Nice Job Breaking It, Hero!
	Karmic Death	Meaningful Echo	Freudian Excuse

doi:10.6342/NTU202401530

A.3 Movie

In this section, we present various statistics related to the TiM dataset, focusing on trope counts per movie, the number of shots, and subtitle counts in terms of lines and characters. These statistics provide a comprehensive overview of the dataset's characteristics.

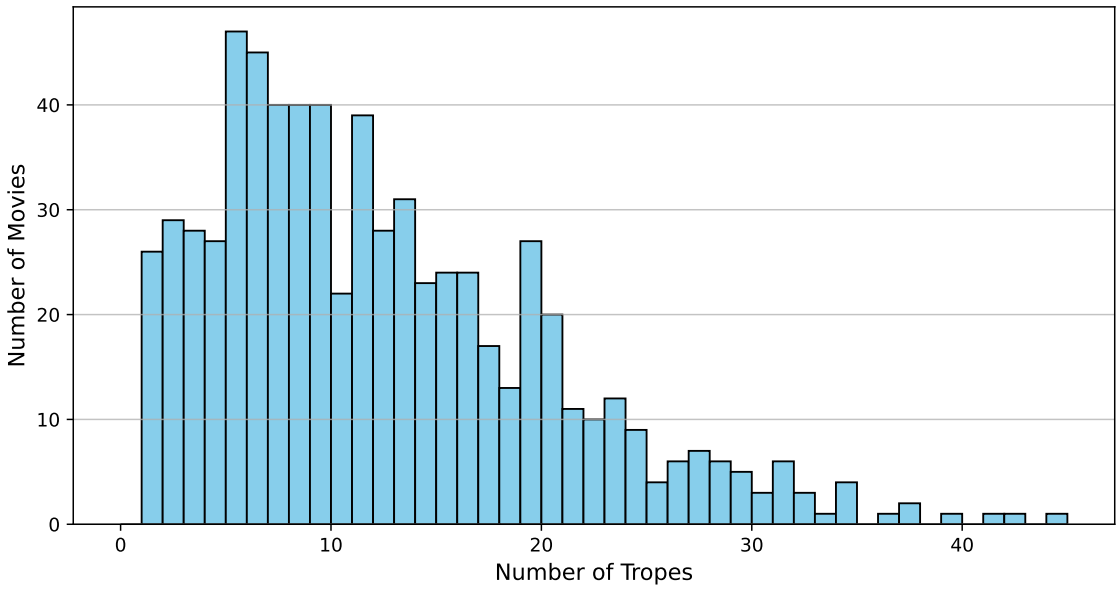


Figure A.6: Histogram of trope counts per movie on *Fullset*

Trope Counts per Movie on Fullset A.6 displays the distribution of trope counts across movies in the Fullset. The distribution shows that most movies contain between 5 and 15 tropes, with a few movies having significantly more or fewer tropes. This variation highlights the diverse narrative complexity within the dataset.

Shot Counts per Movie on Fullset A.7 illustrates the number of shots per movie in the Fullset. The distribution is approximately normal, with most movies having between 500 and 2000 shots. This metric is crucial for understanding the visual dynamism and pacing of the films in the dataset.

doi:10.6342/NTU202401530

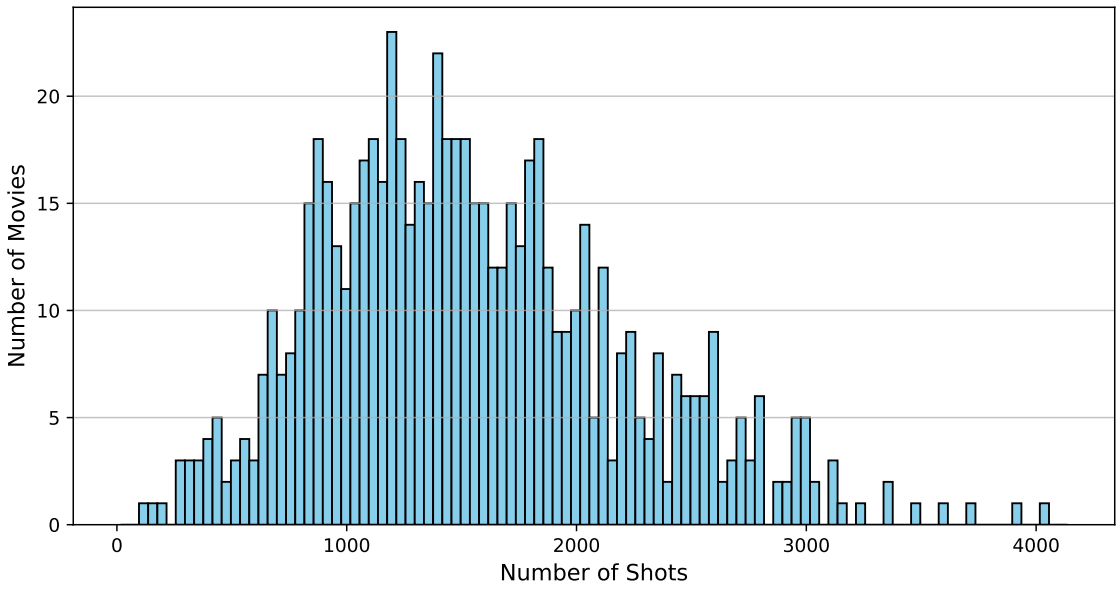


Figure A.7: Histogram of shot counts per movie on *Fullset*

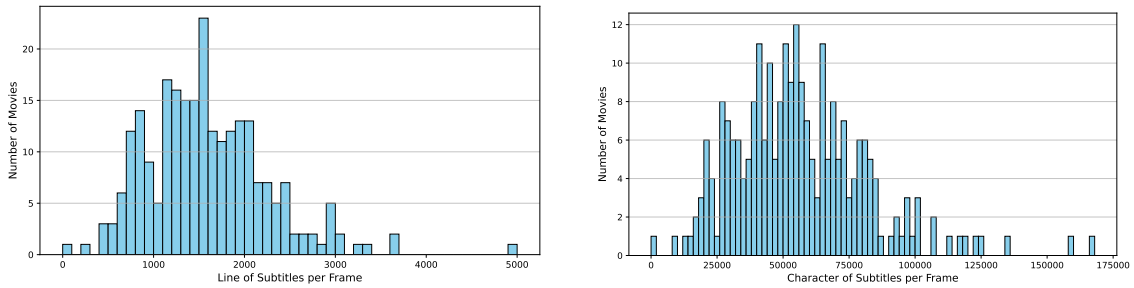


Figure A.8: Histogram of subtitle counts per Movie on *VDset*: Lines (left) vs. Characters (right)

Subtitle Counts per Movie on VDtest A.8 provides two histograms comparing the subtitle counts per movie in VDtest. The left histogram shows the distribution of subtitle lines, while the right histogram presents the distribution of subtitle characters. Both distributions indicate that most movies have between 500 and 1500 lines of subtitles, corresponding to between 50,000 and 150,000 characters. These metrics are essential for evaluating the dialogue density and the amount of textual information available in the movies. Since not all movies in the Fullset have subtitles, we used the VDset instead for these statistics.

doi:10.6342/NTU202401530

doi:10.6342/NTU202401530

Appendix B — Implementation Details

B.1 FEVoRI

```
1  def execute_command(video, annotation, possible_answers, query):
2      # Trope: Big Bad
3      # Definition: The character who is the direct cause of all of the bad happenings in a story.
4      # Thought Process:
5      # 1. Frame Selection: Analyze each frame to identify key characters and their actions.
6      # 2. Character Analysis: Identify the main antagonist and their actions throughout the video.
7      # 3. Answer Selection: Determine if there is a single character causing most of the negative events.
8
9      video_segment = VideoSegment(video, annotation)
10     info = {
11         "character_actions": {},
12         "negative_impacts": {}
13     }
14     for i, frame in enumerate(video_segment.frame_iterator()):
15         # Identify all characters in the frame
16         for character in frame.find("person"):
17             character_id = video_segment.face_identify(character)
18             if character_id is None:
19                 continue
20             # Query the action of the character in the frame
21             action_query = frame.simple_query("What is this person doing?")
22             # Check if the action has a negative impact
23             negative_query = f"Does the action '{action_query}' have a negative impact?"
24             has_negative_impact = frame.llm_query(negative_query, to_yesno=True)
25             # Store character actions and their impacts
26             if character_id not in info["character_actions"]:
27                 info["character_actions"][character_id] = []
28             info["character_actions"][character_id].append(action_query)
29             if "yes" in has_negative_impact.lower():
30                 if character_id not in info["negative_impacts"]:
31                     info["negative_impacts"][character_id] = 0
32                 info["negative_impacts"][character_id] += 1
33
34     # After collecting information, use it to determine the presence of the trope
35     answer, reason = video_segment.select_answer(info, query, possible_answers)
36     return answer, reason, info
```

Listing B.1: FEVoRI ICL Example

We have integrated *face_identify*, which utilizes DeepFace [22] to assign a unique ID to each character. As shown in Line 17, FEVoRI enhances fine-grained tokenization, extending beyond the generic object ”human” to more effectively address Abstract Perception.

doi:10.6342/NTU202401530

B.2 ConQueR

```
def execute_command(video, annotation, possible_answers, query)->[str, str, dict]:
    # Trope: Big Bad
    # Definition: The character who is the direct cause of all of the bad happenings in a story.
    # Thought Process:
    # 1. Character Identification: Identify characters and track their actions across frames.
    # 2. Event Linking: Determine which negative events are directly caused by the actions of a character.
    # 3. Consistency Check: Check for consistency in the character's negative influence over the story arc.
    video_segment = VideoSegment(video, annotation)
    # Initialize a dictionary to store information collected during analysis
    info = {
        "happened bad events": {},
        "character infos": {}
    }
    for i, frame in enumerate(video_segment.frame_iterator()):
        for person in frame.find("person"):
            # identify the person in the frame
            person_id = video_segment.face_identify(person)
            if person_id is None:
                continue
            # query the character"s description and add into character_description
            if person_id not in info["character infos"]:
                descriptino_query = "Please describe his/her appearance in 10 words"
                character_description = person.simple_query(descriptino_query)
                info["character infos"][person_id] = {
                    "description": character_description,
                    "actions": {}
                }
            # query the character"s action in the frame
            action = person.simple_query("Please describe his/her action in the scene")
            info["character infos"][person_id]["actions"][f"{i} frame"] = action
        # check if there is any negative event happening in the scene
        check_negative_query = "Is there any negative event happening in the scene?"
        any_negative_event = frame.simple_query(check_negative_query, to_yesno=True)
        if "yes" in any_negative_event.lower():
            # query the negative events happening in the scene
            event = frame.simple_query("What's happening in the scene")
            info["happened bad events"][f"{i} frame"] = {
                "event": event,
                "potential cause": []
            }
            for pid, character_infos in info["character infos"].items():
                # check if the character is a potential cause of the negative event
                character_description = character_infos["description"]
                for prev_i in range(i, max(i-5, 0), -1):
                    prev_action = character_infos["actions"].get(f"{prev_i} frame", None)
                    if prev_action is not None:
                        person_query = f"Is person with '{character_description}' a potential cause of '{event}'?"
                        is_person_potential = frame.simple_query(person_query, to_yesno=True)
                        action_query = f"Is action '{prev_action}' a potential cause of '{event}'?"
                        is_action_potential = frame.simple_query(action_query, to_yesno=True)
                        if "yes" in is_person_potential.lower() or "yes" in is_action_potential:
                            info["happened bad events"][f"{i} frame"]["potential cause"].append(pid)
                        break
    # After collecting information, use it to determine the presence of the trope
    answer, reason = video_segment.select_answer(info, query, possible_answers)
    return answer, reason, info
```

Listing B.2: ConQueR ICL Example

ConQueR enhances the model's ability to tackle Long-range Compositional Reasoning by decomposing the movie narrative (context) and the trope (query). In this instance, ConQueR systematically breaks down the identified characters and actions to align with the "Big Bad" trope query, as demonstrated in Lines 33, 36, 48, and 50.

doi:10.6342/NTU202401530

B.3 ABCD

We utilized all generated code from TiM and sampled 512 codes from NExT-QA [29], OKVQA [20], and GQA [10]. We constructed AST trees using the Python AST module and excluded codes that could not be parsed by AST (less than 3%) from our analysis. For VLM token analysis, we used NLTK's word_tokenize to split the VLM queries into tokens. The implementation details can be found in the repository.

B.4 Code Generation Prompt

We modified the prompting in Viper and FEVoRI so both can generate code for detecting tropes. Below, we present the prompt template. Notice that in B.3, we can replace **API_SPEC** and **EXAMPLES** according to the chosen ablation setting.

```
You are a professional programmer. You would be asked to follow the API specification and examples to complete the function.

You are only allowed to use imported package and defined class below to complete the function:

{API_SPEC}

The function parameter and return value should follow the signature below:

def execute_command(video, annotation, possible_answers, query)->[str, str, dict]
"""Returns (answer, reason, info) tuple when answering query with possible_answers
Parameters
----------
video (VideoSegment):
    Target video on which the command will be executed
possible_answers (list[str]):
    A list of possible answers that the command might return as answer
query (str):
    The question or query that needs to be answered using the video
Returns
-------
answer (str):
    The chosen answer from possible_answers list
reason (str):
    reason for why choosing the answer
info (dict):
    intermediate results collected from every api call
"""

Here's some function examples to refer:
```


doi:10.6342/NTU202401530

```
{EXAMPLES}


Please follow the format to return the function.


def execute_command(video, annotation, possible_answers, query)->[str, dict]
# Trope: FILL
# Definition: FILL
# Thought: FILL
# 1. FILL: FILL
# ...
# n. FILL: FILL
function logic implementation
return answer, reason, info
```

Listing B.3: Prompt template for Code generation

B.5 API Spec

Base setting In base setting, the api spec basically follow original setting in Viper. We design a new In-Context Learning(ICL) examples for codeLLM, in order to make code more suitable for detecting tropes

```
class ImagePatch:
    """A Python class containing a crop of an image centered around a particular object,
       as well as relevant information.
    Attributes
    ----------
    cropped_image : array_like
        An array-like of the cropped image taken from the original image.
    left, lower, right, upper : int
        An int describing the position of the (left/lower/right/upper) border of the crop's
        bounding box in the original image.

    Methods
    -------
    find(object_name: str)->List[ImagePatch]
        Returns a list of new ImagePatch objects containing crops of the image centered
        around any objects found in the image matching the object_name.
    exists(object_name: str)->bool
        Returns True if the object specified by object_name is found in the image, and
        False otherwise.
    best_text_match(option_list: List[str], prefix: str)->str
        Returns the string that best matches the image.
    simple_query(question: str=None, to_yesno: bool=False)->str
        Returns the answer to a basic question asked about the image. If no question is
        provided, returns the answer to "What is this?".
        If to_yesno is set to True, the answer must contain 'yes' or 'no'
    crop(left: int, lower: int, right: int, upper: int)->ImagePatch
        Returns a new ImagePatch object containing a crop of the image at the given coordinates.
    llm_query(question: str, to_yesno: bool=False)->str
        Returns the answer to a basic question which is unrelevant to the image.
        If to_yesno is set to True, the answer must contain 'yes' or 'no'
    """

class VideoSegment:
    """A Python class containing a set of frames represented as ImagePatch objects,
    as well as relevant information.
    Attributes
    ----------
    video : torch.Tensor
        A tensor of the original video.
    start : int
        An int describing the starting frame in this video segment with respect to the original video.
    end : int
        An int describing the ending frame in this video segment with respect to the original video.
    num_frames->int
        An int containing the number of frames in the video segment.

    Methods
    -------
    select_answer(self, info: dict, question: str, options: List[str]) -> (str, str):
```


doi:10.6342/NTU202401530

```
    Return (answer, reason) for the question and options according to given information
trim(start, end) -> VideoSegment
    Returns a new VideoSegment containing a trimmed version of the original video at the
    [start, end] segment.
frame_iterator() -> Iterator[ImagePatch]
    Returns an iterator over the frames in the video segment.
"""
```

Listing B.4: API spec of base setting

```
def execute_command(video, annotation, possible_answers, query)->[str, dict]:
    # Trope: Asshole Victim
    # Definition: A narrative trope where the victim of a crime or misdeed
    # is someone who had it coming because they were themselves morally dubious
    # or outright villainous.
    # Thought Process:
    # 1. Frame Selection: This trope involves identifying both the 'victim' and
    # the act leading to their victimhood, suggesting a need for comprehensive
    # analysis throughout the video.
    # 2. Character Analysis: Identify the 'victim' character and analyze their
    # actions or character traits that justify the trope's criteria.
    # 3. Incident Analysis: Look for an incident within the video that cements
    # the character's role as a victim.
    # 4. Morality Check: Determine if there's a narrative or visual cue indicating
    # the victim's negative moral standing.
    # 5. Answer Selection: Using the collected data, decide whether the "Asshole
    # Victim" trope is present.
    video_segment = VideoSegment(video, annotation)
    # Initialize a dictionary to store information collected during analysis
    info = {}
    for i, frame in enumerate(video_segment.frame_iterator()):
        # Assume function exists to identify characters and incidents
        if frame.exists("person"):
            incident_description = frame.simple_query("Describe the incident happened in the image.")
            info[f"Character trait in {i}th frame"] = []
            info[f"Morality check in {i}th frame"] = []
            for person in frame.find("person"):
                # Analyze the character's actions or traits
                person_trait = person.simple_query("What is the person doing? What are his/her traits?")
                morality_query = frame.simple_query("Does the he/she show negative moral traits?", to_yesno=True)
                # Store the collected information
                info[f"Character trait in {i}th frame"].append(person_trait)
                info[f"Morality check in {i}th frame"].append(morality_query)
            info[f"Incident description in {i}th frame"] = incident_description
    # After collecting information, use it to determine the presence of the trope
    answer, reason = video_segment.select_answer(info, query, possible_answers)
    return answer, reason, info
```

Listing B.5: ICL examples of base setting

Subtitle-Support Setting We extends the ability of Viper by adding implemented method in class ImagePatch as show in B.6. The new method is called get_subtitles(), which enable executed program to accessing subtitles. Additionally, we also design another version of ICL exampleB.7 based on B.5.

```
class ImagePatch:
    """A Python class containing a crop of an image centered around a particular object,
       as well as relevant information.
    Attributes
    ----------
    cropped_image : array_like
        An array-like of the cropped image taken from the original image.
    left, lower, right, upper : int
        An int describing the position of the (left/lower/right/upper) border of the crop's
        bounding box in the original image.

    Methods
    -------
    find(object_name: str)->List[ImagePatch]
        Returns a list of new ImagePatch objects containing crops of the image centered
        around any objects found in the image matching the object_name.
    exists(object_name: str)->bool
        Returns True if the object specified by object_name is found in the image, and
        False otherwise.
    get_subtitles() -> List[str]
        Return a list of str, which is subtitles that present in the current frame
    best_text_match(option_list: List[str], prefix: str)->str
        Returns the string that best matches the image.
    simple_query(question: str=None, to_yesno: bool=False)->str
```


doi:10.6342/NTU202401530

```
        Returns the answer to a basic question asked about the image. If no question is
        provided, returns the answer to "What is this?".
        If to_yesno is set to True, the answer must contain 'yes' or 'no'
    crop(left: int, lower: int, right: int, upper: int)->ImagePatch
        Returns a new ImagePatch object containing a crop of the image at the given coordinates.
    llm_query(question: str, to_yesno: bool=False)->str
        Returns the answer to a basic question which is unrelevant to the image.
        If to_yesno is set to True, the answer must contain 'yes' or 'no'
    """

class VideoSegment:
    """A Python class containing a set of frames represented as ImagePatch objects,
    as well as relevant information.
    Attributes
    ----------
    video : torch.Tensor
        A tensor of the original video.
    start : int
        An int describing the starting frame in this video segment with respect to the original video.
    end : int
        An int describing the ending frame in this video segment with respect to the original video.
    num_frames->int
        An int containing the number of frames in the video segment.

    Methods
    -------
    select_answer(self, info: dict, question: str, options: List[str]) -> (str, str):
        Return (answer, reason) for the question and options according to given information
    trim(start, end) -> VideoSegment
        Returns a new VideoSegment containing a trimmed version of the original video at the
        [start, end] segment.
    frame_iterator() -> Iterator[ImagePatch]
        Returns an iterator over the frames in the video segment.
    """
```

Listing B.6: API spec of subtitle-support setting

```
def execute_command(video, annotation, possible_answers, query)->[str, str, dict]:
    # Trope: Asshole Victim
    # Definition: A narrative trope where the victim of a crime or misdeed is someone
    # who had it coming because they were themselves morally dubious or outright villainous.
    # Thought Process:
    # 1. Frame Selection: This trope involves identifying both the 'victim' and the
    # act leading to their victimhood, suggesting a need for comprehensive analysis
    # throughout the video.
    # 2. Character Analysis: Identify the 'victim' character and analyze their actions or
    # character traits that justify the trope's criteria.
    # 3. Incident Analysis: Look for an incident within the video that cements the
    # character's role as a victim.
    # 4. Morality Check: Determine if there's a narrative or visual cue indicating the
    # victim's negative moral standing.
    # 5. Answer Selection: Using the collected data, decide whether the "Asshole Victim"
    # trope is present.
    video_segment = VideoSegment(video, annotation)
    # Initialize a dictionary to store information collected during analysis
    info = {}
    for i, frame in enumerate(video_segment.frame_iterator()):
        # the trope usually present with human character, thus detect person first
        if frame.exists("person"):
            # use ImagePatch.get_subtitles() to get dialogue, latter use the dialogue
            # with query as context information
            subtitles_info = "With subtitles '" + ' '.join(frame.get_subtitles()) + "'"
            incident_description = frame.simple_query(subtiltles_info + "Describe the incident happened in the image.")
            info[f"Character trait in {i}th frame"] = []
            info[f"Morality check in {i}th frame"] = []
            for person in frame.find("person"):
                # Analyze the character's actions or traits
                trait_query = subtitles_info + "What is the person doing? What are his/her traits?"
                person_trait = person.simple_query(trait_query)
                morality_query = subtitles_info + "Does the he/she show negative moral traits?", to_yesno=True
                morality_query = person.simple_query(morality_query)
                # Store the collected information
                info[f"Character trait in {i}th frame"].append(person_trait)
                info[f"Morality check in {i}th frame"].append(morality_query)
            info[f"Incident description in {i}th frame"] = incident_description
    # After collecting information, use it to determine the presence of the trope
    answer, reason = video_segment.select_answer(info, query, possible_answers)
    return answer, reason, info
```

Listing B.7: ICL examples of subtitle-support setting

Face Identification Setting To make our proposed method, FEVoRI, has ability to identify character. We introduce a new method, which is called *face_identify()*, for class

doi:10.6342/NTU202401530

VideoSegmentB.8 to verify face through the video. We also design new ICL promptB.9 based on B.5

```
class ImagePatch:
    """A Python class containing a crop of an image centered around a particular
       object, as well as relevant information.
    Attributes
    ----------
    cropped_image : array_like
        An array-like of the cropped image taken from the original image.
    left, lower, right, upper : int
        An int describing the position of the (left/lower/right/upper) border
        of the crop's bounding box in the original image.

    Methods
    -------
    find(object_name: str)->List[ImagePatch]
        Returns a list of new ImagePatch objects containing crops of the image
        centered around any objects found in the image matching the object_name.
    exists(object_name: str)->bool
        Returns True if the object specified by object_name is found in the
        image, and False otherwise.
    best_text_match(option_list: List[str], prefix: str)->str
        Returns the string that best matches the image.
    simple_query(question: str=None, to_yesno: bool=False)->str
        Returns the answer to a basic question asked about the image. If no question
        is provided, returns the answer to "What is this?".
        If to_yesno is set to True, the answer must contain 'yes' or 'no'
    crop(left: int, lower: int, right: int, upper: int)->ImagePatch
        Returns a new ImagePatch object containing a crop of the image at the given coordinates.
    llm_query(question: str, to_yesno: bool=False)->str
        Returns the answer to a basic question which is unrelevant to the image.
        If to_yesno is set to True, the answer must contain 'yes' or 'no'
    """

class VideoSegment:
    """A Python class containing a set of frames represented as ImagePatch objects,
       as well as relevant information.
    Attributes
    ----------
    video : torch.Tensor
        A tensor of the original video.
    start : int
        An int describing the starting frame in this video segment with respect
        to the original video.
    end : int
        An int describing the ending frame in this video segment with respect to
        the original video.
    num_frames->int
        An int containing the number of frames in the video segment.

    Methods
    -------
    face_identify(image: ImagePatch) -> str
        Return an unique identifier according to person in image
    select_answer(self, info: dict, question: str, options: List[str]) -> (str, str):
        Return (answer, reason) for the question and options according to given information
    trim(start, end) -> VideoSegment
        Returns a new VideoSegment containing a trimmed version of the original
        video at the [start, end] segment.
    frame_iterator() -> Iterator[ImagePatch]
        Returns an iterator over the frames in the video segment.
    """
```

Listing B.8: API spec of face identification setting

```
def execute_command(video, annotation, possible_answers, query)->[str, str, dict]:
    # Trope: Asshole Victim
    # Definition: A narrative trope where the victim of a crime or misdeed is
    # someone who had it coming because they were themselves morally dubious or outright villainous.
    # Thought Process:
    # 1. Frame Selection: This trope involves identifying both the 'victim' and the act
    # leading to their victimhood, suggesting a need for comprehensive analysis throughout the video.
    # 2. Character Analysis: Identify each character and collect their actions or character traits
    # 3. Answer Selection: Using the collected data, decide whether the "Asshole Victim" trope is present.
    video_segment = VideoSegment(video, annotation)
    # Initialize a dictionary to store information collected during analysis
    info = {
        "captions": {}
        "character_behaviors": {}
    }
    for i, frame in enumerate(video_segment.frame_iterator()):
        # collect background story from caption of frame
        caption = frame.simple_query("What's happening in the scene?")
        info["captions"][f"{i} frame"] = caption
        # identify person in frame
        for person in frame.find("person"):
            person_id = video_segment.face_identify(person)
            if person_id is None:
                continue
            # get description of person
            person_description = person.simple_query("What's his/her appearance characteristic? Describe in 10 words")
            # track character behavior
            person_behavior_in_frame = frame.simple_query(f"What's action of person with appearance '{person_description}'")
            if person_id not in info["character_behaviors"]:
```


doi:10.6342/NTU202401530

```
            info["character_behaviors"][person_id] = {}
        info["character_behaviors"][person_id].update({
            f"action in {i} frame": person_behavior_in_frame
        })
# After collecting information, use it to determine the presence of the trope
answer, reason = video_segment.select_answer(info, query, possible_answers)
return answer, reason, info
```

Listing B.9: ICL examples of face identification setting

doi:10.6342/NTU202401530