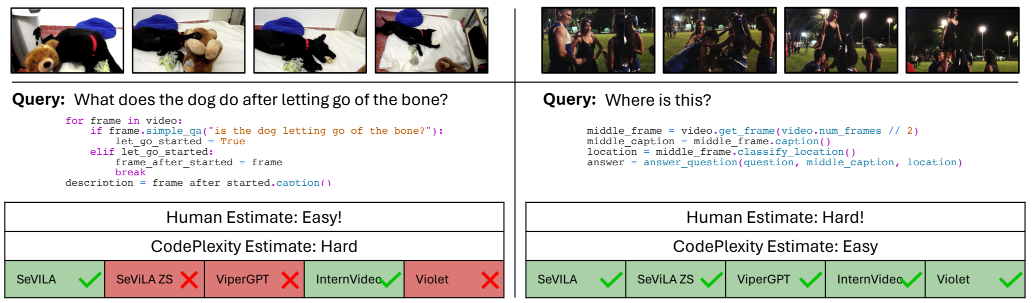
Understanding Complexity in VideoQA via Visual Program Generation

Anonymous authors
Paper under double-blind review

Abstract

We propose a data-driven approach to analyzing query complexity in Video Question Answering (VideoQA). Previous efforts in benchmark design have largely relied on human expertise to construct challenging samples. In this work, we experimentally demonstrate that humans struggle to accurately estimate which questions are hard to answer for machine learning models. Our alternative, automated approach takes advantage of recent advances in code generation for visual question answering. In particular, we use the complexity of generated code as a proxy for the question complexity and demonstrate that it indeed shows a much stronger correlation with the models' performance, compared to human estimates. We then present a novel algorithm for estimating question complexity from code. It identifies fine-grained primitives that correlate with the hardest questions. These human-interpretable results lead to a number of discoveries about the key sources of complexity for VideoQA models. Finally, we extend our approach to generate complex questions for a given set of videos. This allows us to automatically construct a new benchmark, which is 1.9 times harder for VideoQA methods than existing manually designed datasets.



```
for frame in video:
    if frame.simple_qa("is the dog letting go of the bone?"):
        let_go_started = True
    elif let_go_started:
        frame_after_started = frame
        break
description = frame_after_started.caption()
```

```
middle_frame = video.get_frame(video.num_frames // 2)
middle_caption = middle_frame.caption()
location = middle_frame.classify_location()
answer = answer_question(question, middle_caption, location)
```

Figure 1: Humans struggle to judge which questions present higher challenges for machine learning models. In our study, the question on the left is universally perceived as being easier than the one on the right, which is inversely correlated with the models' performance. We show that the complexity of the corresponding visual program can serve as a much more reliable predictor.

1 Introduction

Humans can effortlessly reason about activities, whether that reasoning requires understanding space and time, cause and effect, or fine-grained details and high-level context (Decety & Grèzes, 1999; Decety et al., 1997; Wurm & Caramazza, 2022; Aflalo et al., 2020). This versatility allows us to function effectively in dynamic environments, yet it simultaneously complicates our ability to assess what is hard for machines. Consider the two video-question pairs shown in Figure 1 (top). In our study, human subjects overwhelmingly perceive the question on the right as the more complex to answer, but evaluating a variety of state-of-the-art VideoQA models (Yu et al., 2023a; Wang et al., 2022a; Surís et al., 2023; Fu et al., 2021) shows that the question on the left presents a significantly greater challenge for them.

While an expert in the field might think that they would do a better job at this task, the history of VideoQA benchmarks has proven otherwise. Several studies have shown that, despite the best efforts of their authors, most datasets are dominated by questions that can be mastered even by naive, single-frame baselines (Buch et al., 2022; Huang et al., 2018; Liu et al., 2021). Although many attempts have been made to address this limitation, they predominantly adopt a top-down approach. In particular, these works start from an expert hypothesis of what is hard and validate this assumption by evaluating models on samples that specifically target the identified skill (Xiao et al., 2021; Mangalam et al., 2023). While this has led to some progress, top performance can still often be achieved by methods that rely almost exclusively on static image cues (Yu et al., 2023a).

In this work, we propose a bottom-up approach instead that discovers human-interpretable insights about the sources of complexity for existing VideoQA models from the data. To this end, we capitalize on a recent large language model (LLM)-based code generation paradigm (Surís et al., 2023; Gupta & Kembhavi, 2023; Subramanian et al., 2023), which produces modular executable programs to answer natural language queries. While this approach has shown promise for zero-shot VideoQA (Surís et al., 2023), we are not interested in its task performance per se. Instead, we propose to use its rich and highly structured intermediate representations—programs, as shown in Figure 1 (bottom)—to capture the elusive complexity of the original questions. It is important to note that the videos themselves also contribute to the complexity of the VideoQA samples. However, we demonstrate that our approach allows us to construct an efficient complexity metric with high predictive power from the questions alone.

To this end, we begin by collecting the visual programs generated by Surís et al. (2023) in their recent approach on the validation set of the challenging NextQA benchmark (Xiao et al., 2021), together with predictions of a large collection of diverse VideoQA algorithms. We then calculate several standard structural complexity metrics (McCabe, 1976) for these programs and additionally collect human judgments about the question complexity for a subset of the dataset. Intriguingly, our analysis demonstrates that, despite the programs being imperfect, even the simplest code complexity metrics correlate better with machine learning models' performance on NextQA than human estimates (see Figure 3).

We then propose CodePlexity – a novel algorithm for estimating question complexity from code that takes into account the content of the program, in addition to its structure (Section 3.2). In particular, it correlates individual subroutines with model performances, effectively mining for human-interpretable patterns that summarize the error modes of each model. In addition to aggregating the subroutines mined from multiple models into a single, robust quantitative metric, our approach allows one to study them individually. In Section 3.3, we propose an algorithm to identify patterns in the code that universally correlate with challenging questions, leading to a number of important discoveries. For example, we find that reasoning about the order of events and about fine-grained object details is difficult for all state-of-the-art models tested.

Finally, equipped with this powerful analysis tool, we design an algorithm for automatically generating challenging questions for any given collection of videos in Section 3.4. In particular, our approach takes as input a compact description of a video and uses an LLM like ChatGPT (OpenAI, 2023a) to generate question candidates first. We then generate visual programs for each question and use our code-based complexity metric to select the hardest subset. We evaluate several zero-shot VideoQA methods on the resulting benchmark and observe a 1.9× gap in performance compared to existing datasets like NextQA (Xiao et al., 2021), confirming the effectiveness of our approach.

To summarize, our contributions are as follows:

1. We demonstrate that generated code complexity can serve as a robust metric of question complexity in VideoQA and propose a novel approach for automatically quantifying it.
2. We present CodePlexity, a novel approach that identifies the key sources of complexity for existing VideoQA models. For example, we discover that most models fail when the order of the frames has to be taken into account.
3. Using CodePlexity, we automatically construct CodePlex-QA– a novel benchmark that is 1.9 times harder for VideoQA methods than existing, manually designed datasets.

2 Related Work

We review the relevant literature in video question-answering models, frame selection and code generation for QA, and measures of code complexity.

Large single-stage models. A number of recent works propose to train end-to-end architectures for video-language understanding. Zellers et al. (2021) propose Merlot, a large video-language dataset obtained through ASR captions to train a model with contrastive frame-caption matching, masked language modeling, and re-ordering of scrambled video frames. VIOLET (Fu et al., 2021) is an end-to-end joint video and text architecture that uses the dVAE (Van Den Oord et al., 2017) from DALL-E (Ramesh et al., 2021) to generate tokens for masked video-text pre-training and is tested on video question-answering and text-to-video retrieval. A number of works train masked space-time autoencoders (TimesFormer (Bertasius et al., 2021), VideoMAE (Feichtenhofer et al., 2022)) on video sequences, testing on tasks like action recognition. mPLUG-2 (Xu et al., 2023) proposes to unify image- and video-language tasks in a single architecture with task-specific modules pre-trained with masked language modeling, vision-language contrastive learning, and task instruction representation.

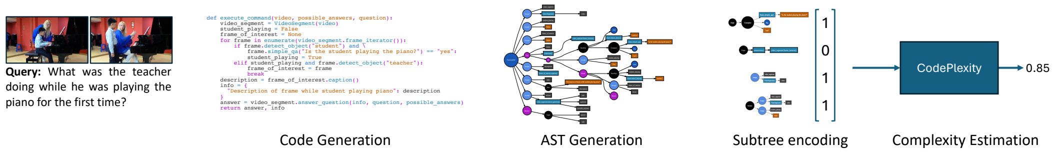
Frame selection models. Other work leverages single-image vision-language models for video understanding, on the hypothesis that only a subset of video frames (often just a single frame) are relevant to answer a given query. Atemporal Probe (ATP) (Buch et al., 2022) proposes a frame selector to measure the extent to which single-frame image-centric baselines can address Video Question Answering. Later, SeViLA (Yu et al., 2023a) builds on this paradigm by fine-tuning BLIP2 twice: first to localize relevant keyframes, and then for the question answering module, which answers the question based on the selected frames, question, and candidate answers.

Code generation models. A number of recent works tackle the frame selection and question answering problem through *code generation*, leveraging the success of recent text-to-code models such as Codex (Chen et al., 2021). VisProg (Gupta & Kembhavi, 2023) decomposes natural language queries into compositional programs, using a variety of zero-shot pretrained models to solve visual question answering problems. ViperGPT (Surís et al., 2023) prompts code generation models with an API that incorporates a variety of vision modules, generating Python code that is executed to answer natural language queries without access to source images or video frames. The approach does not require further training and has demonstrated state-of-the-art results across various complex visual tasks. CodeVQA (Subramanian et al., 2023) is a concurrent work to ViperGPT with a smaller API, specialized to single-frame QA. Recursive Visual Programming or RVP Ge et al. (2023) employs a recursive code generation strategy, which systematically breaks down complex problems into manageable subproblems. This allows it to handle intricate question structures with greater flexibility. Precursors to these models (Andreas et al., 2016; Johnson et al., 2017b; Kim et al., 2018; Hu et al., 2017; Yi et al., 2018) would implement code generation modules using neural networks which were trained either with supervision or via reinforcement learning.

Complexity estimation. Prior work has proposed ways to estimate complexity for other settings and modalities. For instance, in NLP, sentence or paragraph length has been used as a proxy for the complexity of a text (Platanios et al., 2019; Spitkovsky et al., 2010; Tay et al., 2019), where a sample with more words is presumed to correlate with harder training data. Related works have explored variations of this metric such as number of conjunctions (Kocmi & Bojar, 2017) (and/or), number of phrases (Tsvetkov et al., 2016a), or depth of the dependency tree (Tsvetkov et al., 2016b). In computer vision, prior work by Wei et al. (2016) has approximated image complexity with the number of objects present in an image, while others have directly collected human annotations for image complexity (Tudor Ionescu et al., 2016; Soviany et al., 2020). Finally, a seminal paper by Graves (2016) suggested that, in reasoning tasks, the complexity of said task can be estimated by the number of reasoning steps required to solve it, and introduced a method to estimate said number. A follow-up work by Eyzaguirre & Soto (2020) refined and advanced the initial method, and used it to quantify the complexity of VQA (Visual Question Answering) queries.

Problem complexity from code. Measuring complexity through code has a rich history; Kolmogorov defines complexity based on the succinctness of the program that can represent said object (Kolmogorov, 1963; Solomonoff, 2009). However, its incomputability limits its practical application (Zvonkin & Levin, 1970). Software engineering rely on tangible metrics like cyclomatic complexity that measure the number of independent paths in a program (McCabe, 1976), a computable yet less philosophically rich approach.

Related to our focus on Video Question Answering, synthetic datasets have become crucial in computer vision, particularly for analysis on question answering tasks. These datasets often include symbolic programs that abstract the task of low-level perception into modular operations, effec-



```
def execute_command(video, possible_answers, question):
    video_segment = VideoSegment(video)
    student_playing = False
    frame_of_interest = None
    for frame in enumerate(video_segment.frame_iterator()):
        if frame.detect_object("student") and \
            frame.simple_qa("Is the student playing the piano?") == "yes":
            student_playing = True
        elif student_playing and frame.detect_object("teacher"):
            frame_of_interest = frame
            break
    description = frame_of_interest.caption()
    info = {
      "Description of frame while student playing piano": description
    }
    answer = video_segment.answer_question(info, question, possible_answers)
    return answer, info
```

Figure 2: **Estimating question complexity via code.** Our approach to estimating question complexity involves converting the question into code, decomposing the pseudo-code into abstract syntax subtrees (S_i), before finally correlating subtree presence with model performance.

tively separating the perceptual components from the higher-level reasoning (Johnson et al., 2017a; Grunde-McLaughlin et al., 2021; Yu et al., 2023b; Wu et al., 2021). These programs can then be grouped into families that necessitate similar skills, thank to which it becomes possible to correlate model performance with program structure, providing insights into the model's reasoning capabilities. Contrasting this, our work pioneers the use of code generation models to estimate question complexity, without needing expensive annotations and generalizing to more question types. Our novel approach offers a direct, computable measure of question complexity, bridging the gap between theoretical definitions of complexity, and practical applications in machine learning.

3 METHODOLOGY

3.1 PRELIMINARIES

We study the problem of estimating the complexity of questions in VideoQA. We are given a dataset consisting of collections of videos, questions, and answers $\mathcal{D} = \{\mathbf{V}, \mathbf{Q}, \mathbf{A}\}$, along with a set of K models already trained on the task $\mathcal{M} = \{m_0, ..., m_K\}$. Our goal then is to design a function $\mathcal{C}$ that allows us to categorize questions $q_i \in \mathbf{Q}$ into groups based on their complexity with respect to $\mathcal{M}$. Crucially, we are interested in a general metric consistent across all models $m_j \in \mathcal{M}$. Concretely, for any two questions $q_1, q_2 \in \mathbf{Q}$, together with corresponding videos $v_1, v_2 \in \mathbf{V}$, if $\mathcal{C}(q_1) > \mathcal{C}(q_2)$, we expect model performance $P(m, q, v)$ to vary accordingly: $P(m_j, q_1, v_1) < P(m_j, q_2, v_2)\ \forall m_j \in \mathcal{M}$, indicating that models perform worse on more complex questions.

However, directly estimating complexity $\mathcal{C}$ from natural language question q is a challenging problem even for humans, as we demonstrate in a Section 4.2. Instead, our key idea, inspired by the notion of Kolmogorov Complexity ($\mathcal{KC}$) (Kolmogorov, 1963), is to utilize the rich and highly-structured intermediate representations - programs, to capture the elusive complexity of the original natural language queries. Concretely, we capitalize on the recent code generation-based methods (Surís et al., 2023; Gupta & Kembhavi, 2023; Subramanian et al., 2023) that operate in a 2-stage fashion: first, given a question q a program generator π from a Large Language Model (LLM) is used to translate it into an executable program $z = \pi(q)$. An off-the-shelf execution engine like Python can then be used to produce an answer $\hat{a} = \phi(v, z)$. Running such an approach on a dataset $\mathcal{D}$ results in a set of programs $\mathcal{P}(\mathcal{D}) = \{z_1, z_2, ..., z_N\}$.

Next, in Section 3.2 we propose several techniques for code analysis of increasing intricacy and show how they can be used to build a function for question complexity estimation via code generation $\mathcal{C}(q) \propto \mathcal{C}(z)$. Then, in Section 3.3, we demonstrate how analysis of the generated code can help gain insights into the failure modes of VideoQA models. Finally, in Section 3.4 we discuss how such algorithms can be used to automatically construct challenging VideoQA benchmarks.

3.2 CODEPLEXITY: ESTIMATING QUESTION COMPLEXITY FROM CODE

As a first step we review existing software engineering metrics that map code into complexity scores $\mathcal{C}(z) \to \mathbb{R}$. In particular, we focus on Lines of Code (LoC) and Cyclomatic Complexity (McCabe, 1976). The former simply correlates the number of lines in a program with its complexity $\mathcal{C}(z) \propto |z|$, whereas the latter quantifies the number of linearly-independent paths through the source code and is denoted as $\mathcal{C}(z) = CC(z)$. To minimize the impact of spurious factors, we pre-process the code by removing all the comments and empty lines first, and make sure to use the same set of basic primitives in all experiments. Both metrics are indicative of the code's structural complexity, with higher values suggesting more intricate control flow. However, they do not take the contents of the code into consideration, which, as we shown in Section 4.2, limits their predictive power.

To address this drawback, we propose a new method, CodePlexity, illustrated in Figure 2. CodePlexity involves analyzing the components of the generated code that affect question complexity, considering both its structure and semantic content. More specifically, we develop a compiler to parse each question's code into its basic syntactic elements, creating a Abstract Syntax Tree or AST (Hoe et al., 1986) $T = compile(z)$ with nodes N and edges E. In this model, nodes represent variables, functions, and control structures, while the edges capture the logical and hierarchical relationships between them. The AST framework abstracts the code away from its literal syntax, allowing us to focus on the underlying logic and structure. By generating ASTs for the entire dataset, we obtain a comprehensive set $\mathcal{T}(\mathcal{D}) = \{compile(z) \mid z \in \mathcal{P}(\mathcal{D})\}$, thus laying the groundwork for a deeper analysis of code complexity factors.

Next, we mine $\mathcal{T}$ for common subroutines (recurring logical patterns or functions) that occur in the code. In ASTs subroutines manifest as subtrees, which we denote as $S = (N', E')$, where $N' \subseteq N$, $E' \subseteq E$ and $\forall (u, v) \in E',\ u \in N' \wedge v \in N'$. Importantly, not all subtrees constitute valid Python code, since they might fail to comply with Python's syntax rules. To systematically identify valid subtrees, we define a function $\mathcal{G}(T)$ that yields an unordered set of all valid subtrees of T, denoted as $\mathcal{G}(T) = \{S_1, S_2, \ldots, S_n\}$. Considering the entire dataset, the collection of all valid subtrees across the dataset can be represented as $\mathcal{S}(\mathcal{D}) = \bigcup_{T \in \mathcal{T}(\mathcal{D})} \mathcal{G}(T)$.

Then, to avoid duplicates, we merge subtrees that always co-occur when one is a descendant of the other. Specifically, $\mathcal{S}_{merged}(\mathcal{D})$ is defined as the subset of $\mathcal{S}(\mathcal{D})$ that excludes S_2 if there exists a subtree S_1 such that S_1 and S_2 always co-occur and S_2 is contained in S_1 (see Section 7.1 for formal definition). The presence of a specific subroutine within a program's AST can be verified via a subgraph isomorphism check:

$$ISO(T, S) \equiv S \in \mathcal{G}(T). \tag{1}$$

To aggregate the identified subtrees into a quantitative metric of complexity, we assign each subtree in $\mathcal{S}_{merged}(\mathcal{D})$ an index and encode each question q_i in the dataset using one-hot encoding $\mathbf{x}_i \in \mathbb{R}^{|\mathcal{S}_{merged}(\mathcal{D})|}$, where a 1 in index k of x_i signifies the presence of subtree S_k in question's AST T_i.

$$x_{ik} = \begin{cases} 1 & \text{if } ISO(T_i, S_k) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

This representation transforms the complex structure of code into a fixed-size vector, enabling straightforward application of machine learning models. We then employ a logistic regression model trained on these one-hot encodings to predict the success of models $m_j \in \mathcal{M}$. Note that the training set effectively treats each $(\mathbf{x}_i, y_i^{(j)})$ pair as a distinct instance, where $y_i^{(j)}$ is the binary outcome for question i with respect to model m_j (1 for success, 0 for failure). This approach is justified by our objective to identify subtrees that universally challenge the models, implying a structural complexity in the code that transcends specific models. We then obtain the final complexity function via:

$$\text{CodePlexity}(z) = -\hat{y}_i = -\sigma(\mathbf{w}\mathbf{x}_i + b). \tag{3}$$

Next, we discuss how our subtree analysis approach allows to obtain deeper insights into the sources of complexity for existing VideoQA models.

3.3 Subtree Analysis

Unlike black-box metrics, in addition to a numerical score, our approach also outputs an interpretable set of subtrees that correlate with challenging questions. We now demonstrate how to identify subroutines that have a high impact on model performance. More specifically, we are interested in subtrees that are linked to a decrease in model m_j's performance with a high degree of statistical significance (set at 0.99). To test this, we establish a null hypothesis ($H0$) stating that the proportion of successes is the same with and without the subtree present:

$$H0 : P(m_j, q_1 | S \in \mathcal{S}(\mathcal{D})) = P(m_j, q_1 | S \notin \mathcal{S}(\mathcal{D})). \tag{4}$$

Conversely, our alternative hypothesis posits that the proportion of successes without the subtree is greater, implying that its presence hurts performance:

$$HA : P(m_j, q_1 | S \in \mathcal{S}(\mathcal{D})) < P(m_j, q_1 | S \notin \mathcal{S}(\mathcal{D})). \tag{5}$$

We conduct a one-sided test to evaluate these hypotheses and define a subset of subtrees, denoted as $\mathcal{S}^*_{m_j}(\mathcal{D})$, for which their presence is statistically correlated with a decrease in the performance of the model m_j:

$$\mathcal{S}^*_{m_j}(\mathcal{D}) = \{S \in \mathcal{S}(\mathcal{D}) \,|\, p(S, m_j) < 0.01\}\,, \tag{6}$$

where $p(S, m_j)$ denotes the corresponding p-value. Finally, to identify the subroutines that are associated with performance decrease for multiple models, we consider the intersection of the sets:

$$\mathcal{S}^* = \bigcap_{m_j \in \mathcal{M}} \mathcal{S}^*_{m_j}(\mathcal{D}). \tag{7}$$

Identifying the specific subtrees that cause a decrease in models' performance allows us to obtain deeper insights into where and how they may falter. In Section 4.3, we perform this analysis for several state-of-the-art approaches and suggest potential areas for improvement in model design.

3.4 Learning to Ask Hard Questions

We now build on our code-based question complexity metric described in Section 3.2, and propose a method for automatically generating challenging question-answer pairs for any given set of videos. Concretely, our approach takes as input a set of videos $\mathbf{V}$ paired with natural language summaries $\mathbf{C}$. We then follow prior work by Mangalam et al. (2023) and prompt a large language model (LLM) to generate question and answer candidates based on each summary individually $\tilde{q}, \tilde{a} = LLM(c, prompt)$. The exact prompts are listed in Section 7.4 of the appendix.

Importantly, our approach is agnostic to the nature of $\mathbf{C}$, which can either be annotated manually, or generated automatically. In this work, we take the latter approach and capitalize on existing datasets with scene graph annotations (Luo et al., 2021; 2022; Zhou et al., 2019; Ji et al., 2020) paired with an image captioning model to generate natural language summaries of the video such that a language model can understand them (Menon & Vondrick, 2022; Wang et al., 2022b; Zeng et al., 2023). We detail this algorithm in Section 8 of the appendix.

Following our approach from Section 3.2, we then convert each generated question $\tilde{q}$ into code, and use our trained CodePlexity model (Equation 3) to estimate its complexity. A set of candidate questions $\tilde{\mathbf{Q}}^*$ can be selected by setting a threshold δ for minimum complexity:

$$\tilde{\mathbf{Q}}^* = \{\tilde{q} \in \tilde{\mathbf{Q}} | \mathcal{C}(\tilde{q}) \geq \delta\}. \tag{8}$$

Finally, we manually filter the candidate dataset $\tilde{\mathcal{D}}^* = \{\mathbf{V}, \tilde{\mathbf{Q}}^*, \tilde{\mathbf{A}}^*\}$ to remove the question/answer pairs that cannot be accurately answered from the corresponding videos due to inaccuracies in the generated summaries or LLM hallucination. We emphasize that this final manual filtering is only needed to ensure the perfect quality of the final dataset $\mathcal{D}^*$. In practice, we only had to remove 12% of the questions, demonstrating that the fully automatic pipeline is capable of producing useful datasets by itself.

4 Evaluating Question Complexity Estimation

In this section, we compare different approaches to estimating question complexity in VideoQA. To this end, we first define a thorough evaluation protocol and detail our experimental setup in Section 4.1. We then evaluate how the code-based metrics proposed in this work compare to human subjects and several simple baselines in predicting the performance of a wide variety of contemporary approaches on the popular NextQA benchmark in Section 4.2. We conclude by performing a detailed analysis of the subroutines that show the strongest correlations with challenging questions in Section 4.3.

4.1 Experimental Setup

Evaluation protocol. Our goal is to compare the predictive power of several approaches for estimating question complexity in VideoQA with respect to a variety of machine learning models $\mathcal{M}$ on a dataset $\mathcal{D}$. Importantly, some of the metrics we study require training data in the form of questions paired with outcomes of a model $m_j \in \mathcal{M}$ on them $(q_i, y_i^{(j)})$. Thus we split the whole pool of models $\mathcal{M}$ into the training $\mathcal{M}_{tr}$ and held-out validation $\mathcal{M}_{val}$ sets and report results on the latter.

To quantitatively compare the effectiveness of different approaches, some of which map a question to a numerical value corresponding to its complexity, whereas others directly return an ordering of the questions, we propose a unifying metric, Performance Extremity Gap (PEG). In particular, we first use numerical complexity estimates to sort questions accordingly. We then measure the disparity in model m_j's performance P between the easiest and the hardest α% of the questions via:

$$\text{PEG}(m_j, \alpha) = \frac{1}{N_\alpha} \sum_{q \in Q_{\text{hardest},\alpha}} P(m_j, q, v_q) - \frac{1}{N_\alpha} \sum_{q \in Q_{\text{easiest},\alpha}} P(m_j, q, v_q) \tag{9}$$

Finally, inspired by the mAP metric (Everingham et al., 2010), we average the PEG values over $\alpha \in (0, 0.5]$ to obtain the final mPEG score.

Dataset. It is crucial that the dataset used to perform our analysis features as many diverse challenges as possible. We choose the NExT-QA (Xiao et al., 2021) benchmark for its size, variety of human-annotated questions, and its focus on spatio-temporal reasoning in videos over mere visual-fact retrieval (Zhong et al., 2022). In addition, its popularity provides a large pool of models with pre-trained, public checkpoints for our study. That said, our method is dataset-agnostic and can be applied to any other VideoQA benchmark in the future. We perform the evaluation on the validation set of NExT-QA, further splitting the questions into 80% used to train the metrics and the other 20% held out for computing mPEG.

Models. We include seven representative methods, chosen for their coverage of existing architectural philosophies, pre-training strategies, and state-of-the-art performance. In particular, we use VIOLET (Fu et al., 2021) and InternVideo (Wang et al., 2022a), which are pre-trained with contrastive visual-language objectives and fine-tuned for VideoQA. We also evaluate SeViLA (Yu et al., 2023a), which is based on the BLIP-2 (Li et al., 2023) large-scale visual-language model; we assess both its zero-shot variant (SeViLA-ZS) and a fine-tuned version (SeViLA). Additionally, we evaluate HGA (Jiang & Han, 2020), a GNN-based model that reasons with heterogeneous graph alignment, representing earlier approaches prior to the prevalence of video large language models (videoLLMs). Furthermore, we include the ViperGPT (Surís et al., 2023) code-generation-based approach, as well the simple but effective ATP baseline (Buch et al., 2022)Finally, we evaluate the current state-of-the-art model, Tarsier (Wang et al., 2024). The models are split into training and validation sets as follows: $\mathcal{M}_{tr} = \{\text{VIOLET}, \text{SeViLA}, \text{ViperGPT}, \text{ATP}\}$, $\mathcal{M}_{val} = \{\text{HGA}, \text{SeViLA-ZS}, \text{InternVideo}, \text{Tarsier}\}$.

Baselines. In addition to the code-based metrics introduced in Section 3.2, we evaluate several baselines that attempt to directly estimate question complexity from the natural language query itself. In particular, as a learning-free baseline, we follow Tsvetkov et al. (2016b) and correlate the complexity of a question with the maximum depth of its parsed dependency tree. To more fairly compare to our learnable, code-based metric we fine-tune BERT (Devlin et al., 2018) to predict the probability of model success given the question using exactly the same training data. We also prompt GPT-4 (OpenAI, 2023b) to estimate the complexity of a question on a Likert scale (Likert, 1932) (details and prompts are provided in Section 7.3 of the appendix).

Finally, we conduct a human study on a subset of 150 questions. To this end, we recruited 30 human subjects via the Prolific platform (Palan & Schitter, 2018). The subjects were asked to sort three questions at a time according to their perceived relative complexity. Consistency was validated by asking to compare the same questions multiple times throughout one session and the subjects who demonstrated low consistency were excluded from the study. The final sequence order of the entire subset was calculated via pairwise ELO scores (Elo, 1967). More details and an example of the annotation interface are provided in the appendix (Section 7.2).

4.2 Results

We begin by visualizing the correlation of human estimates of question complexity with the performance of all 6 models used in our study on the 150 manually annotated questions from NExT-QA in Figure 3 (left). We observe that, while a downward trend clearly exists, with the questions labeled as the hardest by humans resulting in lower success rate for models compared to the easiest ones, the correlation is very weak. Notably, the questions that are ranked as being average in complexity are in fact the hardest for the models.

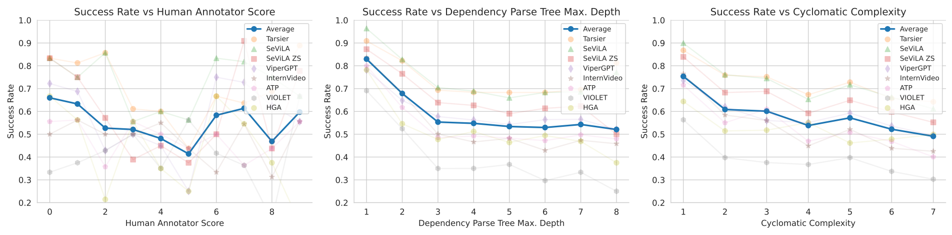


Figure 3: Correlation of various approaches for estimating question complexity with VideoQA models' success rate on these questions. We observe that humans struggle to accurately predict what's hard for machine learning models and that code can serve as a more reliable source of prediction than natural language questions.

	Train Models				Val. Models			
	SeViLA	ViperGPT	ATP	VIOLET	HGA	SeViLA ZS	InternVideo	Tarsier
Dependency Tree Depth	12.9	7.9	11.1	15.9	7.4	13.5	17.7	10.1
GPT-4 (OpenAI, 2023b)	9.6	8.9	11.6	5.8	7.8	14.6	13.9	10.8
BERT (Devlin et al., 2018)	12.5	6.0	18.3	17.3	7.7	14.3	21.1	10.8
Lines of Code	16.4	15.3	14.2	12.0	9.9	16.2	17.5	14.4
Cyclomatic Complexity	18.2	14.2	18.7	15.9	8.9	17.2	24.2	16.7
CodePlexity (Ours)	26.7	21.3	21.0	15.8	**14.1**	**25.6**	**26.6**	**24.9**

Table 1: Comparison of question complexity metrics using mPEG on the validation set of NExT-QA. BERT and CodePlexity are trained on the first four models. Text-based metrics (above) perform worse than the code-based ones (below), and our approach demonstrates the highest correlation with the models' performance.

We then evaluate two baselines on the same set of questions, one based on the natural language queries (dependency tree depth shown in Figure 3, center) and one based on the generated code (cyclomatic complexity, Figure 3, right). Both show a much stronger correlation with the models' performance, with cyclomatic complexity being the most consistent. These results demonstrate that human intuition about sources of complexity in VideoQA does not reflect the main challenges for machine learning models, and that generated code can be a more reliable source for estimating complexity than natural language.

Next, we report a more systematic comparison of different text- and code-based metrics using mPEG on the validation set of NExT-QA in Table 1. Comparing the three language-based metrics in the upper part of the table on the held-out models, we find them to perform relatively similarly. Notably, the BERT-based model which is trained on the questions and prediction outcomes of the four models, performs better than the learning-free baselines for InternVideo, but fails to generalize to SeViLA. This demonstrates that the space of the natural language is not structured enough to fit a robust complexity estimation model.

In contrast, code-based metrics, shown in the lower part of Table 1 demonstrate better predictive ability overall, with even the simplest Lines of Code baseline outperforming the text-based metrics in most scenarios. Cyclomatic Complexity shows top results among all non-learning-based metrics, and our proposed approach, CodePlexity, achieves significant improvements over it by learning to identify code primitives which correlate with challenging questions. This brings us to the final aspect of our analysis: understanding these structural elements of the code that contribute to question complexity.

4.3 Subtree Analysis

We follow the approach proposed in Section 3.3 and identify the subtrees which are statistically correlated with a decrease in the performance $\mathcal{S}^*_{m_j}$ for three models out of out training set $\mathcal{M}_{tr}$: SeViLA, ViperGPT, and VIOLET. In Figure 4 (right) we visualize the intersections between these three individual sets $\mathcal{S}^*$ (Equation 7) as a Venn diagram. A perceptible common trend is apparent: different architectures have their own weaknesses, but the commonalities are surprisingly frequent.

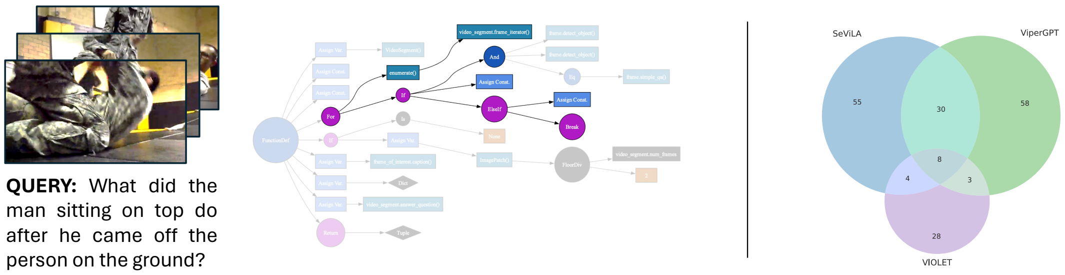

QUERY: What did the man sitting on top do after he came off the person on the ground?
video_segment.frame_iterator()
enumerate()
And
Assign Const
If
For
Elseif
Break
SeViLA
ViperGPT
VIOLET
55
30
58
8
4
3
28

Figure 4: Detailed analysis of subtrees that correlate with challenging questions among several models. We find that, although each model has its own error-modes, 8 subroutines are shared among all 3 of them (right). One of the patterns we find then analysing the shared code structures is reasoning about the order of events (left).

We manually inspect the eight subroutines that are shared among all three sets and identify that they represent two clear patterns (the actual subtrees are listed in the appendix, Section 9.2).

The first group of primitives, manifesting in such structures as those containing *For loops* with complex control flow in them, captures reasoning about not just the content of the frame, but also its placement in a sequence of events. We provide an example of a corresponding subtree together with a question that requires this reasoning pattern in Figure 4 (left). The second group contains primitives that represent detailed analysis of specific elements (objects, relationships) within a scene. The examples include questions that require identifying the precise placement of an object within a frame.

Finally, we note that models with uniformly low performance may not identify some certain subroutines as distinctly more challenging than average. Focusing on subroutines that challenge only the state-of-the-art model, we find that those involving long-term activities, as opposed to atomic actions, are particularly difficult for SeViLA, but not for others.

In summary, we discovered that VideoQA methods struggle with fine-grained temporal reasoning and lack spatio-temporal, object-centric representations. This is in accord with prior studies (Buch et al., 2022; Huang et al., 2018; Liu et al., 2021) that demonstrated that naive, single-frame baselines can achieve top performance on mainstream VideoQA benchmarks, which were used to develop these methods. Next, we show how our approach can be used to automatically generate a new benchmark that challenges existing approaches and encourages the development of a novel generation of video-language representations.

5 Dataset Generation

We apply our method for automatically generating challenging questions described in Section 3.4 to create a new VideoQA benchmark, CodePlex-QA. We begin by detailing the source datasets and key implementation details in Section 5.1. We then compare the performance of these methods on the popular NExT-QA (Xiao et al., 2021) to that on CodePlex-QA in Section 5.2 to validate the effectiveness of our approach. Our dataset will be released.

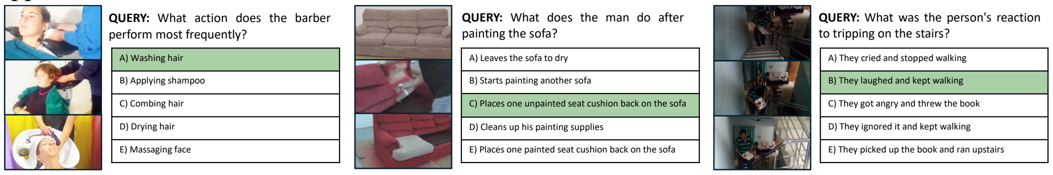
5.1 Experimental Setup

Source datasets. To encourage diversity we generate questions using 3 different datasets, all of which provide scene-graphs annotations: MOMA (Luo et al., 2021; 2022); ActivityNet (Caba Heilbron et al., 2015), which we combine with ActivityNet-Entities (Zhou et al., 2019) and ActivityNet-Captions (Krishna et al., 2017), and the ActionGenome (Ji et al., 2020) annotations for Charades (Sigurdsson et al., 2016). This results in pool of 4191 videos that are passed to our algorithm.

Implementation details. We use GPT-4 (OpenAI, 2023b) to generate question and answer candidates following prior work by Mangalam et al. (2023) and leverage a state-of-the-art image captioning model, LLaVA 1.5 (Liu et al., 2023a;b), to list visual attributes of the main actors and objects in the videos. Following Xiao et al. (2021) we generate 5 answer candidates for each question (1 correct answer and 4 distractors), and use accuracy as the evaluation metric. We set the δ in Equation 8

Dataset	Tarsier	SeViLA ZS	ViperGPT	InternVideo	VIOLET	Random
NExT-QA	70.9%	64.2%	60.0%	50.9%	37.7%	20.0%
ATP-Hard	59.8%	54.9%	51.8%	24.6%	25.4%	20.0%
CodeplexQA	52.5%	43.7%	45.8%	29.9%	27.6%	20.0%

Table 2: Difference in prediction accuracy between the manually annotated NExT-QA and our automatically generated CodePlex-QA for a representative set of zero-shot VideoQA models. Our benchmark is empirically 1.9 times harder, validating the effectiveness of our complexity estimation approach.



QUERY: What action does the barber perform most frequently?

- A) Washing hair
- B) Applying shampoo
- C) Combing hair
- D) Drying hair
- E) Massaging face

QUERY: What does the man do after painting the sofa?

- A) Leaves the sofa to dry
- B) Starts painting another sofa
- C) Places one unpainted seat cushion back on the sofa
- D) Cleans up his painting supplies
- E) Places one painted seat cushion back on the sofa

QUERY: What was the person's reaction to tripping on the stairs?

- A) They cried and stopped walking
- B) They laughed and kept walking
- C) They got angry and threw the book
- D) They ignored it and kept walking
- E) They picked up the book and ran upstairs

Figure 5: Example questions in CodePlex-QA generated with our approach. It features many challenges that are under-represented in existing, manually-designed benchmarks and motivates the development of novel approaches with enhanced spatio-temporal modeling capabilities.

to select the top 10% of the data according to the estimated complexity (calibrated on NExT-QA). Further details are provided in Section 8 of the appendix.

5.2 Results

To construct CodePlex-QA, we run the generation pipeline described in Section 3.4, obtaining 20791 candidate questions (several question candidates are generated for each video). Then we calculate each question's complexity score using CodePlexity to only retain questions that meet or exceed the minimum complexity threshold as in Equation 8. The resulting datasets consists of 2261 questions. The final manual filtering to ensure the answerability of the generated questions removes only 12% of the candidates, leaving us 1981 samples, all of which are used for evaluation.

We then evaluate all the zero-shot baselines from our pool of methods on CodePlex-QA and report their accuracy in Table 2. We additionally report the performance of these models on the popular NExT-QA benchmark for reference. Note that the performance of the random choice baseline is the same for the two benchmarks, so the numbers are directly comparable. Firstly, we observe that that the average success rate of models on our generated questions is significantly lower than on NExT-QA. Specifically, CodePlex-QA is 1.9 times harder than the manually annotated NExT-QA (dataset complexity estimated by taking the average performance of the 5 methods and subtracting random chance). The ranking is consistent across benchmarks, but it is notable that the gap between InternVideo and VIOLET is smaller on CodePlex-QA, indicating limitations in InternVideo's video understanding despite more visual-language data.

Finally, we visualize a representative sample of generated questions in Figure 5. We observe that CodePlex-QA features a variety of challenging scenarios that require fine-grained temporal reasoning (e.g., comparing the frequency of different actions), as well as reasoning about objects in videos (e.g., tracking seat cushions). More examples are shown in the supplementary video.

6 Conclusion

We have demonstrated that generated code complexity is a reliable measure of question complexity for VideoQA, proposing a novel metric that outperforms existing metrics on this task. In addition, our approach allows us to identify individual subroutines that correlate with challenging questions for a wide range of models, yielding new insights into the key challenges of VideoQA. Finally, we have shown how our metric can be used to automatically generate a novel benchmark – CodePlex-QA, which is 1.9 times harder for existing models than the popular, manually labeled NExT-QA dataset. As new methods and benchmarks are developed to address the challenges identified in our work, our approach can be re-applied, thereby ensuring continued progress in the field.

References

T Aflalo, CY Zhang, ER Rosario, N Pouratian, GA Orban, and RA Andersen. A shared neural substrate for action verbs and observed actions in human posterior parietal cortex. *Science advances*, 6(43), 2020. 1

Sandhini Agarwal, Gretchen Krueger, Jack Clark, Alec Radford, Jong Wook Kim, and Miles Brundage. Evaluating CLIP: towards characterization of broader capabilities and downstream implications. *arXiv preprint arXiv:2108.02818*, 2021. 28

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *CVPR*, 2016. 3

Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding? In *ICML*, 2021. 3

Shyamal Buch, Cristóbal Eyzaguirre, Adrien Gaidon, Jiajun Wu, Li Fei-Fei, and Juan Carlos Niebles. Revisiting the" video" in video-language understanding. In *CVPR*, 2022. 2, 3, 7, 9

Richard H Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on scientific computing*, 16(5):1190–1208, 1995. 18

Fabian Caba Heilbron, Victor Escorcia, Bernard Ghanem, and Juan Carlos Niebles. Activitynet: A large-scale video benchmark for human activity understanding. In *CVPR*, 2015. 9, 18, 19, 20, 25, 28

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. 3, 18

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*, 2023. 17

Jean Decety and Julie Grèzes. Neural mechanisms subserving the perception of human actions. *Trends in cognitive sciences*, 3(5):172–178, 1999. 1

Jean Decety, Julie Grezes, Nicolas Costes, Daniela Perani, Marc Jeannerod, Emmanuel Procyk, Franco Grassi, and Ferruccio Fazio. Brain activity during observation of actions. influence of action content and subject's strategy. *Brain: a journal of neurology*, 120(10):1763–1777, 1997. 1

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 7, 8

Arpad E Elo. The proposed uscf rating system, its development, theory, and applications. *Chess Life*, 22(8):242–247, 1967. 7, 17

Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88: 303–338, 2010. 7

Cristobal Eyzaguirre and Alvaro Soto. Differentiable adaptive computation time for visual reasoning. In *CVPR*, 2020. 3

Christoph Feichtenhofer, Yanghao Li, Kaiming He, et al. Masked autoencoders as spatiotemporal learners. *NeurIPS*, 2022. 3

Jinlan Fu, See-Kiong Ng, Zhengbao Jiang, and Pengfei Liu. GPTScore: Evaluate as you desire. *arXiv preprint arXiv:2302.04166*, 2023. 17

Tsu-Jui Fu, Linjie Li, Zhe Gan, Kevin Lin, William Yang Wang, Lijuan Wang, and Zicheng Liu. VIOLET: End-to-end video-language transformers with masked visual-token modeling. *arXiv preprint arXiv:2111.12681*, 2021. 1, 3, 7

Jiaxin Ge, Sanjay Subramanian, Baifeng Shi, Roei Herzig, and Trevor Darrell. Recursive visual programming. *arXiv preprint arXiv:2312.02249*, 2023. 3, 26

Alex Graves. Adaptive computation time for recurrent neural networks. In *NIPS*, 2016. 3

Madeleine Grunde-McLaughlin, Ranjay Krishna, and Maneesh Agrawala. AgQA: A benchmark for compositional spatio-temporal reasoning. In *CVPR*, 2021. 4

Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *CVPR*, 2023. 2, 3, 4

Alfred V Hoe, Ravi Sethi, and Jeffrey D Ullman. *Compilers—principles, techniques, and tools*. Pearson Addison Wesley Longman, 1986. 5

Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. Learning to reason: End-to-end module networks for visual question answering. In *ICCV*, 2017. 3

De-An Huang, Vignesh Ramanathan, Dhruv Mahajan, Lorenzo Torresani, Manohar Paluri, Li Fei-Fei, and Juan Carlos Niebles. What makes a video a video: Analyzing temporal information in video understanding models and datasets. In *CVPR*, 2018. 2, 9

Jingwei Ji, Ranjay Krishna, Li Fei-Fei, and Juan Carlos Niebles. Action genome: Actions as compositions of spatio-temporal scene graphs. In *CVPR*, 2020. 6, 9, 18, 19, 20, 25, 28

Pin Jiang and Yahong Han. Reasoning with heterogeneous graph alignment for video question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 11109–11116, 2020. 7

Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *CVPR*, 2017a. 4

Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Judy Hoffman, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Inferring and executing programs for visual reasoning. In *ICCV*, 2017b. 3

Seung Wook Kim, Makarand Tapaswi, and Sanja Fidler. Visual reasoning by progressive module networks. In *ICLR*, 2018. 3

Tom Kocmi and Ondřej Bojar. Curriculum learning and minibatch bucketing in neural machine translation. In *RANLP*, 2017. 3

Andrei N Kolmogorov. On tables of random numbers. *Sankhyā: The Indian Journal of Statistics, Series A*, pp. 369–376, 1963. 3, 4

Oliver Kramer and Oliver Kramer. Scikit-learn. *Machine learning for evolution strategies*, pp. 45–53, 2016. 18

Ranjay Krishna, Kenji Hata, Frederic Ren, Li Fei-Fei, and Juan Carlos Niebles. Dense-captioning events in videos. In *ICCV*, 2017. 9, 20

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. BLIP-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *ICML*, 2023. 7

Kunchang Li, Yali Wang, Yinan He, Yizhuo Li, Yi Wang, Yi Liu, Zun Wang, Jilan Xu, Guo Chen, Ping Luo, et al. Mvbench: A comprehensive multi-modal video understanding benchmark. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22195–22206, 2024. 27

Rensis Likert. A technique for the measurement of attitudes. *Archives of psychology*, 1932. 7, 17

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning. In *NeurIPS Workshop on Instruction Tuning and Instruction Following*, 2023a. 9, 19

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *NeurIPS*, 2023b. 9, 19

Haotian Liu, Chunyuan Li, Yuheng Li, Bo Li, Yuanhan Zhang, Sheng Shen, and Yong Jae Lee. Llava-next: Improved reasoning, ocr, and world knowledge, January 2024. URL https://llava-vl.github.io/blog/2024-01-30-llava-next/. 27

Xin Liu, Silvia L Pintea, Fatemeh Karimi Nejadasl, Olaf Booij, and Jan C Van Gemert. No frame left behind: Full video action recognition. In *CVPR*, 2021. 2, 9

Zelun Luo, Wanze Xie, Siddharth Kapoor, Yiyun Liang, Michael Cooper, Juan Carlos Niebles, Ehsan Adeli, and Fei-Fei Li. MOMA: Multi-object multi-actor activity parsing. *NeurIPS*, 2021. 6, 9, 18, 25, 28

Zelun Luo, Zane Durante, Linden Li, Wanze Xie, Ruochen Liu, Emily Jin, Zhuoyi Huang, Lun Yu Li, Jiajun Wu, Juan Carlos Niebles, et al. MOMA-LRG: Language-refined graphs for multi-object multi-actor activity parsing. *NeurIPS*, 2022. 6, 9, 18, 25, 28

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *NeurIPS*, 2023. 17

Karttikeya Mangalam, Raiymbek Akshulakov, and Jitendra Malik. EgoSchema: A diagnostic benchmark for very long-form video language understanding. In *NeurIPS*, 2023. 2, 6, 9, 24

Thomas J McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4): 308–320, 1976. 2, 3, 4

Sachit Menon and Carl Vondrick. Visual classification via description from large language models. In *ICLR*, 2022. 6

OpenAI. Chatgpt. https://www.openai.com, 2023a. 2

OpenAI. Gpt-4 technical report, 2023b. 7, 8, 9, 17

Stefan Palan and Christian Schitter. Prolific. ac—a subject pool for online experiments. *Journal of Behavioral and Experimental Finance*, 17:22–27, 2018. 7, 16

Emmanouil Antonios Platanios, Otilia Stretcu, Graham Neubig, Barnabás Poczós, and Tom Mitchell. Competence-based curriculum learning for neural machine translation. In *ACL*, 2019. 3

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *ICML*, 2021. 28

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *NeurIPS*, 2023. 17

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *ICML*, 2021. 3

Xindi Shang, Donglin Di, Junbin Xiao, Yu Cao, Xun Yang, and Tat-Seng Chua. Annotating objects and relations in user-generated videos. In *Proceedings of the 2019 on International Conference on Multimedia Retrieval*, pp. 279–287. ACM, 2019. 22

Gunnar A Sigurdsson, Gül Varol, Xiaolong Wang, Ali Farhadi, Ivan Laptev, and Abhinav Gupta. Hollywood in homes: Crowdsourcing data collection for activity understanding. In *ECCV*, 2016. 9, 20

Ray J Solomonoff. Algorithmic probability: Theory and applications. *Information theory and statistical learning*, pp. 1–23, 2009. 3

Petru Soviany, Claudiu Ardei, Radu Tudor Ionescu, and Marius Leordeanu. Image difficulty curriculum for generative adversarial networks (CuGAN). In *WACV*, 2020. 3

Valentin I Spitkovsky, Hiyan Alshawi, and Dan Jurafsky. From baby steps to leapfrog: How "less is more" in unsupervised dependency parsing. In *ACL*, 2010. 3

Sanjay Subramanian, Medhini Narasimhan, Kushal Khangaonkar, Kevin Yang, Arsha Nagrani, Cordelia Schmid, Andy Zeng, Trevor Darrell, and Dan Klein. Modular visual question answering via code generation. In *ACL*, 2023. 2, 3, 4

Dídac Surís, Sachit Menon, and Carl Vondrick. ViperGPT: Visual inference via python execution for reasoning. In *ICCV*, 2023. 1, 2, 3, 4, 7, 18

Yi Tay, Shuohang Wang, Anh Tuan Luu, Jie Fu, Minh C Phan, Xingdi Yuan, Jinfeng Rao, Siu Cheung Hui, and Aston Zhang. Simple and effective curriculum pointer-generator networks for reading comprehension over long narratives. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4922–4931, 2019. 3

Bart Thomee, David A Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. Yfcc100m: The new data in multimedia research. *Communications of the ACM*, 59(2):64–73, 2016. 22

Yulia Tsvetkov, Manaal Faruqui, Wang Ling, Brian MacWhinney, and Chris Dyer. Learning the curriculum with bayesian optimization for task-specific word representation learning. In *ACL*, 2016a. 3

Yulia Tsvetkov, Manaal Faruqui, Wang Ling, Brian MacWhinney, and Chris Dyer. Learning the curriculum with bayesian optimization for task-specific word representation learning. In *ACL*, 2016b. 3, 7

Radu Tudor Ionescu, Bogdan Alexe, Marius Leordeanu, Marius Popescu, Dim P Papadopoulos, and Vittorio Ferrari. How hard can it be? estimating the difficulty of visual search in an image. In *CVPR*, 2016. 3

Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *NeurIPS*, 2017. 3

Jiawei Wang, Liping Yuan, Yuchen Zhang, and Haomiao Sun. Tarsier: Recipes for training and evaluating large video description models. *arXiv preprint arXiv:2407.00634*, 2024. 7

Yi Wang, Kunchang Li, Yizhuo Li, Yinan He, Bingkun Huang, Zhiyu Zhao, Hongjie Zhang, Jilan Xu, Yi Liu, Zun Wang, et al. InternVideo: General video foundation models via generative and discriminative learning. *arXiv preprint arXiv:2212.03191*, 2022a. 1, 7

Zhenhailong Wang, Manling Li, Ruochen Xu, Luowei Zhou, Jie Lei, Xudong Lin, Shuohang Wang, Ziyi Yang, Chenguang Zhu, Derek Hoiem, et al. Language models with image descriptors are strong few-shot video-language learners. *NeurIPS*, 2022b. 6

Yunchao Wei, Xiaodan Liang, Yunpeng Chen, Xiaohui Shen, Ming-Ming Cheng, Jiashi Feng, Yao Zhao, and Shuicheng Yan. Stc: A simple to complex framework for weakly-supervised semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(11):2314–2320, 2016. 3, 22

Bo Wu, Shoubin Yu, Zhenfang Chen, Joshua B Tenenbaum, and Chuang Gan. STAR: A benchmark for situated reasoning in real-world videos. In *NeurIPS*, 2021. 4

Moritz F Wurm and Alfonso Caramazza. Two 'what' pathways for action and object recognition. *Trends in cognitive sciences*, 26(2):103–116, 2022. 1

Junbin Xiao, Xindi Shang, Angela Yao, and Tat-Seng Chua. Next-QA: Next phase of question-answering to explaining temporal actions. In *CVPR*, 2021. 2, 7, 9

Haiyang Xu, Qinghao Ye, Ming Yan, Yaya Shi, Jiabo Ye, Yuanhong Xu, Chenliang Li, Bin Bi, Qi Qian, Wei Wang, et al. mPLUG-2: A modularized multi-modal foundation model across text, image and video. *arXiv preprint arXiv:2302.00402*, 2023. 3

Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Josh Tenenbaum. Neural-symbolic vqa: Disentangling reasoning from vision and language understanding. *NeurIPS*, 2018. 3

Shoubin Yu, Jaemin Cho, Prateek Yadav, and Mohit Bansal. Self-chained image-language model for video localization and question answering. In *NeurIPS*, 2023a. 1, 2, 3, 7

Zhou Yu, Lixiang Zheng, Zhou Zhao, Fei Wu, Jianping Fan, Kui Ren, and Jun Yu. ANetQA: A large-scale benchmark for fine-grained compositional reasoning over untrimmed videos. In *CVPR*, 2023b. 4

Rowan Zellers, Ximing Lu, Jack Hessel, Youngjae Yu, Jae Sung Park, Jize Cao, Ali Farhadi, and Yejin Choi. Merlot: Multimodal neural script knowledge models. *NeurIPS*, 2021. 3

Andy Zeng, Maria Attarian, Brian Ichter, Krzysztof Choromanski, Adrian Wong, Stefan Welker, Federico Tombari, Aveek Purohit, Michael Ryoo, Vikas Sindhwani, et al. Socratic models: Composing zero-shot multimodal reasoning with language. In *ICLR*, 2023. 6

Yaoyao Zhong, Wei Ji, Junbin Xiao, Yicong Li, Weihong Deng, and Tat-Seng Chua. Video question answering: Datasets, algorithms and challenges. In *EMNLP*, 2022. 7

Luowei Zhou, Yannis Kalantidis, Xinlei Chen, Jason J. Corso, and Marcus Rohrbach. Grounded video description. In *CVPR*, 2019. 6, 9, 20

Alexander K Zvonkin and Leonid A Levin. The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms. *Russian Mathematical Surveys*, 25(6):83, 1970. 3

APPENDIX

This appendix includes further details, results and discussions that were not included in the main paper due to space limitations:

1. Section 7 provides **additional technical details** for our baselines and complexity estimation methods.
2. Section 8 compliments Section 5 in the main paper providing **extra technical specifics** regarding the dataset generation pipeline.
3. Section 9 reports **additional results and analysis** to those in Sections 4.2 and 4.3 in main paper.
4. Section 11 includes a **discussion of broader impact and limitations** of our work.

We also include a separate video with qualitative examples of the analyzed questions and samples from our new dataset as part of the supplementary materials (see `1394.mp4`, Codec H.264). Finally, we re-iterate that we plan to release code, models, and other materials as part of the final supplement.

7 ADDITIONAL TECHNICAL DETAILS

7.1 MERGING DUPLICATE SUBTREES

To avoid duplicated subtrees and reduce redundancy, we merge subtrees that always co-occur when one is a descendant of the other. Specifically, a subtree S_1 is said to always co-occur with another subtree S_2 if every occurrence of S_2 in the dataset $\mathcal{D}$ is also an occurrence of S_1. In such cases, since S_2 is always contained within S_1, we can merge S_2 into S_1 without losing any unique patterns.

Merging these subtrees does not risk missing important patterns because any syntactic or semantic information captured by S_2 is inherently included in S_1. This is due to the fact that S_1 encompasses all occurrences of S_2, ensuring that the features associated with S_2 are preserved within S_1. By eliminating redundant subtrees, we streamline the dataset, which can improve computational efficiency without compromising data integrity.

The merged set of subtrees $\mathcal{S}_{\text{merged}}(\mathcal{D})$ is defined as:

$$\begin{aligned}\mathcal{S}_{\text{merged}}(\mathcal{D}) = \mathcal{S}(\mathcal{D}) \setminus \{S_2 \in \mathcal{S}(\mathcal{D}) \mid \exists S_1 \in \mathcal{S}(\mathcal{D}) :\\ (\forall T \in \mathcal{D},\ ISO(T, S_2) \rightarrow ISO(T, S_1)) \land (S_2 \subseteq S_1)\}\end{aligned} \tag{10}$$

Here, $ISO(T, S)$ indicates that subtree S is isomorphic to a subtree within program T, and $S_2 \subseteq S_1$ denotes that S_2 is contained within S_1.

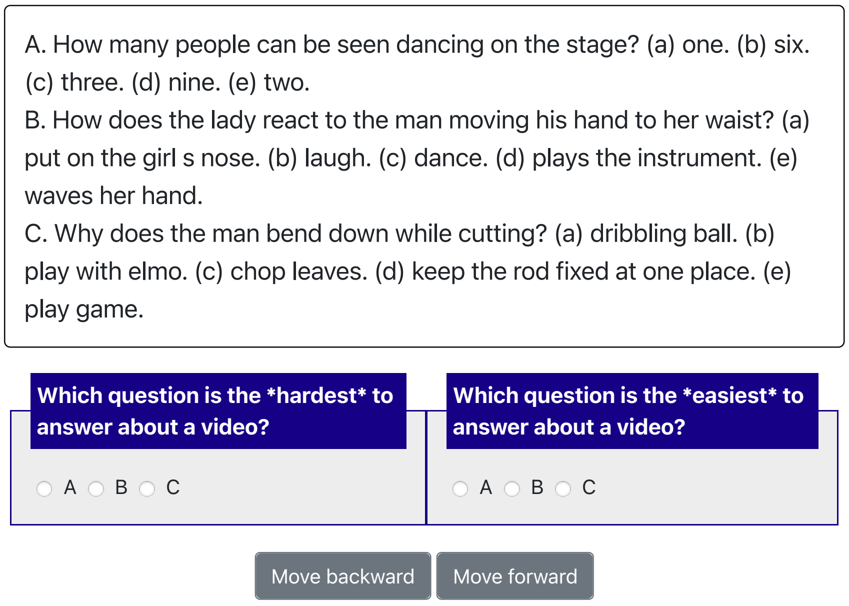
By applying this merging strategy, we ensure that all significant patterns are retained. The one-hot encodings of S_1 and S_2 are identical across all programs where they appear, so merging them does not alter the representation of the data. This approach maintains the richness of the syntactic structures while optimizing the dataset for analysis.

7.2 HUMAN ANNOTATION INTERFACE AND PROCESSING

For our human baseline we conduct an annotation effort on a subset of 150 questions from the validation set of the NExT-QA dataset. To this end, we recruited 65 human subjects via the Prolific platform (Palan & Schitter, 2018), using the provided filters to select for annotators that are proficient in English.

The annotators were shown 50 sets of 3 questions (one set at a time), where they were asked to sort the questions according to their perceived complexity by indicating which questions were the *easiest* and *hardest*. An example set and the annotation interface is shown in Figure 6. Consistency was validated by repeating pairs of questions multiple times (the third question can vary). We check that relative orders remain consistent and don't consider subjects who demonstrated low consistency.

We further filter out annotations that were done in too little time, and annotators who finished the complete study in less than a minimum reasonable time. The annotations from the remaining 30 subjects were used to calculate the total ordering of the questions.


A. How many people can be seen dancing on the stage? (a) one. (b) six. (c) three. (d) nine. (e) two.
B. How does the lady react to the man moving his hand to her waist? (a) put on the girl s nose. (b) laugh. (c) dance. (d) plays the instrument. (e) waves her hand.
C. Why does the man bend down while cutting? (a) dribbling ball. (b) play with elmo. (c) chop leaves. (d) keep the rod fixed at one place. (e) play game.

Which question is the *hardest* to answer about a video?

○ A ○ B ○ C

Which question is the *easiest* to answer about a video?

○ A ○ B ○ C

Figure 6: We ask human annotators to provide the relative ordering of three provided questions according to the estimated complexity of answering the question about an *unseen* video.

We compute the final order of the questions using Elo scores (Elo, 1967). Originally developed to rank chess players, the Elo system models the outcome probability of unseen comparison between a pair of entities (*eg.* chess players or, in our case, questions) as a function of their score ratings s_i. In a comparison each entity's comparative performance is assumed to be Normally distributed around their score with fixed variance β^2. The probability of a favorable outcome for entity i when compared to an opponent entity j is given by the probability that the performance of i surpasses the performance of j:

$$\Phi\left(\frac{s_i - s_j}{\sqrt{2}\beta}\right), \tag{11}$$

where Φ represents the cumulative distribution of a zero-mean, unit-variance Gaussian. The scores are updated after every comparison according to the Elo update rule.

7.3 GPT QUESTION COMPLEXITY SCORING

In order to automatically estimate the complexity of a question directly from it's text without biasing the method towards any specific definition of complexity we refer to the Natural Language Processing literature which has shown that assessments made by Large Language Models correlate with human judgement (Madaan et al., 2023; Fu et al., 2023; Chiang et al., 2023; Rafailov et al., 2023). To this end, we leverage GPT-4 (OpenAI, 2023b) to generate a complexity score on a Likert scale (Likert, 1932) (ranging from one to five). We set the temperature to zero (for replicable results) and generate a single token with the score. We prompt the model as follows:

Prompt

[SYSTEM] You are an assistant that -for the provided question and its corresponding answer options- estimates the complexity of answering said question about an unknown video. Return your answer as score from 1 to 5 (1 being the easiest and 5 being the hardest).

[USER] I'll provide a question and its candidate answers. Estimate the complexity of answering the question about a (unseen) video. Output should ONLY be the integer score (1-5) that you assign to the question (ie. no JSON, no text, no markdown, no nothing).

$query$ A: $answers[0]$, B: $answers[1]$, C: $answers[2]$, D: $answers[3]$, E: $answers[4]$

7.4 CodeGen details

We use the same API as in the original ViperGPT paper (Surís et al., 2023). For NExT-QA analysis we use the programs and predictions from (Surís et al., 2023), which were generated by Codex (Chen et al., 2021), a Code Completion LLM. Since Codex is no longer avaialble, for the CodePlex-QA code generations we instead use a text variant of GPT-3.5 with support for 16k context window (necessary because of the long API Specification). We prompt this model with both the API and a System message to make sure the output is usable as code. Additionally, we process the output to extract the code and format it such that it can be executed and analyzed. We include the prompt used:

Prompt

[SYSTEM]Only use the functions you have been provided with."
[SYSTEM]Only complete the code. Don't include markdown syntax (eg. ticks).
[USER]<API Spec.>
# $query$
# [$answers[0]$, $answers[1]$, $answers[2]$, $answers[3]$, $answers[4]$]
def execute_command(video, possible_answers, question):
# Reason every step

7.5 Logistic Regression Parameters

We use the SciKit-Learn (Kramer & Kramer, 2016) implementation of Logistic Regression model. We train until convergence and choose parameters based on the reported mean accuracy over 5 folds. The resulting model uses L2 regularization with weight $c = 1.0$ and is trained with the L-BFGS solver (Byrd et al., 1995).

7.6 Cyclomatic Complexity Calculation

We compute of Cyclomatic Complexity via an open source implementation[1].

8 Additional Question Generation Details

As noted in Section 5 of the main paper, we leverage existing video datasets with scene-graph annotations MOMA (Luo et al., 2021; 2022), ActivityNet (Caba Heilbron et al., 2015), and Action Genome (Ji et al., 2020). As necessary step we need to translate the annotations into a textual format such that a generative language model could use it. We begin by identifying the main activity and its sub-activities, including the start and end times of each. This temporal framework serves as a scaffold for the detailed enumeration of the actors and objects involved. Actors and objects are cataloged not just by their presence, but also in relation to specific sub-activities. When a high level textual description is not available we leverage captioning models to generate visual descriptors of the actors in the video. More details in Section 8.1.

The resulting dataset has an average of 2.40 questions per video. The duration of each video ranges from approximately 3 seconds to 10 minutes with an average video duration of about 1.5 minutes. This diverse range of video lengths is desirable as it is conducent to generating a wide variety of questions.

The following sections describe the methods and prompts we used to translate the graphs into textual *scripts* for each specific dataset: MOMA (Luo et al., 2021; 2022) in Section 8.2, ActivityNet

[1] https://radon.readthedocs.org

(Caba Heilbron et al., 2015) in Section 8.3, and Action Genome (Ji et al., 2020) in Section 8.4. Finally, Section 8.5 describes how the generated *scripts* are used to generate new questions.

8.1 Visual Descriptors extraction

A limitation of using Scene Graphs is that they tipically don't include visual descriptions of the nodes they relate. This is in juxtaposition with the way humans typically refer to actors and objects. To this end we describe the main actors in the video using a Captioning model. In particular, we use Llava1.5 (Liu et al., 2023a;b) to describe a single instance of the actor in the video. We leverage the included bounding box annotations for the actors in each annotated interaction. We choose the bounding box with the largest area (in pixels) in the first subactivity the actor appears in. We then crop the relevant area and zero-pad the borders to make the final image square, as this is the format that Llava1.5 was trained with. The resulting cropped image is passed into the captioning model along with a prompt modified from Llava.

Prompt

> ”A chat between a curious human and an artificial intelligence assistant. The assistant gives helpful, detailed, and polite answers to the human's questions.
> [USER]<image>
> Look at the picture and tell me only about the person's looks that don't change. Like what they're wearing, their hair color and style. **Don't talk about where they are OR what they're doing**. (Tag is <actor_classname>).
> [ASSISTANT]

Importantly, we also pass in the textual identifier for the actor, indicated in the prompt as <actor_classname>.

8.2 MOMA

A significant limitation specific to MOMA is the inconsistency in annotated identifiers for object or actor throughout the entire video. Consequently, we exclude videos in the dataset for which objects or actors cannot be reliably identified. Our implementation of the filtering process eliminates any videos from the dataset where the identities of actors are not consistently recognizable based on their $class\_name$ identifier. In practice, we define a Python function to detect 'collisions' - instances where the same identifier is used for different class names within a subactivity, or across different subastivities without a consistent mapping.

As noted in Section 5 of the main paper, we need to translate the contents of the scene-graph-in-time annotated in MOMA into a textual format such that a generative language model could use it. We now describe the method we used to translate the graphs into textual *scripts*.

We begin by identifying the main activity and its sub-activities, including the start and end times of each. This temporal framework serves as a scaffold for the detailed enumeration of the actors and objects involved. Actors and objects are cataloged not just by their presence, but also in relation to specific sub-activities. We identify their class names and descriptive attributes along with arrival their departure times within each sub-activity and store these for later. We also track state changes and action, both transitive and intransitive, that occur during the sub-activities, along with the identifiers that map to the actors and objects involved.

The final script is structured in a hierarchical format, starting with the main activity title and its timeframe, followed by detailed sections for each sub-activity. These sections enumerate the actors present, and a chronological account of events, actions, and state changes. We also generate a descriptive caption for each actor involved following Section 8.1 and include it in the prompt. An example of an activity and its first sub-activity is shown:

> # **Activity:** ”Dining” (0-597)
> All actors:

- **ROLE:** customer. **Visual description:** The person in the picture is a woman with long, dark hair. She is wearing a white shirt and a black tie.
- **ROLE:** waiter. **Visual description:** The waiter in the picture is a young man wearing a white shirt and a black tie.

## **Sub activity (0-10):** The waiter is talking to the customer or helping them into their seat

- **Actors present:** customer, waiter
- **Happened during sub-activity:**
  - (attribute) waiter standing
  - (transitive action) waiter talking to customer
  - (intransitive action) waiter bending

8.3 ActivityNet

Although the original ActivityNet (Caba Heilbron et al., 2015) dataset didn't include scene-graphs, follow up work ActivityNet-Entities (Zhou et al., 2019) provides additional annotations for objects, attributes, relationships and actions. Further, we also use the per-subactivity captions in the ActivityNet-Captions (Krishna et al., 2017) dataset.

As was the case with MOMA, we translate the contents of the scene-graph-in-time annotated into a textual format such that it can be parsed by a generative language model. We once again divide a video into a main activity and its component subactivities, and take note of their start and end times. Actors present in a particular subactivity are listed within the subactivity description, along with their provided visual descriptions when available in ActivityNet-Entities (Zhou et al., 2019). Finally, we filter relationships such that we only keep those that involve actors and list those for each actor. An example of an activity and a sub-activity is shown below:

# **Activity:** ”doing archery” (time: 11-177)

## **Sub activity** (15-39): He loads an arrow in the bow.
All actors descriptions from subactivity:

- Visual description (time: 31): attributeclass: person - age&sex: man - hairstyle: straight - hairlength: short - haircolor: [’black’] - accessory: [’glove’] - skincolor: white - upperclothestype: t-shirt - upperclothescolor: [’white’]- lowerclothestype: shorts - lowerclothescolor: [’black’] - status: [’standing’, ’shooting’] - location: outdoors
- Relations for actor in subactivity:
  - person pulling bow
  - person holding arrow

8.4 Action Genome

We leverage the scene-graph annotations in ActionGenome (Ji et al., 2020) for the Charades video dataset (Sigurdsson et al., 2016) and use the annotated activity along with its duration and high level description. When available we also include the location and other descriptions. We then list the annotated actions (along with their respective start and end times). Finally, we iterate over the annotated-per-frame object-actor relationships and track when their state changes. We provide the timestamp at which the state-change occurred and the change itself to the generation model. As was the case with ActivityNet, we don't need to generate visual attributes as the high level description often provides them. An example *script* is shown below:

# **Activity** (duration: 30.62): ”A person sits at a desk in the living room. The person laughs as they pick up a bag of groceries from under the desk.”

Location: ”Bedroom”

Other descriptions:

- A person is sitting at adesk they pick up a bag and then they get up
- The person is sitting at the computer desk and bends over to pick up the garbage, which he sits on his lap, and then gets up carrying the garbage.

Actions:

- Taking a bag from somewhere (9.00, 16.40)
- Sitting at a table (0.00, 29.30)
- Someone is standing up from somewhere (25.00, 30.80)
- Holding a bag (12.70, 32.00)
- Someone is laughing (0.00, 30.80)
- Sitting in a chair (0.00, 32.00)

Relation Changes (wrt. actor):

- bag goes from ”’holding’” to ”’touching’ at 18.0”
- bag goes from ”’touching’” to ”’holding’ at 20.0”
- bag goes from ”’holding’” to ”’touching’ at 26.0”
- bag goes from ”’touching’” to ”’holding’ at 27.0”
- chair goes from ”’sitting_on’” to ”None at 28.0”
- chair goes from ”None” to ”’not_contacting’ at 28.0”

8.5 Question Generation from Scripts

Each textual *script* generated above is combined with a prompt requesting that the language model output *interesting* questions about the video, without specifying that they should be hard, or how *interesting* should be interpreted. The complete prompt is used to condition a Large Language Model to generate the requested questions and answer candidates in a JSON format. The chosen language model is GPT-4. We set the sampling temperature to zero and decode greedily (for replicability). We include the exact prompt used here:

Prompt

[SYSTEM]You generate *interesting* questions to ask about the video for which the description is provided. Pretend you don’t get the exact description (ie. no exact times or player ids) but you did watch the video, so you have a notion of what happens, and when.
[SYSTEM]Return a list of Multiple Choice questions formated as a json with q, ans, dist1, dist2, dist3, dist4 keys. ‘distN‘ are 4 distractors..
[USER]What are interesting questions to ask about this video? (description provided)
Return a numbered list of Multiple Choice questions formated as a json with q, ans, dist1, dist2, dist3, dist4 keys. ‘distN‘ are 4 distractors
Try to use visual descriptions of the actors instead of their role sometimes (eg. the person with the red shirt instead of the waiter).

NEVER say subactivities. Eg. don’t say ”first subactivity”, instead say ”while the waiters served the drinks”.

Video description:

—
<DESC>

—

Remember, **NEVER** say subactivities. Eg. don't say "first subactivity", instead say "while the waiters served the drinks".

When generating questions for videos from the MOMA dataset we also include an additional instruction to make sure the model doesn't refer to actors as *unclassified* when the annotations are missing descriptions:

Also avoid saying "unclassified ..." to refer to actors. If you weren't provided with a role use visual descriptions instead.

9 Additional Results and Analysis

9.1 Relation to Video Complexity

Our approach to estimating the complexity of VideoQA tasks is grounded in insights from classical complexity theory, specifically the Chain Rule for Kolmogorov Complexity. According to this rule, the complexity of a composite entity, such as a (video, question) pair, can be expressed as the sum of the complexity of one component (e.g., the question) and the conditional complexity of the other component (e.g., the video conditioned on the question), plus a logarithmic term:

$$K(x, y) = K(x) + K(y|x) + O(\log(K(x, y))) \tag{12}$$

In cases where there is minimal shared information between the video and the question, the complexity of the pair can be approximated as the sum of their individual complexities, up to this logarithmic term:

$$K(x, y) \approx K(x) + K(y) + O(\log(K(x, y))) \tag{13}$$

In this work, we focus on estimating the complexity of the question, which parallels the structure suggested by the Chain Rule for Kolmogorov Complexity. If we treat the video as x and the question as y, then the complexity of the (video, question) pair can be thought of in a similar way, where the complexity of the question $K(y)$ is a key component of the overall task complexity. While we do not directly compute Kolmogorov complexities, this analogy provides a theoretical motivation for focusing only on the question complexity. Estimating the complexity of the question is valuable because it can later be combined with robust methods for assessing the complexity of the video to achieve a more comprehensive measure of the total task complexity in future work.

The difference between the predicted complexity of the question and the actual model performance on a given question can be used as a proxy for estimating the video's complexity. Prior work by Wei et al. (2016) has approximated image complexity by the number of objects present in an image. We extend this approach to video complexity, utilizing the VidOR dataset (Shang et al., 2019), which provides annotations of entities and their relations in videos. Conveniently, VidOR and NextQA share the same video source, YFCC100M (Thomee et al., 2016), allowing us to align entity counts with our models' performance on NextQA.

Figure 7 compares the difference in the average number of entities (subjects and objects) in videos where models perform poorly on low complexity tasks (easier questions) versus where they perform well on high complexity tasks (harder questions). The trend indicates that, for easier questions (low complexity), videos with more subjects/objects tend to result in poorer model performance, while for harder questions (high complexity), models perform better when fewer subjects/objects are present in the video.

This pattern supports the idea that the complexity of a VideoQA task is additive. Simple questions become more challenging when accompanied by complex videos, and difficult questions can be made easier in the presence of simpler videos. This additive relationship between video complexity and question complexity reinforces the importance of estimating both components. While our

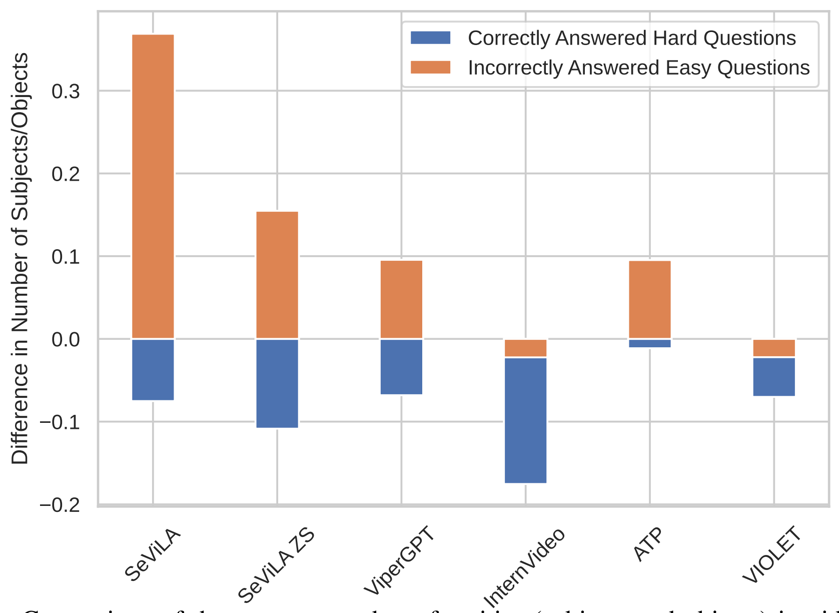


Figure 7: Comparison of the average number of entities (subjects and objects) in videos where models perform poorly on low complexity tasks (easier questions) versus where they perform well on high complexity tasks (harder questions). More entities make easy questions harder, while fewer entities make hard questions easier.

current approach focuses on question complexity, combining it with accurate video complexity estimations can yield a more precise measure of the overall task complexity. We leave the exploration of more accurate methods for video complexity estimation to future work.

9.2 Subtrees Visualization

In this section, we present visualizations of subtrees which correlate with question that are challenging to answer for all 3 models analyzed in the main paper (see Sections 3.3 and 4.3). A majority of the nodes present in the ASTs encode non-essential information such as variable names, while we care about the actual structure and the operations being executed on the frames. For this reason, we ignore variable names and values when comparing two subtrees to one another. Similarly, we develop a tool to visualize the general structure of subtrees that performs a related node-trimming step. Finally, the visualizations of ASTs in the paper (*eg*. Figure 4 of the main paper) include an additional simplification step in which nodes are merged to aid in understanding and interpretability.

All the 8 subtrees that are shared by the 3 models in the main paper are shown in Figure 16. There are two principal patterns that can bee seen from analyzing them. The first group includes primitives that allow for temporal reasoning (Figures 8 to 13). The other common pattern group includes questions that require more detailed analysis of specific elements (objects, relationships) within a scene (Figures 14 and 15).

First, we consider the primitives necessary for temporal reasoning, *i.e.* for questions that necessitate taking into account a specific frame's placement in a sequence of events. The subtrees shown in Figures 8 and 9 both contain the control flow necessary for identifying an event that happens after a particular condition has been met. Figure 8 in particular illustrates a common pattern for finding the frame *after* something has happened, with a *For Loop* that identifies the relevant part of the video, followed by an addition to look *after*. Similarly, Figure 9 is common in programs that have to identify a second condition that happens after a first one. For this reason the control-flow is slightly more complex, and includes a second conditional that is only checked for after the first condition has been satisfied.

Figures 10 to 13, while also temporal, are less obviously so. The presence of the primitives shown in Figure 10 correlates with code that includes a loop over frames in the video. This pattern is commonly used to setup for iterating over the video until a relevant frame is found. Upon identification,

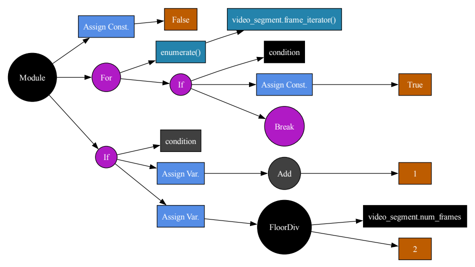


Figure 8:

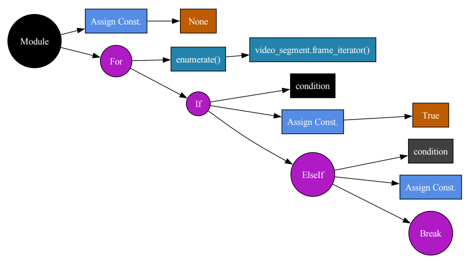


Figure 9:

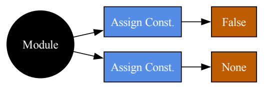


Figure 10:

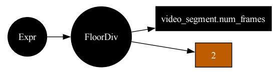


Figure 11:

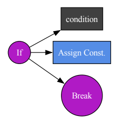


Figure 12:

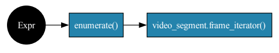


Figure 13:

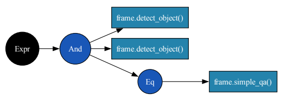


Figure 14:

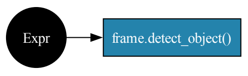


Figure 15:

Figure 16: Subtrees identified as hard.

the first boolean variable switches to *True* and the other one is used to store the frame. Intuitively, this pattern is useful for answering questions about a specific moment of a video. The primitives in Figure 11 show code that selects a frame from the middle of the video. In practice, programs include this code as a fail-safe when searching for a specific frame, falling back to selecting the middle frame in case no satisfying frame is found (*eg.* Fig. 8). Figure 12 shows a *break* statement that will halt an iteration over the video when a frame that meets the required criteria is seen. Intuitively, this pattern allows for the identification of the first event in the video that meets some criteria, as the break in the loop avoids overwriting the variable with frames that come in the future. And Figure 13 shows an iterator over the video, which is the main primitive necessary to consider frames in order. This primitive is often used in conjunction with others shown, *eg.* with the *break* statement in Fig. 12 to find the first frame that meets the condition.

The other common pattern group involves questions that require a more granular consideration of specific elements (objects, relationships) within a scene. For example, the subtree shown in Figure 15 is included in questions that require focusing on a single specific object or actor in a frame of the video. Relatedly, programs that include the subtree in Figure 14 require identifying at least two objects or actors and then relating them (by calling *simple_qa()*, an image question answering module).

9.3 Additional Analysis Results: Temporal Support

The interpretable nature of subtrees allows us to manually identify a subset of subtrees we know correspond to subroutines that store frames. We leverage Equation 1 to count the appearance of said subtrees in each program's ASTs to find the temporal support of each question. As previous works have proposed (Mangalam et al., 2023), we validate that a significant source of question complexity in Video Question Answering is owed to the number of frames needed to answer the question (Figure 17).

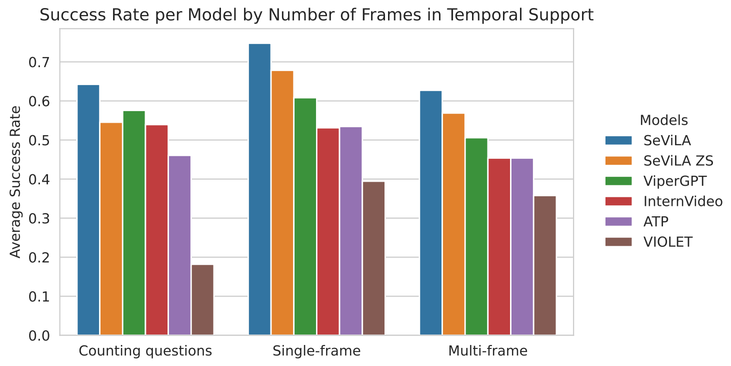


Figure 17: Temporal support (*i.e.* number of frames a question needs) according to the generated program. All models tested perform significantly worse on questions that require more frames. Counting questions are listed separately, as they potentially require every frame to be checked.

	#Questions	# Videos
Action Genome (Ji et al., 2020)	1572	1154
ActivityNet (Caba Heilbron et al., 2015)	749	594
MOMA (Luo et al., 2021; 2022)	133	93

Table 3: Composition of CodePlex-QA in terms of number of questions and videos from each source dataset.

9.4 Dataset Statistics: CodePlex-QA

The outcome of the data generation process is summarized in Table 3, which presents a breakdown of CodePlex-QA. For each source dataset, the table shows the number of questions and videos that pass the described CodePlexity filter. Importantly, a majority of the questions are associated with a unique video.

10 Ablations

10.1 Validating Question Selection Algorithm

We now validate our approach for selecting hard questions based on our complexity estimation using the NExT-QA validation set. In particular, we follow the same approach (and with the same threshold and parameters) as when constructing CodePlex-QA, and select the most challenging questions from this set. We then evaluate the same models from Section 5.2 on this subset, which we call NExT-QA*, and compare to both the original NExT-QA dataset, and CodePlex-QA. This allows to separate the effects of question generation and question filtering when constructing CodePlex-QA as NExT-QA and NExT-QA* share exactly the same base set of questions.

Figure 18 indeed validates that our approach is successful in identifying a subset of NExT-QA that is more challenging for all models evaluated compared to the original dataset. However, our final dataset generation pipeline results in an even more challenging benchmark by first constructing a more diverse pool of videos and questions for the filtering approach to select from.

10.2 Impact of Code Generation Correctness on Complexity Metrics

To investigate the relationship between code generation correctness and the alignment of complexity metrics to problem structure, we conduct an ablation study comparing cases where the generated code produces correct answers with those where it does not. Note that ”incorrect answers” do not necessarily imply that the code itself is invalid; rather, it may fail to produce the expected output.

The results are summarized in Table 4, which presents the correlation between the complexity metrics and the mPEG metric for various models. From Table 4, it is evident that the correlation between

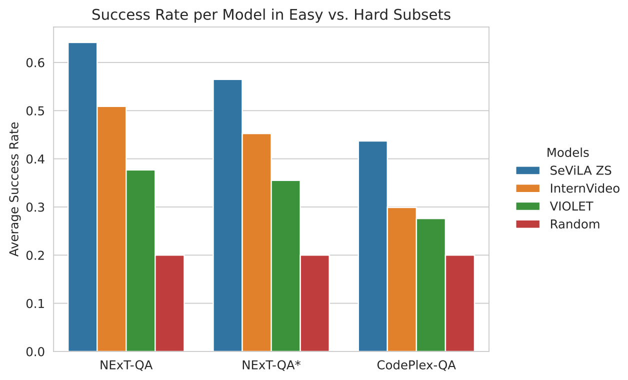


Figure 18: Filtering NExT-QA using our approach indeed results in a more challenging subset for all the evaluated models. However, our full dataset construction pipeline results in an even more challenging benchmark by first generating a more diverse pool of samples to select from.

	Train Models				Validation Models			
	SeViLA	ViperGPT	ATP	VIOLET	HGA	SeViLA ZS	InternVideo	Tarsier
Lines of Code								
Correct	0.1373	—	0.1747	0.1455	0.0712	0.1654	0.2022	0.1475
Incorrect	0.1245	—	0.0540	0.0656	0.0735	0.0756	0.0831	0.0696
Cyclomatic Complexity								
Correct	0.1702	—	0.2128	0.1930	0.0649	0.1739	0.2825	0.1634
Incorrect	0.1351	—	0.1118	0.0881	0.0664	0.0973	0.1388	0.1071
CodePlexity								
Correct	0.2608	—	0.3128	0.3178	0.0867	0.2095	0.2877	0.1950
Incorrect	0.2810	—	0.2041	0.2542	0.1087	0.1839	0.1700	0.1857

Table 4: Correlation of complexity metrics with mPEG for cases where the generated code produces correct and incorrect answers.

the mPEG metric and the various complexity metrics is consistently higher for cases where the generated code produces correct answers.

These results suggest that correct code generation often aligns better with problem complexity, as reflected in higher correlations with the mPEG metric. By contrast, incorrect code, while potentially valid in syntax or structure, often fails to capture the underlying complexity of the problem, thereby diluting the relationship between the metrics.

10.3 Impact of Code Generation Model Choice

This section evaluates the influence of the code generation model on the performance of our complexity estimation framework. Specifically, we compare ViperGPT, the primary model used in our analysis, with Recursive Visual Programming (RVP) Ge et al. (2023), a newer model designed for visual programming tasks. Unlike traditional approaches, RVP employs a recursive code generation strategy, which systematically breaks down complex problems into manageable subproblems. This allows it to handle intricate question structures with greater flexibility.

Figure 19 illustrates the relationship between the estimated complexity of questions and the performance of Visual Programming models. For both ViperGPT and RVP, we observe a significant negative correlation between the complexity metric and the model's success rate. This trend highlights that as the estimated complexity of a question increases, the likelihood of the model correctly addressing it decreases. This correlation underscores the utility of the complexity metric as a predictive tool for identifying challenging questions.

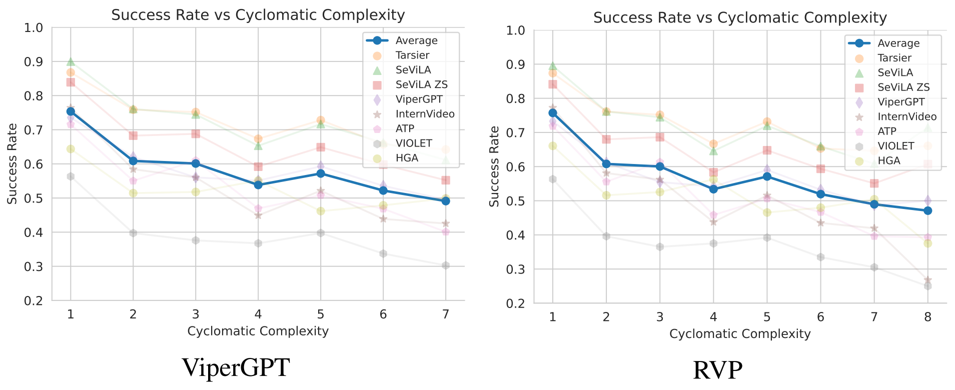


Figure 19: Comparison of code-based complexity metrics when using different code generation models. Lines in both cases show significant negative correlation of complexity metric with model performance for both variants.

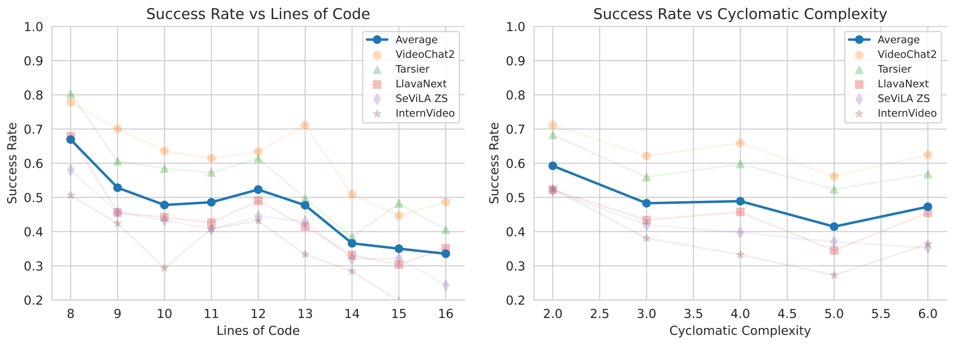


Figure 20: Correlation of VideoQA models' success rate on MVBench for various approaches for estimating question complexity. As was the case for NExT-QA, we observe that code complexity correlated strongly with question complexity.

10.4 Impact of Analysis Dataset Choice

To assess the generality and robustness of our findings, we replicated our analysis using a different dataset, MVBench (Li et al., 2024), which offers a diverse set of videos and questions compared to NExT-QA. Specifically, we repeat the same experimental setup as that in Section 4.1, first generating programs for the questions in MVBench, and then rerunning our pipeline to extract code based metrics and train our CodePlexity metric. Furthermore, we additionally consider two new models in our analysis: VideoChat2 (Li et al., 2024) and Llava-NExT (Liu et al., 2024) as these represent the state of the art on the MVBench dataset.

We first visualize the correlation between code-based complexity metrics and the performance of various VideoQA models on MVBench in Figure 20. Consistent with our observations on NExT-QA, we found that code-based complexity metrics exhibit a strong negative correlation with model performance on MVBench. Specifically, both Lines of Code and Cyclomatic Complexity continued to demonstrate a consistent and strong correlation, indicating that questions requiring more intricate code are more challenging for all the models evaluated.

We further conducted a systematic evaluation of different code-based metrics using the mPEG metric on the validation set of MVBench, summarized in Table 5. Our proposed CodePlexity metric significantly outperformed the naive code complexity measures, such as Lines of Code and Cyclomatic Complexity. CodePlexity achieved higher predictive accuracy in estimating question difficulty across all evaluated models on MVBench. Note that CodePlexity generalizes to the held-out models in our analysis (VideoChat2 (Li et al., 2024) and Llava-NExT (Liu et al., 2024)).

	Train Models			Val. Models	
	InternVideo	SeViLA ZS	Tarsier	VideoChat2	LLaVA-NeXT
Lines of Code	0.0441	0.0951	0.1254	0.1027	0.0919
Cyclomatic Complexity	0.1299	0.0964	0.0614	0.0521	0.0447
CodePlexity (Ours)	0.4134	0.3303	0.4444	**0.2991**	**0.2743**

Table 5: Comparison of question complexity metrics using mPEG on the validation set of MVBench. CodePlexity is trained on the first three models. Our approach demonstrates the highest correlation with the models' performance.

These consistent results across two distinct datasets suggest that code-based complexity metrics, and CodePlexity in particular, are effective tools for assessing question difficulty in VideoQA tasks regardless of the dataset's characteristics.

11 Broader Impacts and Limitations

In our study, we utilize four distinct pre-trained models, each with its inherent biases, to identify challenging questions within an existing dataset. Although they have different pre-training schemes, these models likely encode similar implicit biases, owning to their training on internet scale data collections. In particular, three of our selected models, along with our visual descriptors extractor, rely on CLIP (Radford et al., 2021) as a visual encoder, meaning they likely replicate the same biases including those identified in previous studies (Agarwal et al., 2021).

Furthermore, the dataset we base the majority of our analysis on, NExT-QA, is not fully representative of real-world diversity and complexity. This limitation in addition to the biases present in the chosen models can lead to skewed or incomplete analysis. Furthermore, our analysis' focus on interplay between the constituent syntactic elements in code may overlooks critical sources of complexity not apparent in the code structure. These include, for example, differences related to gender and ethnicity, which are not explicitly manifested in the code.

Similarly, our own proposed dataset, CodePlex-QA, builds upon the existing datasets MOMA (Luo et al., 2021; 2022), ActivityNet (Caba Heilbron et al., 2015), and Action Genome (Ji et al., 2020), and therefore includes the same biases. We urge researchers and practitioners to refer to the relevant dataset cards. Finally, our selection methodology for filtering videos and questions may inadvertently introduce new biases, or amplify existing ones.