Video Question Answering with Procedural Programs

Rohan Choudhury[1], Koichiro Niinuma[2], Kris M. Kitani[1], and László A. Jeni[1]

[1] Carnegie Mellon University
[2] Fujitsu Research of America
{rchoudhu, kmkitani}@andrew.cmu.edu kniinuma@fujitsu.com
laszlojeni@cmu.edu

Abstract. We propose to answer questions about videos by generating short procedural programs that solve visual subtasks to obtain a final answer. We present **Pro**cedural **Vi**deo **Q**uerying (ProViQ), which uses a large language model to generate such programs from an input question and an API of visual modules in the prompt, then executes them to obtain the output. Recent similar procedural approaches have proven successful for image question answering, but cannot effectively or efficiently answer questions about videos due to their image-centric modules and lack of temporal reasoning ability. We address this by providing ProViQ with novel modules intended for video understanding, allowing it to generalize to a wide variety of videos with no additional training. As a result, ProViQ can efficiently find relevant moments in long videos, do causal and temporal reasoning, and summarize videos over long time horizons in order to answer complex questions. This code generation framework additionally enables ProViQ to perform other video tasks beyond question answering, such as multi-object tracking or basic video editing. ProViQ achieves state-of-the-art results on a diverse range of benchmarks, with improvements of up to 25% on short, long, open-ended, multiple-choice and multimodal video question-answering datasets. Our project page is at `https://rccchoudhury.github.io/proviq2023/`.

1 Introduction

Consider the video in Figure 1. What color jacket did the skier in orange pants wear? To answer this question, one would look for a skier in each frame, search those frames for a skier wearing orange pants, then check what color jacket they were wearing. Like this example, humans tend to solve questions *procedurally*, breaking problems down into a sequence of steps, each with concrete results. We hypothesize that using this type of reasoning can significantly improve performance for video question answering (QA).

Existing video QA methods do not follow this approach. The predominant paradigm for video understanding is to train a supervised end-to-end model, typically by pre-training on large video datasets such as Kinetics [6] or Ego4D

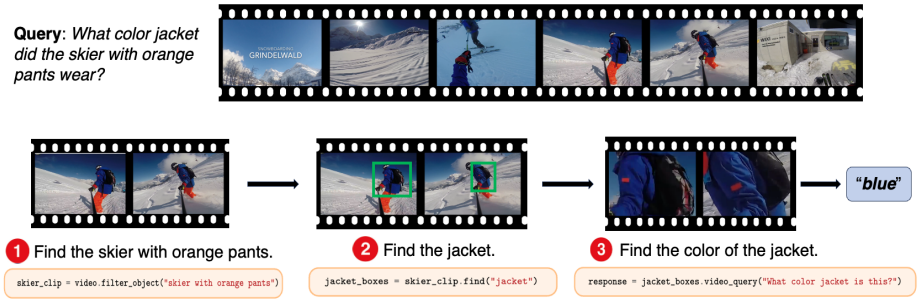

Fig. 1: Our method, **ProViQ**, reasons procedurally about videos by generating and executing Python programs that solve visual subtasks, mimicking how humans might approach such problems.

[12], then fine-tuning on relatively smaller QA benchmarks. Some zero-shot and few-shot QA methods such as FrozenBilM [55] or SeViLa [60] combine pre-trained video backbones with language models to some success, but are still unable to explicitly carry out procedural reasoning. Recent large multimodal language models like GPT-4V [1] and Gemini [47] exhibit very strong performance, but are closed-source, expensive to run and not easily intepretable.

On the other hand, recent works like ViperGPT [46] and VISPROG [13] used large language models (LLMs) to generate short programs for this exact type of reasoning, achieving strong results for compositional image tasks. Given an API of modules that solve visual subtasks, these methods generate programs that call these modules and execute the program to obtain a final answer. This requires no training, as LLMs can generate working, correct code and the provided modules rely on models pre-trained on large image datasets.

Although these works show strong results on images, they do not transfer well to video. Because they were designed to reason about images, they have no clear way to aggregate information over several frames, to efficiently find information in a long video, or to understand the high-level narrative; put simply, their modules are not capable of reasoning about video. For example, ViperGPT needs to iterate through each frame in a video to satisfy individual predicates in its generated code, and can only consider a single frame for any piece of reasoning. This is prohibitively slow as well as fragile to noisy predictions from underlying modules. Furthermore, replacing these underlying large pretrained image models with analogous video models is not straightforward. Due to the large computational burden and relative lack of training data, significantly fewer high-quality video models are available, and those that do, do not offer the same level of performance.

We introduce **Pro**cedural **Vi**deo **Q**uerying (ProViQ), which addresses these shortcomings. ProViQ provides API of *video-centric* modules, an input query, and relevant in-context examples to an LLM. The LLM then generates a Python program that uses the video modules, and executes it to obtain a final answer to the query. While on their own, the individual video modules are not able to directly answer queries, ProViQ compensates by programmatically combining

them to solve subtasks for question answering. In particular, we include modules to enable video retrieval, captioning, summarization, and multi-object tracking, as well as image-based tasks like object detection, segmentation and captioning. These modules enable frame-level, clip-level, and video-level reasoning, allowing ProViQ to efficiently and effectively answer questions about videos.

ProViQ's procedural reasoning over videos has several benefits. Firstly, like prior visual programming works, ProViQ requires no further training. This allows smooth incorporation of task-specific modules, making it simple to add capabilities for solving new types of questions without fine tuning. Secondly, the program's reasoning is interpretable: each line in the program can help attribute errors to specific modules. Thirdly, the LLM can compose the modules freely, enabling capabilities beyond question-answering. We demonstrate that combining the object detector and tracker modules yields a query-based multi-object tracking system, and the retrieval module can be used for basic video editing, just through generating a few lines of code.

ProViQ leverages these advantages to significantly improve on a wide range of zero-shot video question answering benchmarks: we improve up to 25% on ActivityNet [61], iVQA [54], MSR-VTT-QA [53], MSVD-QA [53], TGIF-QA [17] and NeXT-QA [52], *without any additional training*, even surpassing the *supervised* state-of-the-art on some datasets. We also demonstrate strong performance on understanding long egocentric videos with a gain of 25% on the challenging EgoSchema benchmark [33], as well as multimodal understanding, obtaining state-of-the-art performance on the TVQA dataset [21].

In summary, our contributions are that

1. We present ProViQ , a method that successfully extends visual programming to video QA through the use of novel video tools.
2. Using these novel modules, we achieve large accuracy improvements on a wide range of video QA benchmarks, setting the state-of-the-art on open-ended, multiple choice, and long video tasks, demonstrated by experiment and comprehensive ablations.
3. ProViQ is capable of additional tasks beyond question answering, such as multi-object tracking or basic video editing.

2 Related Work

2.1 Video Question Answering

Compared to other video tasks, video question-answering datasets are relatively small. As in image QA, the predominant paradigm for video QA is to pre-train models on large datasets like Kinetics [6], Ego4D [12], HowTo100M [34], or YouTube-100M [62], then fine-tune on smaller annotated QA datasets [8–10, 16, 19, 20, 28, 37, 43, 58]. Recent work like InternVideo [49], InternVid [48] and mPLUG-Owl [59] scale up this type of training significantly, but still do not generalize well enough to answer questions about videos outside their training

distribution zero-shot. Recently, massive multimodal language models like GPT-4V [1] and Gemini 1.5 [47] have been found to perform well on zero-shot video QA, but are extremely expensive to train and run.

Comparatively few works directly address zero-shot video QA. BLIP [24] trains a large model on image-question-answer triplets and evaluates the transfer to video QA tasks, but includes a fine-tuning step. Current methods are typically trained on web-scale datasets with audio or speech transcripts providing weak language supervision [54, 55, 62, 63]. In particular, FrozenBiLM [55] connects a frozen bidirectional language encoder with a trainable video model, trains on WebVid10M [4] and measures zero-shot question-answering performance. More recently, SeViLa [60] fine-tunes BLIP to identify video frames relevant for an input query and generate an answer. These methods currently obtain state-of-the-art results, and we compare against them in Section 4.2. Another recent line of work [25, 27, 32, 44] use combinations of language models and visual inputs such as textual descriptions or CLIP [38] features from sparsely sampled frames to enable conversations about videos, but do not achieve strong quantitative results on standard benchmarks.

2.2 Modular Vision

Neural Modular Networks (NMNs) [3] introduced modular visual question answering approaches, using parsers to compose learned modules into single trainable network. Follow-up methods to NMNs jointly trained the layout generator and the visual modules with reinforcement learning and weak supervision [14, 15, 41]. Several other works train large models that contain modules for different modalities and tasks [11, 40, 50, 64], but cannot freely compose them or alter their layout.

In the past year, CodeVQA [45], VISPROG [13] and ViperGPT [46] leveraged the large improvements in language modeling to reformulate modular VQA as code generation: they use the strong performance of GPT-3 [5] and GPT-4 [36] on code generation to formulate answers to visual questions as short Python programs, enabling use of mathematical operations, if-statements, and logical operators to manipulate the outputs of visual models. ViperGPT provides strong results on the NeXT-QA dataset [52], but cites the length and inability to temporally reason as limiting factors for further experiments. We directly build off ViperGPT's approach with the same program generation framework and successfully extend it to video QA.

2.3 Prompting and Tool Use

With the recent surge of interest in large language models, many papers have studied how to effectively incorporate additional tools, either through fine-tuning [42] or prompting [25,51,56,57]. Following the success of large multimodal models like Flamingo [2] and GPT-4 [36], recent methods train multimodal language models, such as LLAVA [29, 30], or MiniGPT-4 [68] to add visual capabilities to

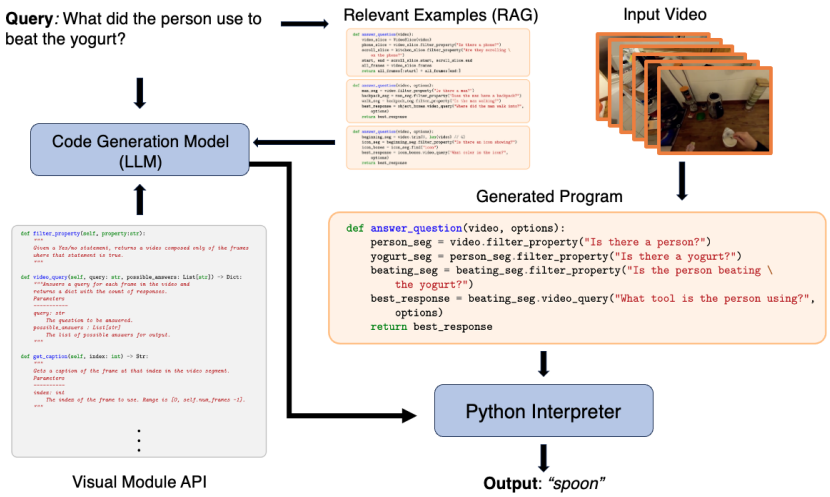

Fig. 2: Our method. ProViQ provides the input question, visual API, and relevant examples in the prompt to the code generation model. The LLM generates a short Python program that uses modules from the API to answer the question. Using the video as input, we execute the program to obtain the final output.

language models. However, these have not successfully incorporated videos, due to the challenges of training on large-scale video data.

Following CodeVQA, VISPROG and ViperGPT, we use several pre-trained models for visual functions like object detection and image QA. We use GroundingDINO [31] for text-conditioned object detection, BLIP-2 [23] for image captioning and QA, and ByteTrack [66] for multi-object tracking. We also use LaViLa [67] for video-to-language generation and use GPT3.5 [5] for generating code and querying summaries. Concurrent work [65] proposes a similar language-based strategy to ours for summarizing long videos, and we encourage readers to review it for a more complete view of the field.

3 Method

3.1 Program Generation

The input to ProViQ is an input video and a query. We then construct a prompt containing a list of video modules in an application programming interface (API), the input query or task, and a few example programs, which is then fed to the LLM. We used `gpt-3.5-turbo` for all experiments in the main paper, but include evaluation with open-source alternatives in Appendix C. The LLM produces a short Python program that decomposes the input query into concrete steps, each calling visual modules specified by the API. The generated program takes as input the question, the video frames, and other relevant information such as a list of multiple choice options. Once the output is generated, we compile and execute the program as in [46] using Python's built-in `exec()` function. The

compiled function runs on the input video to output the final answer or modified video as specified by the task.

3.2 Video Modules

The list of video modules in the provided API are intended as a toolbox for the generated programs to use for decomposing and answering questions. We used these specific methods in order to encompass a wide variety of datasets and possible questions, with room for the LLM to improvise and combine methods together as it sees fit. While this API is not necessarily exhaustive, we found it sufficient to answer the vast majority of questions in QA benchmarks. We provide this list of modules to the LLM in the prompt in the form of methods with documentation, which we include in Appendix E. While all modules are intended to run on collections of frames, they work on single images as well, enabling reasoning about singular frames as well as clips. Specifically, they include:

`filter_property` Given a boolean predicate such as *"Is the person running?"*, this method finds all frames in the video that satisfy the predicate. It works by calling BLIP-2 on an input batch of frames with the question and collecting all frames where the answer is `yes`.

`filter_object` Given a specific object, such as *yogurt*, this method finds all frames in the video where the object exists.This method runs an object detector over the input clip and returns all the frames where the given object is found. While `filter_property` can handle this functionality, using an object detector works much better for finding specific objects.

`find` This method calls a text-conditioned object detector to return each crop of the input object found in the collection of frames. This method is useful for zooming in to the input object over time, which can improve performance on visual queries about it.

`track_objects` Given a set of detections over continuous frames, associates them together using ByteTrack [66] and returns each tracked object in the scene.

`video_query` This method invokes BLIP-2 [23]'s QA capability to answer a question about collections of frames: for example, *"what is the person doing?"* It computes the answer to the input question for each frame in the video and collates them. It then uses frame-wise voting to choose the most salient answer. We used voting over choosing a single frame's value in order to reduce potential for errors from a single frame.

`get_summary` Given a collection of frames, this method uses a video captioning model on discrete slices of the video and produces an overarching paragraph summary of the events in the video. We elaborate more on this capability in Section 3.3. This module is useful for answering questions about the high-level narrative structure of a video, especially for longer clips such as those in Ego4D.

`get_script` This computes a transcript of the audio from the input video using Whisper [39], or returns the transcript if one already exists. This method is particularly useful for benchmarks like TVQA, where obtaining the character's dialogue is essential for solving questions.

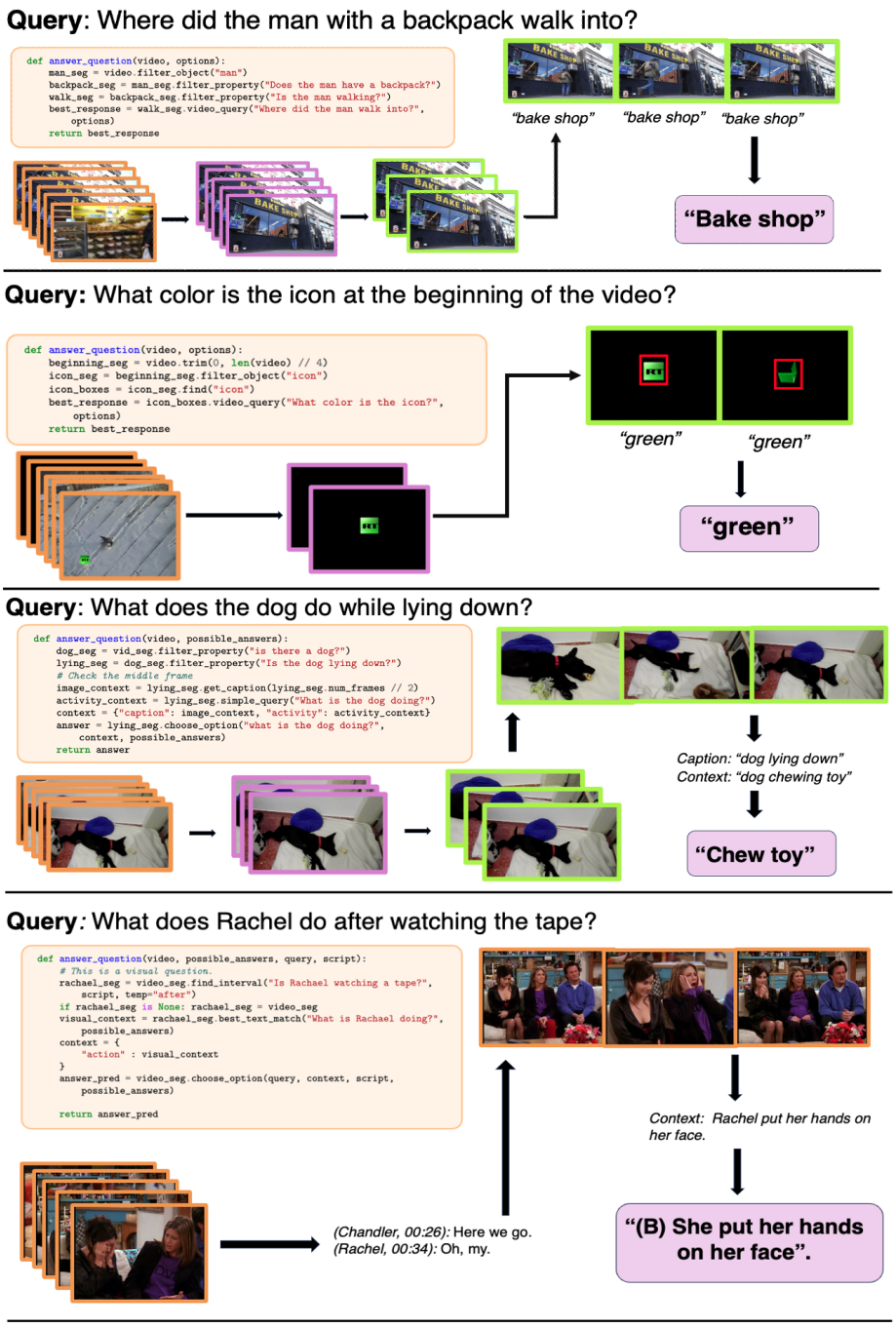

Fig. 3: Qualitative samples from ProViQ. Provided an input video and question, ProViQ can effectively decompose a query into steps, translate them into function calls, and execute them to obtain a final answer. Since our results are on video, we urge readers to view our video results on our project page

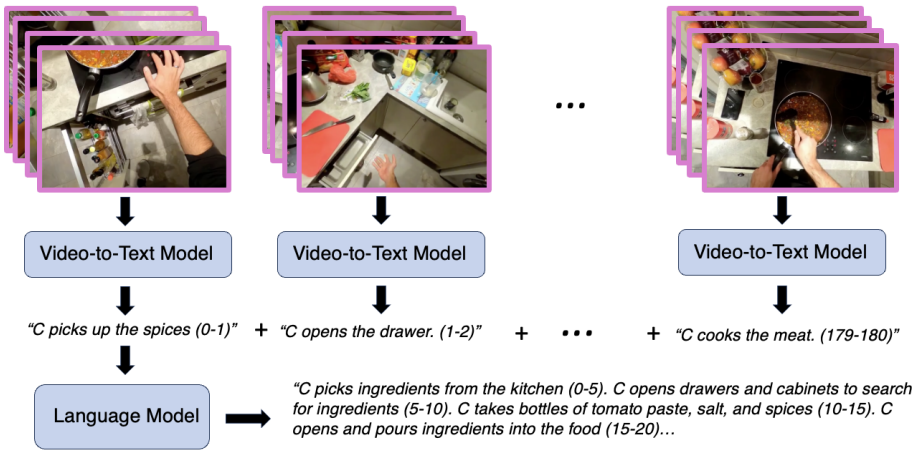

Fig. 4: Our long video summarization module. In order to answer questions about video narratives, our summarization module uses an LLM to fuse the outputs of video-to-text models on subclips into a coherent paragraph summary. This method produces qualitatively accurate summaries, but depends on a high-quality video-to-text model that captures the action in short video clips. We used LaViLa [67], which is meant to be used on the Ego4D dataset.

`get_caption` Computes captions for the input image or set of frames. This is useful for giving visual context of the entire scene, which can help guide choosing multiple choice answers on datasets like TVQA or NeXT-QA.

`best_text_match` Given a video segment and a set of choices, returns the option that best matches the content of the video segment. This is done by querying the video segment whether each individual choice is true or false, and return the option with the highest score. We compute the score of an option by averaging the confidence of the video querying module over the segment. This module is particularly useful for multiple-choice questions, as it can help simulate process-of-elimination reasoning.

`choose_option` Given input context, a question and options, uses an LLM to answer a multiple choice question by choosing the most relevant answer. The input context can be visual, such as a caption or visual outputs from other modules, or textual, such as a narrative summary or transcript. This method is crucial for solving multiple choice benchmarks, and for reasoning about input context from `get_script` or `get_summary`.

3.3 Long Video Summarization

One advantage of our modular approach is that we can define different modules that are better adapted to certain tasks. We use this to address long-video question answering, a task that has remained out of reach for all but the largest models [47, 49]. Consider a question that requires understanding higher-level semantics, such as *"Which option best describes the overarching narrative of the video?"*. A human solving this would construct a mental narrative of

the video, then match it to the list of given options. We implement a module `get_summary()` that leverages pretrained video-to-text models to understand the high-level story of a video, which we illustrate in Figure 4. This works on videos of any length, allowing it to summarize 5 minute videos, much longer than any existing model except Google's recent Gemini 1.5 Ultra [47]. Given a long video V and pre-trained model M that accepts a contiguous segment of frames and outputs a caption, such as LaViLa [67], we partition V into equal-sized chunks of 1 second. For each chunk, we run M, outputting a caption and the timestamp of the chunk, resulting in a list of timestep annotated captions. We then aggregate this caption stream into a paragraph summary with an LLM which we use as the "narrative" of the video, with each sentence describing a 5-second interval. Importantly, this module relies on a high-quality video captioning model. This approach demonstrates strong results on the EgoSchema dataset due to the high density of annotated narrations in the underlying Ego4D dataset, enabling high-quality models like LaViLa to be trained. For other datasets, this approach is less effective due to the lack of comparable quality video captioning models.

3.4 Prompting and In-Context Examples

A long line of work [22, 51] has demonstrated that the wording of the input prompt and set of examples used greatly influence LLM performance for downstream tasks While ViperGPT includes a fixed set of example programs in the API and eight in-context examples, we follow VISPROG and CodeVQA and annotate up to five example programs for each benchmark dataset. At inference time, we sample the three examples whose corresponding queries are closest in embedding space to the input query. We use OpenAI's `text-embedding-3-small` to embed all queries. Using these in-context examples results in significantly higher quality generated programs. For individual modules that use language models, such as `choose_option` or `get_summary`, we keep their prompts fixed, using a static set of example inputs. All example queries and answers are sampled from training splits of benchmark datasets to avoid contamination in downstream evaluation. We provide detailed ablations on prompt components in Section 4.3, and more details on in-context examples and the prompt in Appendix D.

3.5 Open-Ended and Multiple-Choice Benchmarks

The output answer from the program needs to be constrained to a fixed vocabulary to evaluate correctness with benchmarks. In multiple-choice QA datasets, this can be accomplished by prompting the LLM in the '`choose_option()`' module to constrain its output to the range of input answers. On the other hand, open-ended benchmarks are typically formulated as K-way classification problems with K ranging into the thousands. To address this, we select the semantically closest vocabulary answer to the string produced by the generated program. Concretely, we embed the output string from `video_query` with a pre-trained phrase embedding model and find the closest match in embedding space from the output vocabulary. We used FastText [18] for this step, and abstracted

	TGIF	MSVD	MSRVTT	ANet	iVQA	NeXT	TVQA	EgoSchema
Random	0.1	0.1	0.1	0.1	0.1	20	20	20
CLIP-VIT-L/14 [38]	3.6	7.2	2.1	1.2	9.2		26.1	-
Just Ask [54]	-	13.3	5.6	12.3	13.3	-	-	-
FrozenBiLM [55]	41.9	33.8	16.9	25.9	26.8	-	59.7	26.1
SeViLa [60]	-	-	-	-	-	63.6	38.2	-
ViperGPT (*3-shot*) [46]	51.2	25.3	17.2	30.6	41.4	60.0	-	-
Supervised SOTA	66.3	54.8	47.0	43.2	40.9	63.1	86.1	32.1
ProViQ (*3-shot*)	**66.1**	**37.5**	**22.1**	**42.3**	**50.7**	**63.8**	**66.3**	**57.1**

Table 1: Comparison of ProViQ to zero-shot video QA benchmarks. Following [46], we compare to other end-to-end zero-shot methods. ProViQ achieves state-of-the-art performance by a wide margin, improving accuracy by up to 26% on both open-ended and multiple-choice benchmarks and is even competitive with supervised methods.

away this logic from the program execution to reduce the calls required for the generated program. Furthermore, several open-ended benchmarks classify their questions into disjoint categories, such as "locations" or "objects". As the question type is available at test time, we use these splits to constrain the vocabulary for each category as well. We ablate these components in Appendix C.

4 Experiments

This section demonstrates the effectiveness of ProViQ and compares to the current state of the art. We compare to other methods inSection 4.2, and present ablation studies and error analysis in Section 4.3. Finally, we present qualitative examples of ProViQ's capabilities beyond question answering in Section 4.4

4.1 Experimental Setup

We evaluate ProViQ on a wide variety of datasets, containing short, long and egocentric videos with visual, narrative and multimodal questions. We briefly describe the benchmarks used here, and provide further details in Appendix B. Unless otherwise stated, we evaluate on the full test split of each dataset. We evaluate ProViQ on open-ended VideoQA (iVQA [54], TGIF-QA FrameQA [26], MSRVTT-QA [53], MSVD-QA [53] and ActivityNet-QA [61]) and multiple-choice VideoQA (TVQA [21], NeXT-QA [52]) and long-video understanding(EgoSchema [33]). Furthermore, none of the underlying models for any module are trained on the benchmarks used for evaluation. On all datasets, we measure performance with top-1 test accuracy for fair comparison with prior work. Some recent methods [25, 32, 44] use a different metric based on language model evaluation; we compare to those in Appendix B. All videos are sampled at 1 FPS and native resolution.

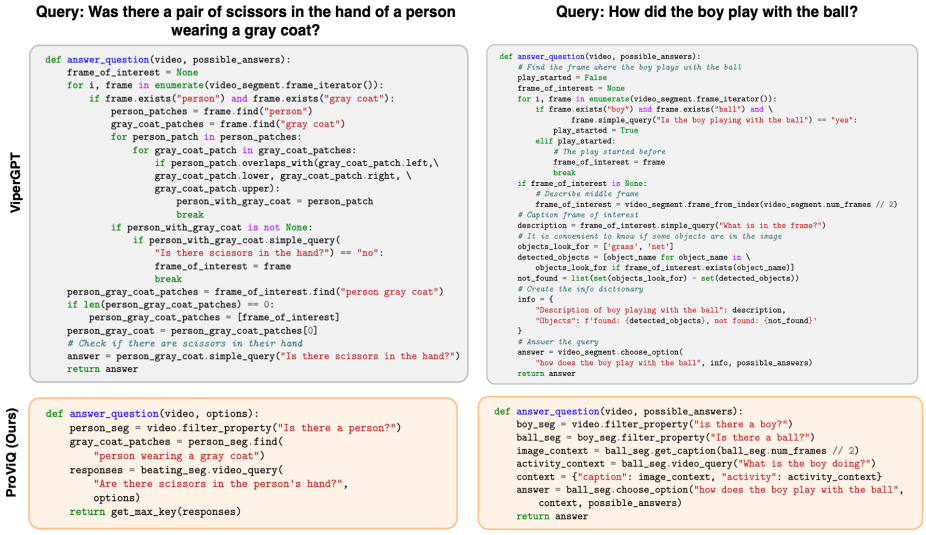

Fig. 5: Qualitative comparisons on ProViQ's generated code to ViperGPT. Thanks to its dedicated video modules, ProViQ generates code that is more concise and less error-prone, enabling significantly higher performance across standard benchmarks.

4.2 Video QA Results

Our main results are contained in Table 1 . We achieve large accuracy improvements on both open-ended and multiple choice benchmarks, with particularly large gains of 9% on iVQA, 15% on TGIF-QA and 25% on the challenging EgoSchema benchmark, as well as state-of-the-art performance on every other benchmark. Surprisingly, we surpass the *supervised* SOTA on iVQA, NeXT-QA and and are competitive on TGIF and ActivityNet. We attribute these improvements to a few different factors.

Firstly, on open-ended benchmarks, such as TGIF, ActivityNet, and iVQA, we observe much larger improvements as ProViQ can effectively find the relevant video segments through its generated program, then use strong vision-language models only on n that segment to compute the answer. In contrast, end-to-end methods like Just Ask and FrozenBiLM consider all frames and are less able to focus on the informative segments of the video. Secondly, we also outperform ViperGPT, which uses an image-centric API and is unable to reason over longer intervals of time. In comparison, ProViQ generates code that is more concise and robust, demonstrated in Figure 5. We also observe that performance improvements were highly correlated with dataset label quality. MSR-VTT and MSVD have a large fraction of ambiguously or incorrectly labeled questions, while iVQA, TGIF-QA and ActivityNet-QA have higher quality labels. In particular, iVQA contains multiple correct answers per question, making the vocabulary-matching much more forgiving. On multiple-choice datasets, such as TVQA and NeXT-QA, the improvement is less due to the constraints imposed by having much fewer choices; on TVQA in particular, most questions are dominated by the lan-

IQ	VQ	FP	find	caption	full	TGIF	MSVD	MSRVTT	ActivityNet	iVQA	NeXT-QA
✓						62.1	34.1	16.6	27.7	41.5	49.3
	✓					63.4	**37.8**	20.1	35.0	46.5	53.2
	✓	✓				66.1	37.5	24.1	39.1	50.3	55.2
	✓	✓	✓			66.1	37.5	**23.5**	**42.6**	**52.0**	55.9
	✓	✓	✓	✓		66.1	22.8	16.9	42.3	50.7	63.8
	✓	✓	✓	✓	✓	**66.1**	37.5	22.1	42.3	50.7	**64.6**

Table 2: Ablating the visual modules. We successively ablate the performance with single-image querying only (`IQ`), video querying (`VQ`), filtering (`FP`), the `find`, `caption` modules, and then the full prompt. Generally, adding more modules can slightly reduce performance on individual datasets, but enables generalization to a much wider range of benchmarks.

guage model reasoning over the input script rather than visual elements, leading to a smaller but still significant improvement.

4.3 Ablation Studies

Individual Modules. We ablate each module to better understand the observed boosts in performance, shown shown in Table 2. We omit the `get_summary` module from this analysis since it only applies to the EgoSchema benchmark; the other datasets do not have sufficiently performant video-to-text models. Including a image QA module already provides a strong baseline, suggesting that simply prompting a strong image QA model with the right questions can be helpful. Including the `video_query` module, enabling QA over video segments, results in a large boost as it is less prone to errors from querying a single frame; this accounts for most of the improvement in shorter video datasets. The `filter_property` and `filter_object` modules especially help on longer video datasets, such as ActivityNet and NeXT-QA. The `find` module helps answer “counting" and “color" questions in ActivityNet, explaining the observed accuracy increase, and the `get_caption` module adds a significant boost on NeXT-QA, which needs visual descriptions of scenes to answer causal questions. In general, adding more and more modules slightly decreases performance, as the language model can misuse modules or combine them in ways that may not compile. However, this effect is mitigated with in-context examples, and including more modules allows ProViQ to generalize to a wider variety of benchmarks with state-of-the-art performance.

In-context Examples. We next ablate the in-context examples provided in the prompt to understand their effect. In Figure 6, we measure the effect of varying the number of in-context examples on downstream benchmarks. With no examples, performance is poor: the LLM generates programs that may not compile, hallucinates methods or writes overly complex code. Adding a single example program greatly improves performance, and while including more examples is helpful, it provides diminishing returns. We found that using 3-4 examples worked well across benchmarks.

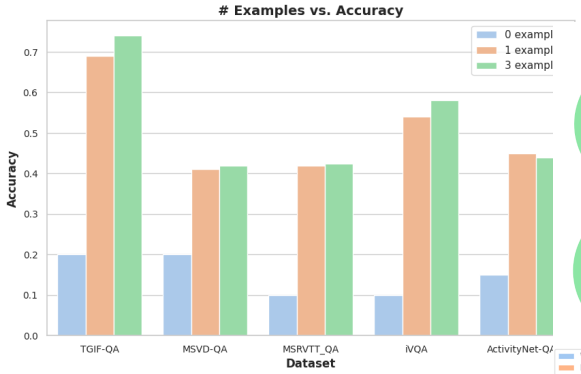

Fig. 6: Impact of in-context examples. ProViQ performs best with in-context examples, but with diminishing returns as more examples are included.

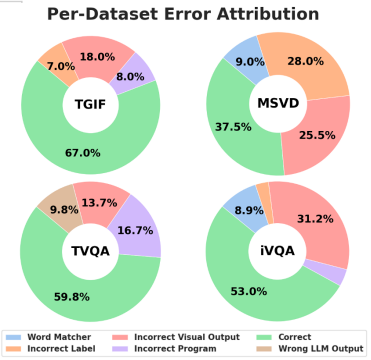

Fig. 7: Error Analysis. On lower-quality datasets, ProViQ suffers from visual module failure and labeling issues, but on harder datasets like TVQA, errs by generating incorrect programs.

Failure Modes. We manually inspected the failure modes of ProViQ, shown in Figure 7, on 100 random samples from each dataset. Due to the interpretability of our method, we can effectively attribute the cause of errors to either an incorrect program, module failure, or incorrect labeling. We conduct this analysis on MSR-VTT, ActivityNet, iVQA and TVQA. On open-ended benchmarks, a considerable portion of the dataset are ambiguously or incorrectly labeled, with ProViQ outputting a correct answer. We find that the balance of program generation vs. module failures depends on the dataset: lower quality and shorter video datasets, such as MSVD and MSR-VTT, mostly suffer from annotation error or mistakes from individual visual modules. Although iVQA has much higher annotation quality, the word-matching embedding model often misclassifies the output from `video_query`, leading to correct output from the program but mistakes in post-processing. On the other hand, ProViQ's mistakes on TVQA, a multimodal dataset, are mostly at the program generation phase, as the language model often uses the the wrong modalities, such as checking the speech transcript to answer visual questions.

4.4 Additional Capabilities

ProViQ can use its modules to perform other video tasks by composing its modules in different ways. We provide qualitative examples in Figure 8 with ProViQ's ability and perform multi-object tracking or video editing. Other tasks are straightforward to implement as well, requiring only a module for the specific functionality. We do not claim to achieve state-of-the-art performance on any of these tasks, but ProViQ is able to accomplish them with no additional training.
Grounded Tracking Since ProViQ can use a text-conditioned object detector, it can combine it with the tracking module to track multiple objects in a scene

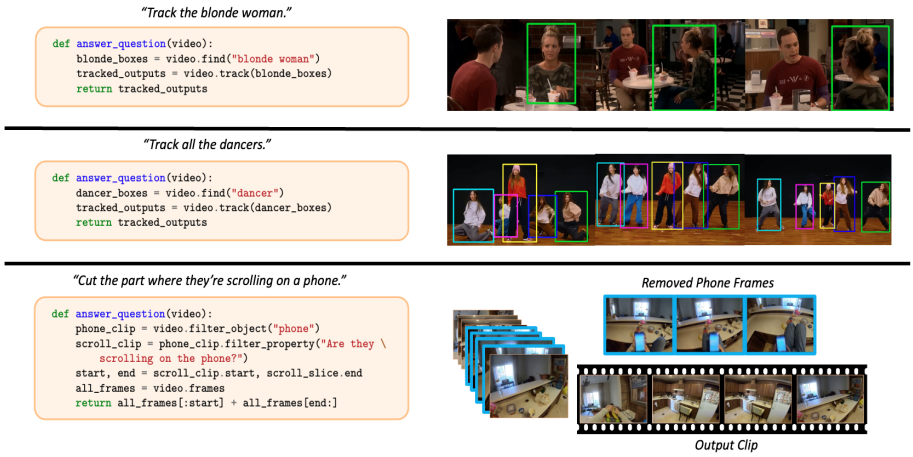

Fig. 8: ProViQ exhibits additional video capabilities. ProViQ can compose its modules into programs besides question answering, such as query-based multi-object tracking or basic video editing.

based on an input query. An example is shown in Figure 8 where we are able to track all the dancers in a complicated scene through detecting them in each frame, and tracking them over time.

Video Editing ProViQ can also combine its modules to retrieve relevant clips and remove clips that do not satisfy input criteria, enabling a basic form of video editing. An example is shown in Figure 8, where we ask ProViQ to cut all parts of a video where the subject is scrolling their phone. By combining retrieval with basic Python list slicing, ProViQ is able to cut and splice videos despite not being trained to do so.

5 Conclusions

We presented ProViQ, a method that extends modular vision methods to zero-shot video question answering. Our approach achieves a significant improvement over the state-of-the-art by using strong language models for temporal reasoning with code rather than using an end-to-end network. Extensive ablation studies and analysis demonstrate the strengths of our method, suggesting procedural reasoning can improve performance for vision tasks in general. We also present other creative capabilities enabled by ProViQ such as grounded tracking and text-based video editing. that require no modifications to the method. We believe ProViQ will serve as a strong video question answering baseline for the community going forward.

Limitations. Although ProViQ is capable across a diverse range of questions, it may struggle for questions for which its modules are not suited due to the lack of helpful in-context examples. Additionally, while it faster than other visual programming methods, it is slower than end-to-end approaches because it needs to generate, then execute the code, which varies across questions. We believe these issues can be overcome in future work.

In this supplement, we include additional results referenced in the main text to support our experiments and analysis, along with our source code. We furthermore include several video demos on our attached project page, which we strongly encourage readers to view. The project page is accessible by opening the `index.html` file in the `website` folder of the supplemental material.

A Implementation Details

Our codebase is a heavily modified fork of the original ViperGPT [46] codebase. We used Langchain for interfacing with large language models and including in-context examples in the prompt. For visual modules, we used BLIP-2 [23] for Image QA, GroundingDino [31] (with Swin-T backbone) for object detection, LaViLa [67] for video captioning, and ByteTrack [66] for object tracking. Each video is represented as 60 frames at their native resolution. All model inference can be done with a single Nvidia A100 GPU, but we split the evalution over multiple GPUs for speed, especially for datasets with large test sets, like MSR-VTT.

B Datasets and Metrics Details

B.1 Datasets

TGIF-QA [26], MSVD-QA [53], and **MSRVTT-QA [53]** are open-ended VideoQA benchmarks automatically generated from captions, with some manual annotations. Each question has a single answer that is one word or phrase. TGIF-QA consists of GIFs that are a few seconds long, while MSVD and MSR-VTT can be up to 15 seconds. For MSVD and MSRVTT, questions are split into five categories: who, what, when, where and how, while TGIF is split into color, object, location, and number.
iVQA [54] is a recent open-ended benchmark based on instructional videos. It only includes visual questions, and for each answer has multiple correct answers, reducing ambiguity for each question.
ActivityNet-QA [61] is an open-ended benchmark containing longer videos, up to 3 minutes long. It contains nine categories: motion, spatial, temporal, yes/no, color, object, location, number and 'free'.
TVQA [21] is a multiple-choice video QA dataset focused on multimodal understanding from clips of popular TV shows. Each question has 5 answers, and each question also has a ground-truth scene transcript. Questions are either visual or focused on the narrative aspect of the scene.
EgoSchema [33] is a recent multiple-choice zero-shot benchmark focused on long-term video understanding. Videos are sourced from Ego4D and are all 3 minutes long, which questions requiring long-horizon understanding. It contains 5K video in a held-out test split and no training data. The questions and answers are created from processing Ego4D ground-truth narrations with an LLM.

	What	Who	Number	Color	When	Where	Full
Just Ask [54]	7.8	1.7	74.3	18.8	3.5	0.0	13.3
FrozenBiLM [55]	26.0	**45.0**	69.9	56.3	5.2	17.9	33.8
ProViQ (Ours)	**38.1**	40.1	**70.9**	**56.3**	**36.5**	**50.0**	**37.5**
Just Ask [54]	1.8	0.7	66.3	0.6	0.6	4.5	5.6
FrozenBiLM [55]	10.7	**28.7**	55.0	11.4	9.2	9.3	16.9
ProViQ (Ours)	**14.6**	28.1	**67.1**	**19.3**	**22.5**	**22.6**	**22.1**

Table 3: Zero-shot QA results on the MSVD (above) and MSR-VTT (below) datasets, separated by category.

Method	iVQA	ANet	MSVD	TGIF
Video-ChatGPT [32]	-	35.2	64.9	60.1
MovieChat [44]	-	35.1	60.1	59.3
Video-Chat [25]	-	26.5	56.3	34.4
ProViQ(ours)	**53.7**	**42.3**	**37.5**	**66.1**

Table 4: Evaluation with language model-based metric. We evaluate following the protocol of [32] instead of using top-1 accuracy. ProViQ is still superior with this metric.

NeXT-QA [52] is a multiple-choice benchmark designed to test causal and temporal reasoning. The answer to the question is typically found in a short timespan, while videos can be up to a minute long. Each question has 5 distinct choices.

B.2 Metrics

A known issue with open-ended video QA datasets is that top-1 accuracy requires a single correct answer, even though the questions are often ill-posed and have multiple valid answers. One line of work [32, 44] that focuses on video conversational assistants propose a different metric based on using LLMs for evaluation. In particular, they compare a sentence or paragraph output from their model to the ground-truth answer (a single word or phrase) and prompt GPT3.5 to output a binary correctness score as well as a subjective score to rate its conversational ability. While this is suitable for measuring conversational ability, we found that this metric is unreliable for measuring answer accuracy and often leads to incorrect evaluation. We compare ProViQ to these works using this metric in Table 4. Although we believe this metric to be flawed, ProViQ still exhibits superior performance on it compared to other works.

B.3 Per-dataset Results

We provide breakdowns of the QA results on each dataset by question type. In Table 5, we present the ActivityNet results, and in Table 3 we show both

ActivityNet-QA	Motion	Spatial	Time	Y/N	Color	Object	Location	Number	Free	Full
Just Ask [54]	2.3	1.1	0.3	36.3	11.3	4.1	6.5	0.2	4.7	12.3
FrozenBiLM [55]	12.7	**6.8**	1.6	53.2	16.5	17.9	18.1	26.2	25.8	25.9
ProViQ (Ours)	**38.3**	5.6	**3.3**	**70.4**	**35.7**	**20.0**	**35.7**	**39.1**	**43.7**	**41.3**

Table 5: Zero-shot QA results per category on the ActivityNet-QA dataset.

	Color	Number	Loc	Obj	Full
FrozenBiLM	31.3	67.8	38.2	40.1	41.9
ProViQ	74.6	81.2	51.2	47.8	66.1

Table 6: Zero-shot video QA results per category on the TGIF-QA dataset.

	Causal	Temporal	Full
ViperGPT	49.8	56.4	60.0
ProViQ	55.6	60.1	63.8

Table 7: Zero-shot video QA results per category on the NeXT-QA dataset.

the MSVD and MSR-VTT results on each category. We also include results on NeXT-QA and TGIF-QA in Tables 7 and 6.

C Additional Ablations

Choice of Language Model The main paper results were based on using `gpt-3.5-turbo` for program generation. We evaluate the difference when using other language models in Table 8. Specifically, we tested CodeLLama13-b, WizardCoder15-b, CodeLlama-34b, and GPT-4. We generally found that open-source language models were reasonably competitive; however, the lower the parameter count, the lower the overall benchmark performance. This was usually due to weaker models producing more code fragments that did not compile, being less able to follow instructions from the prompt, or not invoking methods from the API correctly.

Word Matcher Open-ended benchmarks are formulated as K-way classification problems, with K, the size of the answer vocabulary, often ranging into the thousands. For end-to-end models typically finetune pretrained features on question-answering datasets, and pick the answer from the output vocabulary with the highest score. Since our setup uses a video-language model, we map outputs to the semantically nearest word or phrase in the output vocabulary. Without this component, any prediction that is not in the vocabulary will automatically be treated as incorrect, leading to significantly worse performance on QA benchmarks. We used FastText, but other options, such as BERT or `word2vec`, could be used as well. We demonstrate the accuracy from using each of these embeddings in Table 9.

Output Vocabulary Some open-ended benchmarks have enormous answer vocabularies: MSR-VTT has 73K questions in the test set, with over 10,000 unique answers. Standard practice [54, 55, 62] is to use the most common 1000 answers from the training set as the vocabulary. One other input at test time is the question category: we used this to constrain the vocabulary further. For a question

Method	iVQA	ANet	MSVD	TGIF
CodeLLama-13B	48.6	34.2	32.5	60.3
WizardCoder-15B	48.9	35.1	33.6	59.8
CodeLlama-34B	49.9	38.2	33.9	62.2
`gpt-3.5`	50.7	42.3	37.5	66.1

Table 8: Comparison with open-source language models. Open-source language models work reasonably well, but are still inferior to larger, closed-source models.

	TGIF-QA	MSVD	MSR-VTT	ActivityNet	iVQA
`word2vec` [35]	63.3	36.7	19.3	41.1	47.5
BERT [7]	61.1	**37.7**	14.4	39.3	44.3
FastText [18]	**66.1**	37.5	**22.1**	**42.3**	**50.7**
No Constraint	61.4	37.3	12.9	35.3	50.1
Top-1000	63.8	37.1	17.8	41.1	**50.7**
Type-based	**66.1**	**37.5**	**22.1**	**42.3**	-

Table 9: Ablation of components related to the output vocabularies on open-ended video dataset benchmarks. The top half of the table shows the impact of using different word-matching embeddings, and the bottom half shows the effect of restricting the output vocabulary in different ways.

type Q, the output vocabulary is the list of all answers for questions of type Q in the training set that are also among the top 1000 answers. Since our method needs no training, dynamically altering the output vocabulary is straightforward, and we found that this can significantly boost performance on datasets with poor label quality and ambiguous phrasing, such as MSRVTT-QA. The results of ablating on these components are in Table 9. We see that the vocabulary size matters most for lower-quality labeled datasets, such as ActivityNet-QA and MSRVTT-QA, while the effect is minimal in higher-quality labels like TGIF and iVQA.

D In-Context Examples

As mentioned in the main text, ViperGPT [46] mostly uses fixed in-context examples, both in the API docstrings and as additional input in the prompt. In particular, they use 8 in-context examples for Next-QA evaluation. On the other hand, CodeVQA [45] and VISPROG [13] annotate a small pool of examples with programs and use a retrieval mechanism to construct the prompt. We opt for this approach. For each benchmark dataset, we annotated several (at most 10) in-context examples, and used an embedding-based retrieval method to add the three closest in-context examples to the prompt. We embed the input questions and retrieve the examples whose questions are closest in embedding space. All examples are annotated from the training set. Some components of our method involve language model calls. For example, the `choose_option` method calls a

language model to choose the most appropriate response in a multiple choice question given some input context. This uses fixed in-context examples, and we do not use any sort of retrieval for these methods, only for the program generation step. We found that example retrieval was unnecessary for strong performance for subtasks and that we could rely on the base model for good performance. Finally, for our open-world demo, we use eight fixed in-context examples as there is no benchmark or standard pool to retrieve from.

E Prompt

We include the full prompt containing the visual API for our model on the next page. The prompt slightly changes for each dataset: following [46], we exclude certain methods when running on datasets where they are not applicable. For example, we only include the `get_summary` method on the EgoSchema benchmark. In addition to the following prompt, we include in-context examples for each dataset, which are appended to the API prompt along with the input question or command.

```
def get_max_key(responses: Dict[str, int]) -> str
    """
    Given a dict, returns the key with the highest count.
    """

class VideoClip:
    """A Python class containing a set of frames and methods for querying them.
    Attributes
    ----------
    video : torch.Tensor
        A tensor of image frames.
    start : int
        An int describing the starting frame in this video segment.
    end : int
        An int describing the ending frame in this video segment .
    num_frames->int
        An int containing the number of frames in the video segment.
    trimmed_video: torch.Tensor
        A trimmed video from start to end of the original input tensor.
    """

    def __init__(self, video: torch.Tensor, start: int = None, end: int = None, parent_start=0, queues=None):
        """Initializes a VideoClip object.

        Parameters
        -------
        video : torch.Tensor
            A tensor of the original video.
        start : int
            An int describing the starting frame in this video segment.
        end : int
            An int describing the ending frame in this video segment.
        """
```

```
    def filter_property(self, property:str) -> VideoClip:
        """
        Given a Yes/no query, returns a VideoClip composed only of the frames
        where that statement is true.
        Parameters
        -----------
        property: str
            A query to filter the video segment with.

        Returns: VideoClip
            A VideoClip composed only of the frames where the input
            property is true.

        Examples
        ----------
        question: What is the party for?
        def answer_question(video, possible_answers):
            party_segment = video.filter_property("Is a party happening?")
            responses = vid_segment.video_query("What is the party for?",
    possible_answers)
            return get_max_key(responses)
        """

    def filter_object(self, object: str) -> VideoClip:
        """
        Given a object, returns a VideoClip composed only of frames where
        that object is present.
        Parameters
        -----------
        object: str
            The object to look for.

        Returns: VideoClip
            A VideoClip composed only of the frames containing the input
            object.

        Examples
        ----------
        question: What color is the skier's jacket?
        def answer_question(video, possible_answers):
            skier_clip = video.filter_object("skier")
            skier_boxes = video.find("skier")
            jacket_boxes = skier_clip.find("jacket")
            responses = jacket_boxes.video_query("What color is this jacket
    ?", possible_answers)
            return get_max_key(responses)
        """

      def video_query(self, query: str, possible_answers: List[str]) -> Dict
    :
        """Answers a query for each frame in the video and returns a dict
    with the count of responses.
        Parameters
        -----------
        query: str
            The question to be answered.
        possible_answers : List[str]
            The list of possible answers for output.

        Returns: Dict
            The query answers, grouped by how many frames they occur for.

        Examples
        --------
        question: what is the person doing?
        def answer_question(video, possible_answers):
            responses = video.video_query("What is the person doing?",
    possible_answers)
            return get_max_key(responses)
        """
```

```
    def get_caption(self, index: int) -> str:
        """
        Gets a caption of the frame at that index in the video segment.
        Parameters
        ----------
        index: int
            The index of the frame to use. Range is [0, self.num_frames -1].

        Returns : str
            The image caption of the frame at that index.
        """

    def find(self, object: str) -> VideoClip:
        """
        Finds all bounding boxes around a certain object in a video segment,
        and collates them into a collection of frames.

        Parameters
        ---------
        object: str
            The object to look for.

        Returns : VideoClip
            A VideoClip object composed of crops of the object.
        """

    # This is only included in the prompt if we can get the script.
    def get_script(self) -> str:
        """
        Returns:
            A string script of the speech spoken during the video, if
     available.
        """

    # This is only included in the prompt for the Egoschema evaluation,
    # and should only be used if a sufficient video captioning model exists.
    def get_summary(self) -> str:
        """
        Returns: str
            A string summary representing the narrative of the video.
        """

    def track_objects(self, input_boxes: List[torch.Tensor]): -> List(STrack)
        """
        Runs a tracker on a set of input bounding boxes, representing some
     object(s)
        detected over time. Returns the boxes grouped by track ID.

        Parameters
        ---------
        input_boxes: List[torch.Tensor]
            A list of all the detected boxes at each frame in the video.

        Returns: List[STrack]
            A list of tracked objects with bounding box and time information.
        --------
        """


    def choose_option(self, question:str, context: Dict, options: List[str])
     -> str:
        """
        Uses a language model to choose the option that best answers the
     question
        given the input context.

        Parameters
        ----------
```

```
    """
        question: str
            The input query.
        context: Dict
            Any useful context, such as scripts, visual information, or summaries.
        options: List[str]
            The list of options to choose from, numbered.

        Returns: str
            A string detailing which number option was chosen with reasoning.

        Examples
        ---------
        question: How was the toy bear moved to the front?
        def answer_question(video, possible_answers):
            vid_seg = video.trim(0, len(video) // 4) # consider the star
            bear_seg = vid_seg.filter_object("bear")
            image_context = bear_seg.get_caption(bear_seg.num_frames // 2)
            activity_context = bear_seg.video_query("What is this?")
            context = {"caption": image_context, "activity": activity_context}
            answer = bear_seg.choose_option("how was the toy bear moved to the front?", context, possible_answers)
            return answer
        """
```

References

1. Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F.L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al.: Gpt-4 technical report. arXiv preprint arXiv:2303.08774 (2023)
2. Alayrac, J.B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., Lenc, K., Mensch, A., Millican, K., Reynolds, M., et al.: Flamingo: a visual language model for few-shot learning. Advances in Neural Information Processing Systems **35**, 23716–23736 (2022)
3. Andreas, J., Rohrbach, M., Darrell, T., Klein, D.: Neural module networks. In: CVPR (2016)
4. Bain, M., Nagrani, A., Varol, G., Zisserman, A.: Frozen in time: A joint video and image encoder for end-to-end retrieval. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 1728–1738 (2021)
5. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. Advances in neural information processing systems **33**, 1877–1901 (2020)
6. Carreira, J., Zisserman, A.: Quo vadis, action recognition? a new model and the kinetics dataset. In: proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 6299–6308 (2017)
7. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
8. Fan, C., Zhang, X., Zhang, S., Wang, W., Zhang, C., Huang, H.: Heterogeneous memory enhanced multimodal attention model for video question answering. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 1999–2007 (2019)
9. Fu, T.J., Li, L., Gan, Z., Lin, K., Wang, W.Y., Wang, L., Liu, Z.: Violet: End-to-end video-language transformers with masked visual-token modeling. arXiv preprint arXiv:2111.12681 (2021)
10. Gao, J., Ge, R., Chen, K., Nevatia, R.: Motion-appearance co-memory networks for video question answering. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 6576–6585 (2018)
11. Girdhar, R., El-Nouby, A., Liu, Z., Singh, M., Alwala, K.V., Joulin, A., Misra, I.: Imagebind: One embedding space to bind them all. In: CVPR (2023)
12. Grauman, K., Westbury, A., Byrne, E., Chavis, Z., Furnari, A., Girdhar, R., Hamburger, J., Jiang, H., Liu, M., Liu, X., et al.: Ego4d: Around the world in 3,000 hours of egocentric video. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 18995–19012 (2022)
13. Gupta, T., Kembhavi, A.: Visual programming: Compositional visual reasoning without training. In: CVPR (2023)
14. Hu, R., Andreas, J., Darrell, T., Saenko, K.: Explainable neural computation via stack neural module networks. In: Proceedings of the European conference on computer vision (ECCV). pp. 53–69 (2018)
15. Hu, R., Andreas, J., Rohrbach, M., Darrell, T., Saenko, K.: Learning to reason: End-to-end module networks for visual question answering. In: ICCV (2017)
16. Huang, D., Chen, P., Zeng, R., Du, Q., Tan, M., Gan, C.: Location-aware graph convolutional networks for video question answering. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34, pp. 11021–11028 (2020)

17. Jang, Y., Song, Y., Yu, Y., Kim, Y., Kim, G.: Tgif-qa: Toward spatio-temporal reasoning in visual question answering. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2758–2766 (2017)
18. Joulin, A., Grave, E., Bojanowski, P., Mikolov, T.: Bag of tricks for efficient text classification. In: Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers. pp. 427–431. Association for Computational Linguistics (April 2017)
19. Kim, J., Ma, M., Pham, T., Kim, K., Yoo, C.D.: Modality shifting attention network for multi-modal video question answering. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 10106–10115 (2020)
20. Le, T.M., Le, V., Venkatesh, S., Tran, T.: Hierarchical conditional relation networks for video question answering. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 9972–9981 (2020)
21. Lei, J., Yu, L., Bansal, M., Berg, T.L.: Tvqa: Localized, compositional video question answering. arXiv preprint arXiv:1809.01696 (2018)
22. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.t., Rocktäschel, T., et al.: Retrieval-augmented generation for knowledge-intensive nlp tasks. Advances in Neural Information Processing Systems **33**, 9459–9474 (2020)
23. Li, J., Li, D., Savarese, S., Hoi, S.: Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. arXiv preprint arXiv:2301.12597 (2023)
24. Li, J., Li, D., Xiong, C., Hoi, S.: Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In: International Conference on Machine Learning. pp. 12888–12900. PMLR (2022)
25. Li, K., He, Y., Wang, Y., Li, Y., Wang, W., Luo, P., Wang, Y., Wang, L., Qiao, Y.: Videochat: Chat-centric video understanding. arXiv preprint arXiv:2305.06355 (2023)
26. Li, Y., Song, Y., Cao, L., Tetreault, J., Goldberg, L., Jaimes, A., Luo, J.: TGIF: A New Dataset and Benchmark on Animated GIF Description. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2016)
27. Lin, B., Zhu, B., Ye, Y., Ning, M., Jin, P., Yuan, L.: Video-llava: Learning united visual representation by alignment before projection. arXiv preprint arXiv:2311.10122 (2023)
28. Lin, X., Bertasius, G., Wang, J., Chang, S.F., Parikh, D., Torresani, L.: Vx2text: End-to-end learning of video-based text generation from multimodal inputs. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 7005–7015 (2021)
29. Liu, H., Li, C., Li, Y., Lee, Y.J.: Improved baselines with visual instruction tuning (2023)
30. Liu, H., Li, C., Wu, Q., Lee, Y.J.: Visual instruction tuning. In: NeurIPS (2023)
31. Liu, S., Zeng, Z., Ren, T., Li, F., Zhang, H., Yang, J., Li, C., Yang, J., Su, H., Zhu, J., et al.: Grounding dino: Marrying dino with grounded pre-training for open-set object detection. arXiv preprint arXiv:2303.05499 (2023)
32. Maaz, M., Rasheed, H., Khan, S., Khan, F.: Video-chatgpt: Towards detailed video understanding via large vision and language models. ArXiv 2306.05424 (2023)
33. Mangalam, K., Akshulakov, R., Malik, J.: Egoschema: A diagnostic benchmark for very long-form video language understanding. arXiv preprint arXiv:2308.09126 (2023)

34. Miech, A., Zhukov, D., Alayrac, J.B., Tapaswi, M., Laptev, I., Sivic, J.: Howto100m: Learning a text-video embedding by watching hundred million narrated video clips. In: Proceedings of the IEEE/CVF international conference on computer vision. pp. 2630–2640 (2019)
35. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
36. OpenAI, R.: Gpt-4 technical report. arXiv pp. 2303–08774 (2023)
37. Park, J., Lee, J., Sohn, K.: Bridge to answer: Structure-aware graph interaction network for video question answering. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 15526–15535 (2021)
38. Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al.: Learning transferable visual models from natural language supervision. In: International conference on machine learning. pp. 8748–8763. PMLR (2021)
39. Radford, A., Kim, J.W., Xu, T., Brockman, G., McLeavey, C., Sutskever, I.: Robust speech recognition via large-scale weak supervision. In: International Conference on Machine Learning. pp. 28492–28518. PMLR (2023)
40. Reddy, R.G., Rui, X., Li, M., Lin, X., Wen, H., Cho, J., Huang, L., Bansal, M., Sil, A., Chang, S.F., et al.: Mumuqa: Multimedia multi-hop news question answering via cross-media knowledge extraction and grounding. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 36, pp. 11200–11208 (2022)
41. Saqur, R., Narasimhan, K.: Multimodal graph networks for compositional generalization in visual question answering. Advances in Neural Information Processing Systems **33**, 3070–3081 (2020)
42. Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., Scialom, T.: Toolformer: Language models can teach themselves to use tools. arXiv preprint arXiv:2302.04761 (2023)
43. Seo, P.H., Nagrani, A., Schmid, C.: Look before you speak: Visually contextualized utterances. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 16877–16887 (2021)
44. Song, E., Chai, W., Wang, G., Zhang, Y., Zhou, H., Wu, F., Guo, X., Ye, T., Lu, Y., Hwang, J.N., et al.: Moviechat: From dense token to sparse memory for long video understanding. arXiv preprint arXiv:2307.16449 (2023)
45. Subramanian, S., Narasimhan, M., Khangaonkar, K., Yang, K., Nagrani, A., Schmid, C., Zeng, A., Darrell, T., Klein, D.: Modular visual question answering via code generation. In: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). pp. 747–761. Association for Computational Linguistics, Toronto, Canada (Jul 2023). https://doi.org/10.18653/v1/2023.acl-short.65, https://aclanthology.org/2023.acl-short.65
46. Surís, D., Menon, S., Vondrick, C.: Vipergpt: Visual inference via python execution for reasoning. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). pp. 11888–11898 (October 2023)
47. Team, G., Anil, R., Borgeaud, S., Wu, Y., Alayrac, J.B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A.M., Hauth, A., et al.: Gemini: a family of highly capable multimodal models. arXiv preprint arXiv:2312.11805 (2023)
48. Wang, Y., He, Y., Li, Y., Li, K., Yu, J., Ma, X., Li, X., Chen, G., Chen, X., Wang, Y., et al.: Internvid: A large-scale video-text dataset for multimodal understanding and generation. arXiv preprint arXiv:2307.06942 (2023)
49. Wang, Y., Li, K., Li, Y., He, Y., Huang, B., Zhao, Z., Zhang, H., Xu, J., Liu, Y., Wang, Z., et al.: Internvideo: General video foundation models via generative and discriminative learning. arXiv preprint arXiv:2212.03191 (2022)

50. Wang, Z., Li, M., Xu, R., Zhou, L., Lei, J., Lin, X., Wang, S., Yang, Z., Zhu, C., Hoiem, D., et al.: Language models with image descriptors are strong few-shot video-language learners. Advances in Neural Information Processing Systems **35**, 8483–8497 (2022)
51. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q.V., Zhou, D., et al.: Chain-of-thought prompting elicits reasoning in large language models. Advances in Neural Information Processing Systems **35**, 24824–24837 (2022)
52. Xiao, J., Shang, X., Yao, A., Chua, T.S.: Next-qa: Next phase of question-answering to explaining temporal actions. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 9777–9786 (2021)
53. Xu, D., Zhao, Z., Xiao, J., Wu, F., Zhang, H., He, X., Zhuang, Y.: Video question answering via gradually refined attention over appearance and motion. In: ACM Multimedia (2017)
54. Yang, A., Miech, A., Sivic, J., Laptev, I., Schmid, C.: Just ask: Learning to answer questions from millions of narrated videos. In: ICCV (2021)
55. Yang, A., Miech, A., Sivic, J., Laptev, I., Schmid, C.: Zero-shot video question answering via frozen bidirectional language models. In: NeurIPS (2022)
56. Yang, R., Song, L., Li, Y., Zhao, S., Ge, Y., Li, X., Shan, Y.: Gpt4tools: Teaching large language model to use tools via self-instruction. arXiv preprint arXiv:2305.18752 (2023)
57. Yang, Z., Gan, Z., Wang, J., Hu, X., Lu, Y., Liu, Z., Wang, L.: An empirical study of gpt-3 for few-shot knowledge-based vqa. In: AAAI (2022)
58. Ye, Q., Xu, G., Yan, M., Xu, H., Qian, Q., Zhang, J., Huang, F.: Hitea: Hierarchical temporal-aware video-language pre-training. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 15405–15416 (2023)
59. Ye, Q., Xu, H., Xu, G., Ye, J., Yan, M., Zhou, Y., Wang, J., Hu, A., Shi, P., Shi, Y., et al.: mplug-owl: Modularization empowers large language models with multimodality. arXiv preprint arXiv:2304.14178 (2023)
60. Yu, S., Cho, J., Yadav, P., Bansal, M.: Self-chained image-language model for video localization and question answering. Advances in Neural Information Processing Systems **36** (2024)
61. Yu, Z., Xu, D., Yu, J., Yu, T., Zhao, Z., Zhuang, Y., Tao, D.: Activitynet-qa: A dataset for understanding complex web videos via question answering. In: AAAI. pp. 9127–9134 (2019)
62. Zellers, R., Lu, J., Lu, X., Yu, Y., Zhao, Y., Salehi, M., Kusupati, A., Hessel, J., Farhadi, A., Choi, Y.: Merlot reserve: Neural script knowledge through vision and language and sound. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 16375–16387 (2022)
63. Zellers, R., Lu, X., Hessel, J., Yu, Y., Park, J.S., Cao, J., Farhadi, A., Choi, Y.: Merlot: Multimodal neural script knowledge models. Advances in Neural Information Processing Systems **34**, 23634–23651 (2021)
64. Zeng, A., Attarian, M., Ichter, B., Choromanski, K., Wong, A., Welker, S., Tombari, F., Purohit, A., Ryoo, M., Sindhwani, V., et al.: Socratic models: Composing zero-shot multimodal reasoning with language. arXiv preprint arXiv:2204.00598 (2022)
65. Zhang, C., Lu, T., Islam, M.M., Wang, Z., Yu, S., Bansal, M., Bertasius, G.: A simple llm framework for long-range video question-answering (2023)
66. Zhang, Y., Sun, P., Jiang, Y., Yu, D., Weng, F., Yuan, Z., Luo, P., Liu, W., Wang, X.: Bytetrack: Multi-object tracking by associating every detection box (2022)
67. Zhao, Y., Misra, I., Krähenbühl, P., Girdhar, R.: Learning video representations from large language models. In: CVPR (2023)

68. Zhu, D., Chen, J., Shen, X., Li, X., Elhoseiny, M.: Minigpt-4: Enhancing vision-language understanding with advanced large language models. arXiv preprint arXiv:2304.10592 (2023)