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 [14] 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 [5] for Image QA, GroundingDino [8] (with Swin-T backbone) for object detection, LaViLa [22] for video captioning, and ByteTrack [21] 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 [7], MSVD-QA [16], and **MSRVTT-QA [16]** 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 [17] 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 [19] 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 [4] 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 [10] 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 [17]	7.8	1.7	74.3	18.8	3.5	0.0	13.3
FrozenBiLM [18]	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 [17]	1.8	0.7	66.3	0.6	0.6	4.5	5.6
FrozenBiLM [18]	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 1: Zero-shot QA results on the MSVD (above) and MSR-VTT (below) datasets, separated by category.

Method	iVQA	ANet	MSVD	TGIF
Video-ChatGPT [9]	-	35.2	64.9	60.1
MovieChat [12]	-	35.1	60.1	59.3
Video-Chat [6]	-	26.5	56.3	34.4
ProViQ(ours)	**53.7**	**42.3**	**37.5**	**66.1**

Table 2: Evaluation with language model-based metric. We evaluate following the protocol of [9] instead of using top-1 accuracy. ProViQ is still superior with this metric.

NeXT-QA [15] 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 [9, 12] 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 2. 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 3, we present the ActivityNet results, and in Table 1 we show both

ActivityNet-QA	Motion	Spatial	Time	Y/N	Color	Object	Location	Number	Free	Full
Just Ask [17]	2.3	1.1	0.3	36.3	11.3	4.1	6.5	0.2	4.7	12.3
FrozenBiLM [18]	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 3: 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 4: 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 5: 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 5 and 4.

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 6. 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 7.

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 [17, 18, 20] 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 6: 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` [11]	63.3	36.7	19.3	41.1	47.5
BERT [1]	61.1	**37.7**	14.4	39.3	44.3
FastText [3]	**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 7: 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 7. 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 [14] 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 [13] and VISPROG [2] 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 [14], 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. 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)
2. Gupta, T., Kembhavi, A.: Visual programming: Compositional visual reasoning without training. In: CVPR (2023)
3. 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)
4. Lei, J., Yu, L., Bansal, M., Berg, T.L.: Tvqa: Localized, compositional video question answering. arXiv preprint arXiv:1809.01696 (2018)
5. 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)
6. 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)
7. 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)
8. 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)
9. Maaz, M., Rasheed, H., Khan, S., Khan, F.: Video-chatgpt: Towards detailed video understanding via large vision and language models. ArXiv 2306.05424 (2023)
10. Mangalam, K., Akshulakov, R., Malik, J.: Egoschema: A diagnostic benchmark for very long-form video language understanding. arXiv preprint arXiv:2308.09126 (2023)
11. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
12. 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)
13. 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
14. 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)
15. 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)
16. 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)

17. Yang, A., Miech, A., Sivic, J., Laptev, I., Schmid, C.: Just ask: Learning to answer questions from millions of narrated videos. In: ICCV (2021)
18. Yang, A., Miech, A., Sivic, J., Laptev, I., Schmid, C.: Zero-shot video question answering via frozen bidirectional language models. In: NeurIPS (2022)
19. 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)
20. 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)
21. 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)
22. Zhao, Y., Misra, I., Krähenbühl, P., Girdhar, R.: Learning video representations from large language models. In: CVPR (2023)