VTimeCoT: Thinking by Drawing for Video Temporal Grounding and Reasoning

Supplementary Material

1. Implementation Details

Prompt of VTimeCoT. The agent framework of the MLLM is adapted from AutoGen [4]. The initialization prompt for temporal grounding and reasoning task is shown in Prompt 1. Prompt 2, 3, 4, 5 provide the example for each dataset. For tool functions, the prompt provides their declarations, functionalities, and input/output interface instructions. The tool function implementations are imported as external programs in the form of Python packages. For video memory, the function to load frames is used to pre-store frames in a designated 'frames' variable to be invoked by the generated program. The 'display' function dynamically updates the video memory during execution. To accomplish frame-synchronization of any FPS sampling, The variables 'fps' and 'total_time' are loaded to be invoked by the generated program.

2. Additional Qualitative Comparisons

In this section, we present additional qualitative analyses of the proposed VTimeCoT, in comparison with the baseline model, GPT-4o. Fig. 1 illustrates the qualitative evaluation. Across these examples, we observe that VTimeCoT consistently demonstrates superior temporal grounding and reasoning capabilities. Notably, the proposed approach effectively leverages tools to identify relevant segments and focus on key frames for precise reasoning. This stands in stark contrast to GPT-4o, which often exhibits errors in questions required for scene retrieval and reasoning.

3. Additional Ablation Study

We perform additional ablation study to compare different moment-highlighting types. As shown in Tab. 1, our approach with both visual and textual cues proves to be highly beneficial.

4. Limitations

First, similar to other thought-based methods [1, 5], VTimeCoT requires generating a longer sequence of tokens and demands greater computational resources compared to directly producing the final answer. Second, similar to prior

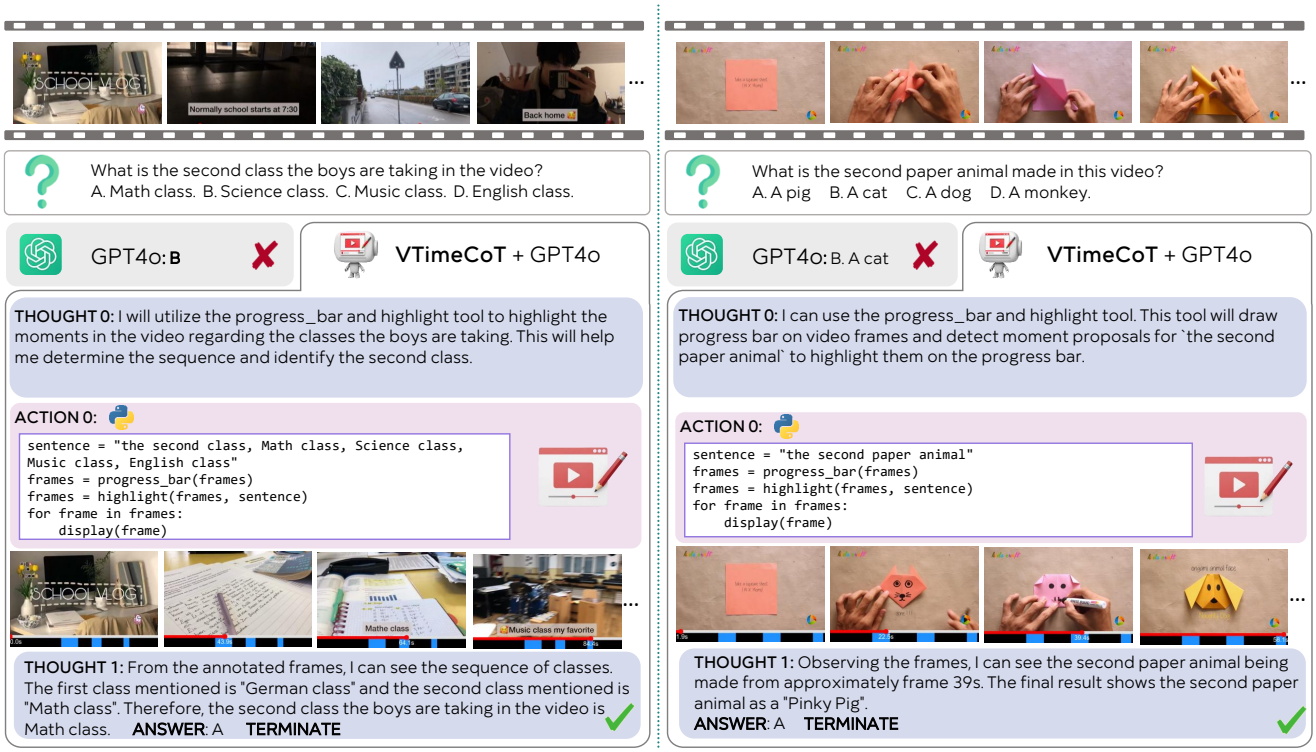

Figure 1. **Additional Qualitative comparison on VideoMME benchmark.** In contrast to GPT-4o, VTimeCoT reasons using the progress bar and highlights the key moments, leading to accurate answers.

Moment Prompt	Text	Visual	Text + Visual
Accuracy	69.7	72.7	73.7

Table 1. **Ablations on moment prompt types.** On a subset of 300 samples from VideoMME, we compare three approaches: prompting moment time using only text, using visual highlights with a progress bar, and combining both.

work based on external tools [2, 6], the performance of the external tool module ($e.g.$, VideoCLIP-XL [3]) has impacts on the effectiveness of the overall framework. We expect that as more powerful video-text foundation models emerge, our entire framework will continue to improve.

5. Future Directions

First, this work focuses on enhancing the performance of existing multimodal large models in a training-free manner. Future research could explore the impact of visual temporal prompts on training. For instance, naturally incorporating progress bars into the training data of multimodal large models may further improve their temporal grounding and reasoning capabilities. VTimeCoT could potentially be further strengthened through such tuned MLLMs. Besides, VTimeCoT could be extended to the domain of video editing. For example, in video montage tasks, VTimeCoT could be leveraged to search for relevant segments within the raw footage, highlight key moments of interest, and assemble clips into a better video.

Prompt 1: Intialization prompt for the MLLM to start chain-of-thought.

````
The image positions in video are as their order. The first image is at the start time of video.
The last image is at the end time of the video. The video is uniformly sampled.
Here are some tools that can help you. All are python codes. They are in tools.py and will be
imported for you.
You can use tools to highlight possible video intervals for answering questions.
Below are the tools in tools.py:
```python
def progress_bar(frames_list, fps, total_time):
    """Draw the progress bar on video frames. Return the annotated frames. Each frame's
    progress bar is added at the bottom.
    Parameters
    -------
    frames_list: List
        PIL image list for frames in the video.
    fps: float
        FPS of video.
    total_time: float
        Total time of video, in seconds.

    Returns
    -------
    List: A PIL image list for annotated frames

    Usage
    -------
    frames_list = progress_bar(frames_list, fps, total_time)
    """

def highlight(frames_list, sentence, fps, total_time, return_segment=True):
    """Detect high-similarity moment proposals for the sentence to highlight on the progress
    bar. Return the annotated highlighted frames. The highlighted time intervals are marked by
    blue masks at the bottom progress bar.
    Parameters
    -------
    frames_list: List
        PIL image list for frames in the video.
    sentence : string
        string describing the sentence.
    fps: float
        FPS of video.
    total_time: float
        Total time of video, in seconds.
    return_segment: bool
        Whether to return the list of intervals for the detected segments.
    Returns
    -------
    List: A PIL image list for annotated frames
    List: List of intervals for detected segments. The format is `[[start_time1, end_time1],
    [start_time2, end_time2], ...]` of seconds.

    Usage
    -------
    frames_list, segments = highlight(frames_list, sentence, fps, total_time)
    """
def cut(frames_list, segments, fps):
    """If the video is too long to find the required information, cut the specific segments in
    frames list. Return the cut frames.
    Parameters
    -------
    frames_list: List
        PIL image list for frames in the video.
    segments : list
        List of intervals for segments needed to be cut. The format is `[[start_time1,
        end_time1], [start_time2, end_time2], ...]` of seconds.

    Returns
    -------
    List: A PIL image list for cut frames

    Usage
    -------
    frames_list = cut(frames_list, segments, fps)
    """
```
````

Prompt 1: Intialization prompt for the MLLM to start chain-of-thought.

````
# GOAL #: Based on the above tools, I want you to reason about how to solve the # USER REQUEST
# and generate the actions step by step (each action is a python jupyter notebook code block)
to solve the request.

You may need to use the tools above to process the images and make decisions based on the
outputs of the previous code blocks.

Your video temporal localizaiton ability is not perfect, so you should use these tools to
assist you in reasoning about the video.

The jupyter notebook has already executed the following code to import the necessary packages:

```python
from PIL import Image
from tools import *
```

# REQUIREMENTS #:
1. The generated actions can resolve the given user request # USER REQUEST # perfectly. The
user request is reasonable and can be solved. Try your best to solve the request.
2. The arguments of a tool must be the same number, modality, and format specified in # TOOL
LIST #;
3. If you think you got the answer, use ANSWER: <your answer> to provide the answer, and ends
with TERMINATE.
4. All images in the initial user request are stored in PIL Image list as frames[0], frames[1],
frames[2] ... You can use these images in your code blocks.
5. Use as few tools as possible. Only use the tools for the use cases written in the tool
description. You can use multiple tools in a single action.
6. `fps` and `total_time` is already loaded. You can use them in your code blocks.

Below are some examples of how to use the tools to solve the user requests. You can refer to
them for help. You can also refer to the tool descriptions for more information.

{Examples for Charades-STA/QVHighlights/Vript-RR/VideoMME}

# USER REQUEST #: {question}
# USER IMAGE stored in frames as PIL image list.
Now please generate only THOUGHT 0 and ACTION 0 in RESULT. If no action needed, also reply with
ANSWER: <your answer> and ends with TERMINATE in the RESULT:
# RESULT #:
````

Prompt 2: Example prompt for Charades-STA

````
# EXAMPLE for Charades-STA: Detecting moments in the video
# USER REQUEST #:<Video frames here> During what time interval can we see cats are there in the
video?
# USER IMAGE stored in frames as PIL image list.
# RESULT #:
THOUGHT 0: I can use the progress_bar and highlight tool. This tool will draw progress bar on
video frames and detect moment proposals for `cats are there` to highlight them on progress
bar. Then I can display the highlighted frames to help me find the target moment interval.
ACTION 0:
```python
sentence = "cats are there"
frames = progress_bar(frames, fps, total_time)
frames, segments = highlight(frames, sentence, fps, total_time)
print(segments)
for frame in frames:
    display(frame)
```
OBSERVATION: Execution success. The output is as follows:
<the video frame outputs of the previous code is here.>
If you can get the answer, please reply with ANSWER: <your answer> and ends with TERMINATE.
Otherwise, please generate the next THOUGHT and ACTION.
THOUGHT 1:The tool is helpful. I see cats are there during 2.0s to 4.0s in the video.
ACTION 1: No action needed.
ANSWER: [2.0, 4.0]
TERMINATE
````

Prompt 3: Example prompt for QVHighlights

```
# EXAMPLE for QVHighlights: Detecting moments in the video
# USER REQUEST #: <Video frames here> During what time interval can we see the man is standing
up from the chair?
# USER IMAGE stored in frames as PIL image list.
# RESULT #:
THOUGHT 0: I can use the progress_bar and highlight tool. This tool will draw progress bar on
video frames and detect moment proposals for `the man is standing up from the chair`. to
highlight them on the progress bar. And I can display the highlighted frames to help me find
the target moment interval.
ACTION 0:
```python
sentence = "the man is standing up from the chair"
frames = progress_bar(frames, fps, total_time)
frames, segments = highlight(frames, sentence, fps, total_time)
print(segments)
for frame in frames:
    display(frame)
```
OBSERVATION: Execution success. The output is as follows:
<the video frame outputs of the previous code is here.>
If you can get the answer, please reply with ANSWER: <your answer> and ends with TERMINATE.
Otherwise, please generate the next THOUGHT and ACTION.
THOUGHT 1:The tool is helpful. I see the man is standing up from the chair during 3.0s to 4.5s
and 7.0s to 8.0s in the video.
ACTION 1: No action needed.
ANSWER: [[3.0, 4.5], [7.0, 8.0]]
TERMINATE
```

Prompt 4: Example prompt for Vript-RR

```
# EXAMPLE for Vript-RR: Answer questions with scene hint in the video
# USER REQUEST #:<Video frames here> Carefully watch the video and pay attention to the cause
and sequence of events, the detail and movement of objects, and the action and pose of persons.
Based on your observations, select the best option that accurately addresses the question.
Question: What is the object that this character receives?
Options:
(A) A yellow trophy.
(B) A yellow ornament.
(C) A yellow medal.
(D) A yellow cup.
Hint:In the video, there is a female pig who is in a pink complexion, wearing a orange coat,
red scarf, and brown hat, standing on yellow skis. This character is presented with an object
that matches the color of the skis she is standing on.
# USER IMAGE stored in frames as PIL image list.
# RESULT #:
THOUGHT 0: I can use the progress_bar and highlight tool to highlight moments of the hint. I
will cut video into segment for a thorough review. Then I can display the highlighted frames to
help me answer the question.
ACTION 0:
```python
sentence = "In the video, there is a female pig who is in a pink complexion, wearing a orange
coat, red scarf, and brown hat, standing on yellow skis. This character is presented with an
object that matches the color of the skis she is standing on."
frames = progress_bar(frames, fps, total_time)
frames, segments = highlight(frames, sentence, fps, total_time)
print(segments)
frames = cut(frames, segments, fps)
for frame in frames:
    display(frame)
```
OBSERVATION: Execution success. The output is as follows:
<the video frame outputs of the previous code is here.>
If you can get the answer, please reply with ANSWER: <your answer> and ends with TERMINATE.
Otherwise, please generate the next THOUGHT and ACTION.
THOUGHT 1:The tool is helpful. In the scenes related to the hint, I see this character receives
a yellow trophy.
ACTION 1: No action needed.
ANSWER: (A)
TERMINATE
```

Prompt 5: Example prompt for VideoMME

```
# EXAMPLE for VideoMME: Answer questions in the video
# USER REQUEST #:<Video frames here> Select the best answer to the following multiple-choice
question based on the video. Respond with only the letter (A, B, C, or D) of the correct option.
What is the color of the clothing worn by the persons in the video?
A. Black.
B. Gray.
C. Green.
D. Brown.
The best answer is:
# USER IMAGE stored in frames as PIL image list.
# RESULT #:
THOUGHT 0: I can use the progress_bar and highlight tool. This tool will draw progress bar on
video frames and detect moment proposals for 'the clothing worn by the persons' to highlight
them on progress bar. Then I can display the highlighted frames to help me answer the question.
ACTION 0:
'''python
sentence = "the clothing worn by the persons"
frames = progress_bar(frames, fps, total_time)
frames, segments = highlight(frames, sentence, fps, total_time)
print(segments)
for frame in frames:
    display(frame)
'''
OBSERVATION: Execution success. The output is as follows:
<the video frame outputs of the previous code is here.>
If you can get the answer, please reply with ANSWER: <your answer> and ends with TERMINATE.
Otherwise, please generate the next THOUGHT and ACTION.
THOUGHT 1:The tool is helpful. I see the color of the clothing worn by the persons is Green.
ACTION 1: No action needed.
ANSWER: C
TERMINATE
```

References

[1] Yushi Hu, Weijia Shi, Xingyu Fu, Dan Roth, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, and Ranjay Krishna. Visual sketchpad: Sketching as a visual chain of thought for multimodal language models. *arXiv preprint arXiv:2406.09403*, 2024. 1

[2] Juhong Min, Shyamal Buch, Arsha Nagrani, Minsu Cho, and Cordelia Schmid. Morevqa: Exploring modular reasoning models for video question answering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13235–13245, 2024. 2

[3] Jiapeng Wang, Chengyu Wang, Kunzhe Huang, Jun Huang, and Lianwen Jin. Videoclip-xl: Advancing long description understanding for video clip models. *arXiv preprint arXiv:2410.00741*, 2024. 2

[4] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*, 2023. 1

[5] Yixuan Wu, Yizhou Wang, Shixiang Tang, Wenhao Wu, Tong He, Wanli Ouyang, Philip Torr, and Jian Wu. Dettoolchain: A new prompting paradigm to unleash detection ability of mllm. In *European Conference on Computer Vision*, pages 164–182. Springer, 2024. 1

[6] Zongxin Yang, Guikun Chen, Xiaodi Li, Wenguan Wang, and Yi Yang. Doraemongpt: Toward understanding dynamic scenes with large language models (exemplified as a video agent). In *Forty-first International Conference on Machine Learning*, 2024. 2