Multimodal video analytics application

1. Concept Introduction

1.1 What is “Video Analysis”?

In the `largemodel` project, the **multimodal video analysis** feature refers to enabling the robot to process a video and summarize its core content, describe key events, or answer specific questions about the video in natural language. This allows the robot to leap from understanding only static images to understanding a dynamic world with temporal relationships.

The core tool for this feature is `analyze_video`. When a user provides a video file and asks a question (e.g., “Summarize what this video is about”), the system calls this tool to process and analyze the video and return the AI's textual answer.

1.2 Brief Description of Implementation Principles

The core challenge of offline video analysis lies in how to enable the model to efficiently process video data containing hundreds or thousands of frames. A mainstream implementation principle is as follows:

1. **Keyframe Extraction:** First, the system doesn't process every frame of the video. Instead, it extracts the most representative **keyframes** using algorithms (such as scene boundary detection or fixed-time interval sampling). This significantly reduces the amount of data that needs to be processed.
2. **Image Encoding:** Each extracted keyframe is fed into a visual encoder, similar to applications focused on visual understanding, and converted into a digital vector containing image information.
3. **Temporal Information Fusion:** This is the biggest difference from single-image understanding. The model needs to understand the temporal order between these keyframes. A recurrent neural network (RNN) or Transformer model is typically used to fuse the vectors of all keyframes, forming a "memory" vector that represents the dynamic content of the entire video.

4. **Question Answering and Generation:** The user's text question is encoded and then fused across modally with this "memory" vector. Finally, the language model generates a summary of the entire video or an answer to a specific question based on this fused information.

In simple terms, it's about **condensing a video into a few key images and their order, then understanding the whole story like reading a comic strip and answering related questions**.

---

2. Code Analysis

Key Code

1. Utility Layer Entry Point (`largemodel/utils/tools_manager.py`)

The `analyze_video` function in this file defines the tool's execution flow.

```
# From largemodel/utils/tools_manager.py
class ToolsManager:
    # ...
    def analyze_video(self, args):
        """
        Analyze video file and provide content description.


        :param args: Arguments containing video path.
        :return: Dictionary with video description and path.
        """
        self.node.get_logger().info(f"Executing analyze_video() tool with args:
{args}")
        try:
            video_path = args.get("video_path")
            # ... (Intelligent path fallback mechanism)

            if video_path and os.path.exists(video_path):
                # ... (Build Prompt)

                # Use a fully isolated, one-time context for video analysis to
ensure a plain text description.
                simple_context = [{
                    "role": "system",
                    "content": "You are a video description assistant. ..."
                }]

                result = self.node.model_client.infer_with_video(video_path,
prompt, message=simple_context)

                # ... (Processing result)
                return {
                    "description": description,
                    "video_path": video_path
                }
            # ... (Error handling)
```

2. Model Interface Layer and Frame Extraction (`largemodel/utils/large_model_interface.py`)

The functions in this file are responsible for processing video files and passing them to the underlying model.

```
# From largemodel/utils/large_model_interface.py

class model_interface:
    # ...
    def infer_with_video(self, video_path, text=None, message=None):
        """Unified video inference interface. """
        # ... (Preparing the message)
        try:
            # The specific implementation to be called is determined by
self.llm_platform.
            if self.llm_platform == 'ollama':
                response_content = self.ollama_infer(self.messages,
video_path=video_path)
            # ... (Logic of other online platforms)
        # ...
        return {'response': response_content, 'messages': self.messages.copy()}

    def _extract_video_frames(self, video_path, max_frames=5):
        """Extract keyframes from a video for analysis. """
        try:
            import cv2
            # ... (Video reading and frame interval calculation)
            while extracted_count < max_frames:
                # ... (Read video frames in a loop)
                if frame_count % frame_interval == 0:
                    # ... (Save the frame as a temporary image)
                    frame_base64 = self.encode_file_to_base64(temp_path)
                    frame_images.append(frame_base64)
            # ...
            return frame_images
        # ... (Exception handling)
```

Code Analysis

The implementation of video analysis is more complex than image analysis. It requires a crucial preprocessing step at the model interface layer: frame extraction.

1. **Tool Layer (`tools_manager.py`):**

- The `analyze_video` function is the business entry point for video analysis. Its responsibility is clear: receive a video file path and construct a prompt to request a description of the video content.
- It initiates the analysis process by calling the `self.node.model_client.infer_with_video` method. Like the visual understanding tool, it is completely unconcerned with the underlying model details, only responsible for passing the "video file" and "analysis instructions."

2. **Model Interface Layer (`large_model_interface.py`):**

- The `infer_with_video` function is the scheduler connecting the upper-layer tool and the lower-layer model. It distributes tasks to the appropriate concrete implementation functions

based on the platform configuration (`self.llm_platform`).
- Unlike processing a single image, processing video requires an additional step. The `_extract_video_frames` method demonstrates the general logic for this step: it uses the `cv2` library to read the video and extracts several (5 by default) keyframes.
- Each extracted frame is treated as an independent image and is typically encoded as a Base64 string.
- Finally, a request containing multiple frame image data and analysis instructions is sent to the large model. The model performs comprehensive analysis on these consecutive images to generate a description of the entire video content.
- This "video-to-multi-image" preprocessing is completely encapsulated in the model interface layer and is transparent to the tool layer.

In summary, the general workflow for video analysis is: `ToolsManager` initiates an analysis request -> `model_interface` intercepts the request and calls `_extract_video_frames` to decompose the video file into multiple keyframe images -> `model_interface` sends these images, along with analysis instructions, to the appropriate model platform according to the configuration -> The model returns a comprehensive description of the video -> The result is finally returned to `ToolsManager`. This design ensures the stability and versatility of the upper-layer applications.

3. Practical Operation

3.1 Configuring Offline Large Models

3.1.1 Configuring the LLM Platform (`yahboom.yaml`)

This file determines which large model platform the `model_service` node loads as its primary language model.

1. **Open the file in the terminal**:

```
vim ~/yahboom_ws/src/largemodel/config/yahboom.yaml
```

2. **Modify/confirm `llm_platform`**:

```
model_service:                                  # Model server node parameters
  ros__parameters:
    language: 'en'                              # Large model interface language
    useolinetts: True                           # This option is invalid in text mode
and can be ignored

    # Large model configuration
    llm_platform: 'ollama'                      # Critical: Ensure this is 'ollama'
```

3.1.2 Configure the model interface (`large_model_interface.yaml`)

This file defines which visual model is used when the platform is selected as `ollama`.

1. Open the file in the terminal

```
vim ~/yahboom_ws/src/largemodel/config/large_model_interface.yaml
```

2. Locate the ollama-related configuration

```
#.....
## Offline Large Language Models
# Ollama Configuration
ollama_host: "http://localhost:11434" # Ollama server address
ollama_model: "llava" # Key: Replace this with your downloaded multimodal model,
such as "llava"
#.....
```

Note: Ensure that the model specified in the configuration parameters (such as `llava`) can handle multimodal input.

3.1.3 Recompile

```
cd ~/yahboom_ws/
colcon build
source install/setup.bash
```

3.2 Startup and Testing

1. **Prepare Video File**:

Place a video file to be tested in the following path: `/home/sunrise/yahboom_ws/src/largemodel/resources_file/analyze_video`

Then name the video `test_video.mp4`

2. **Start the `largemodel` main program**:

Open a terminal and run the following command:

```
ros2 launch largemodel largemodel_control.launch.py
```

3. **Test**:

- **Wake-up**: Say into the microphone, "Hello,yahboom."
- **Dialogue**: After the speaker responds, you can say: `Analyze the video content`
- **Observe the logs**: In the terminal running the `launch` file, you should see:

1. The ASR node recognizes your question and prints it out.
2. The `model_service` node receives the text, calls the LLM, and prints the LLM's response.

- **Listen to the response:** Shortly after, you should hear the response from the speaker.