ComfyBench: Benchmarking LLM-based Agents in ComfyUI for Autonomously Designing Collaborative AI Systems

Supplementary Material

Overview

This supplementary document provides additional details to support our main manuscript, organized as follows:

- Section A presents more examples of task instructions that are included in ComfyBench.
- Section B provides more examples generated by ComfyAgent on ComfyBench, as well as a typical trajectory of ComfyAgent to solve a task on ComfyBench.
- Section C summarizes the detailed implementation of GPT-4o evaluaiton for resolve rate computation.
- Section D demonstrates the details of human evaluation, certifying the stability of our VLM evaluation system.
- Section E showcases the detailed prompt implementation for each agent in the ComfyAgent framework.
- Section F provides the details of inference parameters.
- Section G presents some metrics on the generation quality of ComfyAgent compared with single-model methods.
- Section H presents the performance comparison between ComfyAgent and some common multi-agent methods.

A. More Task Instructions

In Table 3, we present more examples of task instruction to provide a deeper preview of ComfyBench. The modalities and categories of the tasks are also included. ComfyBench covers a wide range of image and video generation tasks. Each instruction describes an expected result to realize. The tasks are categorized into three difficulty levels: *vanilla*, *complex*, and *creative*, which reflect the generalization capability of LLM-based agents.

B. More ComfyAgent Examples

As an extension, we present more examples generated by ComfyAgent on ComfyBench in Table 4 to demonstrate the fantastic effects that ComfyAgent can achieve. Restricted by the format, we only present the image examples. Complete examples, including those involving videos as inputs or results, are available on our project website: https://xxyqwq.github.io/ComfyBench.

Furthermore, we present a typical example trajectory of ComfyAgent in Figure 8, which specifically demonstrates how ComfyAgent gradually designs a complete workflow for the collaborative AI system according to the task instruction. The task is selected from the complex category in ComfyBench, which requires generating a video and interpolating the frames, involving common techniques such as text-to-video generation and video interpolation.

C. VLM-based Evaluation Implementation

In ComfyBench, we adopt the latest GPT-4o model to compute the resolve rate. We design prompt templates for each task modality, where the images and videos can be encoded. Considering GPT-4o cannot directly process videos, we uniformly sample up to 10 frames and input them as a sequence of images, so that the context length can be controlled within a reasonable range. Since the tasks in ComfyBench involve operations such as upscaling and interpolation, we also provide the model with the original resolution and frame rate information. The model must conduct the necessary analysis before providing the result so that the judgment can be more reasonable and reliable.

Specifically, we present a typical example of VLM-based evaluation in Figure 3, which evaluates a text-to-image generation task. The answer given by GPT-4o involves a thorough analysis of the consistency between the task requirements and the generated image, followed by a correct final judgment, demonstrating a strong capability to understand the points and determine the consistency.

D. Details for Human Evaluation

We select 50 tasks from ComfyBench that are completed by ComfyAgent. For the rigor of the human evaluation, we choose additional 20 tasks from ComfyBench that are not completed by ComfyAgent and manually construct their generation results. Therefore, a total of 70 questions are formed finally. In each question, we provide the task instruction, the input image or video (if any), and the generated image or video, where the answer should be either "Yes" or "No", representing whether the generated result is consistent with the task instruction. We create multiple questionnaires in Google Forms for distribution, each containing 20 questions. A sample question selected from the questionnaires is presented in Figure 1.

We collect 28 answer sheets from human evaluators to form a sample size of 560, with each question answered by 8 human evaluators. We also prompt the latest GPT-4o model to sample 8 answers for each question. All the answers are viewed as $0/1$ variables to compute the average scores given by human evaluators and GPT-4o respectively, which indicate their tendency on each question.

Based on the 70 pairs of average scores, we present a heatmap in Figure 2 to intuitively demonstrate the correlation between the scores given by human evaluators and GPT-4o. We also calculate Kendall's τ, Pearson's r, and

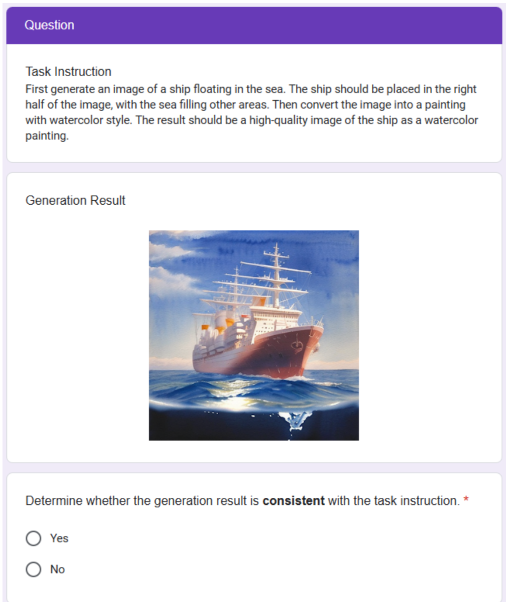


Figure 1. A sample question selected from the created questionnaires on Google Forms in the human evaluation.

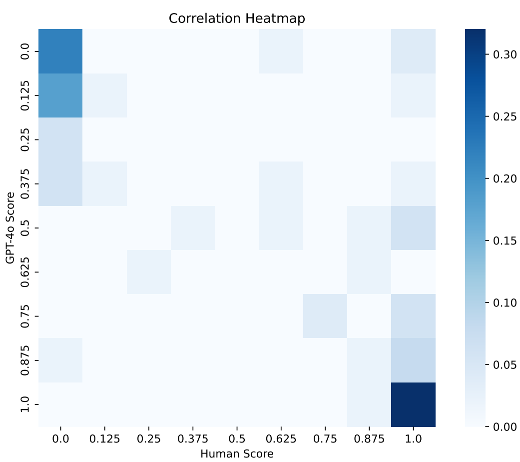


Figure 2. Correlation heatmap of the scores given by human evaluators and GPT-4o in the human evaluation.

Spearman's ρ. Both the heatmap and the statistics indicate a strong agreement between human evaluators and GPT-4o.

E. ComfyAgent Prompt Implementation

We provide the prompt implementation of ComfyAgent, including the prompt templates for PlanAgent, CombineAgent, AdaptAgent, and RefineAgent, which are respectively presented in Figure 4, 5, 6, and 7. Generally, we first introduce the background information of the ComfyUI platform, then present the necessary information for the current task, and finally specify the answer format. We follow the prompting strategy in CodeAct, which encloses answers with XML tags, to extract the results. The memory and RetrieveAgent are implemented as rule-based modules, so prompt templates are not available for them.

F. Details for Inference Parameters

Taking into account the variations among different LLMs, we retain most parameters, such as context length and probability threshold, at their default settings, allowing each model to fully utilize its capabilities. Given that the implementation of various agents relies on multiple samplings, the temperature parameter is set to 1.0 to balance reliability with creativity. Meanwhile, for stable VLM-based evaluations, the temperature parameter is fixed at 0.

G. Metrics for Generation Quality

We use DALL-E 3 and InstructPix2Pix as baseline single-model methods for image generation and editing respectively. ComfyAgent completes 112 out of 200 tasks on ComfyBench, while DALL-E 3 and InstructPix2Pix only complete 44 and 20 tasks respectively. The HPSv2 and Aesthetic Score on the available results are shown in Table 1, demonstrating that ComfyAgent can achieve similar generation quality compared to common single-model methods.

Table 1. Comparison with common single-model methods.

Method	Image Generation		Image Editing	
	HPSv2	Aesthetic	HPSv2	Aesthetic
DALL-E 3	0.299	6.212	-	-
InstructPix2Pix	-	-	0.225	5.078
ComfyAgent	0.289	6.353	0.197	4.976

H. Comparison with Multi-agent Methods

We select CAMEL and Multi-agent Debate (MAD) as baseline multi-agent approaches, as they can be seamlessly adapted to address the tasks in ComfyBench. The evaluation results, summarized in Table 2, demonstrate that ComfyAgent significantly outperforms these baseline methods, underscoring its effectiveness in autonomously designing collaborative AI systems.

Table 2. Comparison with other multi-agent frameworks.

Method	Vanilla		Complex		Creative	
	%P	%R	%P	%R	%P	%R
CAMEL	23.0	14.0	13.3	3.3	5.0	0.0
MAD	49.0	23.0	28.3	5.0	30.0	2.5
ComfyAgent	**67.0**	**46.0**	**48.3**	**21.7**	**40.0**	**15.0**

Table 3. More examples of task instruction in ComfyBench. We present the instructions together with their modalities and categories.

Task Instruction	**Modality**	**Category**
Generate an image of a hotel room containing a bed, a desk, and a window. The result should be a high-quality image.	T2I	Vanilla
You are given an image "many_people.png", which is a photo of a crowd of people. Create a 2-second video of the people cheering based on the image. The result should be a high-quality video.	I2V	Vanilla
You are given an image "street_car.png" of a car parked on the street. Replace the red car with a green one. The result should be a high-quality image without visible artifacts.	I2I	Vanilla
You are given a video "passing_car.mp4" of a gray car passing by on the road. Interpolate the video to increase the frame rate by 2x. The result should be a smoother video of the gray car passing by.	V2V	Vanilla
First generate an image of a city street at night. Then upscale it by 2x. The result should be a high-resolution image of a city street.	T2I	Complex
First generate a 2-second video of a bustling city at night with skyscrapers and bright lights. Then interpolate the video to increase the frame rate by 3x. The result should be a smoother video of the city at night.	T2V	Complex
You are given an image "mountain_stream.png" of a stream flowing through a mountain. First remove the train near the stream. Then convert the image into a painting with watercolor style. The result should be a high-quality image of the stream as a watercolor painting.	I2I	Complex
You are given an image "woman_photo.jpg", which is a photo of a woman smiling. First convert it into a portrait while keeping other details. Then replace the background with a scene of a night street. The result should be an image of the woman as a portrait in a night street.	I2I	Complex
Generate a poster for a series named "Breaking Bad". The result should be a vertical image of the main character, Walter White, wearing a white vest with a serious expression, standing in front of a motorhome in the desert. The title "Breaking Bad" should be displayed in the top center of the image, followed by the tagline "Say my name".	T2I	Creative
You are given an image "cosmetic_product.jpg", which contains two bottles of cosmetic products illuminated by a soft yellow light. Modify the illumination into a bright pink light to create a more vibrant and attractive appearance. The result should be an image of the cosmetic products with the new illumination.	I2I	Creative
You are given an image "warm_bedroom.jpg", which is a photo of a bedroom including a bed, a chair and some decorations. Generate a 3-second video based on the image to show the panoramic view of the bedroom from different angles. The result should be a video that explores the bedroom in a smooth manner.	T2V	Creative
You are given a video "male_idol.mp4" of a male idol dancing in a room. Convert the video into a sketch-style animation with black strokes and white background. The result should be a sketch-style video of the idol dancing while maintaining the original elements and movements.	V2V	Creative

Table 4. More examples produced with the collaborative AI systems designed by ComfyAgent on ComfyBench. We present the task instructions, generated results, along with image or video inputs if they are required in the task.

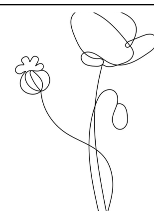
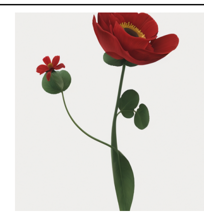
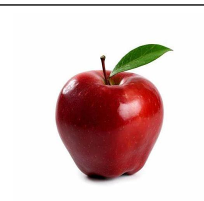
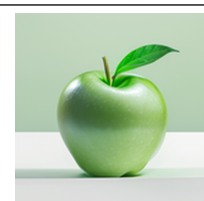
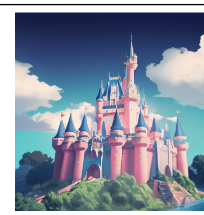
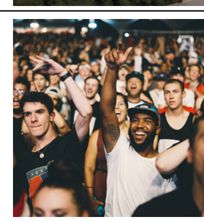
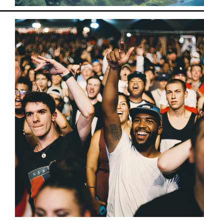
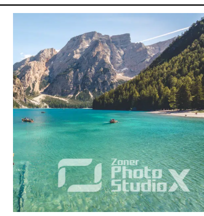
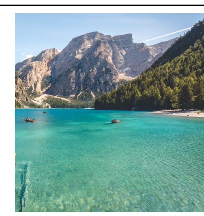
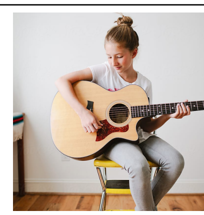
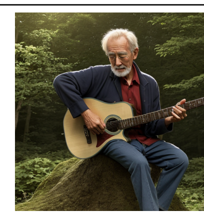
Task Instruction	Input	Result
Generate an image of a hot air balloon floating over a scenic valley at sunrise. The result should be a high-quality image.	N/A	
Generate an image of a modern city skyline at night with illuminated skyscrapers. The result should be a high-quality image.	N/A	
You are given an image of a scribble flower. Repaint the scribble into a realistic red flower. The result should be an image of a red flower.		
You are given an image of a red apple. Change it into a green apple on a table while maintaining other details. The result should be an image of a green apple.		
You are given an image of a sample logo containing a bird pattern. Convert it into a cubist art poster with dark colors. The result should be an image of a poster without watermark.		
You are given an image of a large castle standing on top of a hill. Convert the castle into the style of ice cream while maintaining its original structure. The result should be an image with the castle transformed into a colorful and fantastic ice cream castle.		
You are given a low-resolution photo of a crowd of people. Upscale the image by 4x. The result should be a high-resolution version of the image.		

Table 4. Continued from previous page.

Task Instruction	Input	Result
You are given an image of a table filled with dishes. Remove the fork on the table. The result should be a high-quality image without visible artifacts.		
You are given an image of a red car parked on the street. Replace the tree behind the car with a white house. The result should be a high-quality image without visible artifacts.		
You are given a photo of mountains and rivers with a visible watermark in the bottom right corner. Remove the watermark from the image while maintaining the quality and content of the original photo. The result should be a high-quality image without the watermark.		
You are given an image of a girl playing the guitar. Generate an image of an old man playing the guitar in a forest with the same pose as the girl. The result should be a realistic image of an old man playing the guitar.		
You are given an image of a man wearing a black jacket. Change the black jacket into a white hoodie while ensuring that the modification looks natural and realistic. The result should be a high-quality image of the man wearing a white hoodie.		
You are given an image of a toy dog. Replace the background with a scene of a sunny park with green grass while keeping the lighting and shadows consistent. The result should be an image of the toy dog in the park scene.		
You are given an image containing two bottles of cosmetic products illuminated by a soft yellow light. Modify the illumination into a bright pink light to create a more vibrant and attractive appearance. The result should be an image of the cosmetic products with the new illumination.		

Evaluation Prompt

```
You are an expert in image and video generation, familiar with the latest tasks and techniques. You are capable of understanding the task instruction, analyzing the generation result, and providing an accurate evaluation. Now you are evaluating the result of a text-to-image generation task. You should be tolerant to the quality of the generation result, and focus on the consistency with the instruction.

The task instruction is described as: First generate an image of a ship floating in the sea. The ship should be placed in the right half of the image, with the sea filling other areas. Then convert the image into a painting with watercolor style. The result should be a high-quality image of the ship as a watercolor painting.

The given image is the generation result, with an actual resolution of 512x512.

First, analyze whether the generation result meets each key point in the instruction. Enclose your analysis in the <analysis> tag. For example: <analysis>There is a cat in an astronaut suit, which is consistent with the instruction. The wall is white, which is different from the "green wall" in the instruction.</analysis>.

Then, provide a final judgment of whether the generation result complies with the instruction. The judgment should either be "True" or "False". Enclose your judgment in the <judgment> tag. For example: <judgment>False</judgment>.
```

GPT-4o Answer

```
<analysis>There is a ship floating in the sea, which matches the first part of the instruction. However, the ship is positioned in the center of the image, not on the right half as specified in the instruction. The sea fills a significant part of the image underneath the ship but does not fill the entire left half as required. The painting is shown in a watercolor style, which satisfies the final part of the instruction. Overall, the main issue is the placement of the ship, which should have been shifted to the right side of the image.</analysis>

<judgment>False</judgment>
```

Figure 3. A typical example for VLM-based evaluation in ComfyBench.

PlanAgent Prompt

```
## Task

ComfyUI uses workflows to create and execute Stable Diffusion pipelines so that users can design their own workflows to generate highly customized artworks. ComfyUI workflows can be formulated into the equivalent Python code, where each statement represents the execution of a single node. You are an expert in ComfyUI, helping users to design workflows according to their requirements.

Now you are required to create a ComfyUI workflow to finish the following task:

{instruction}

The core requirements and the expected paradigm are analyzed as follows:

{analysis}

Improving the quality of the generation result with additional steps, such as upscaling and interpolation, is not recommended, unless specified in the requirements.

## Reference

According to the requirements, we have retrieved some relevant workflows which may be helpful:

{reference}

## History

Here is a recent history of your thought, plan and action in the previous steps. The most recent record is at the bottom.

{history}

## Workspace

The code and annotation of the current workflow you are working on are presented as follows:

{workspace}

## Action

Based on the history and workspace, you should first think about what functions have been implemented and what modules remain to be added. Your thought should be enclosed with "<thought>" tag. For example: <thought>The basic pipeline has been implemented, but a module is needed to improve the quality.</thought>.

After that, you should update your step-by-step plan to further modify your workflow. There are {limitation} steps remaining, so your plan should contain at most {limitation} steps. Make sure that each step is feasible to be converted into a single action. Your plan should be enclosed with "<plan>" tag. For example: <plan>Step 1: I will refer to "reference_name" to add a module. Step 2: I will finish the task since the expected effects are realized.</plan>.

Finally, you should choose one of the following actions and specify the arguments (if required), so that the updated workflow can realize the first step in your plan. You should provide your action with the format of function calls in Python. Your action should be enclosed with "<action>" tag. For example: <action>combine(name="reference_name")</action>, <action>adapt(prompt="Change the factor to 0.5 and rewrite the prompt.")</action>, and <action>finish()</action>.

- `load`: Load a reference workflow into the workspace to replace the current workflow, so that you can start over. Arguments:
  - `name`: The name of the reference workflow you want to load.
```

```
- `combine`: Combine the current workflow with a reference workflow, so that necessary modules
↪  can be added. Arguments:
  - `name`: The name of the reference workflow you want to combine.
- `adapt`: Adapt some parameters in the current workflow, so that the expected effects can be
↪  realized. Arguments:
  - `prompt`: The prompt to specify the adaptation you want to make.
- `retrieve`: Retrieve a new batch of reference workflows, so that more useful references can be
↪  found. Arguments:
  - `prompt`: The prompt to describe the reference you want to retrieve.
- `finish`: Finish the task since the current workflow can realize the expected effects.

Refer to the history before making a decision. Here are some general rules you should follow:

1. You should choose the `load` action if and only if the history is empty.
2. If you choose the `load` or `combine` action, make sure the name exists in the reference.
↪  Otherwise, try to update the reference with the `retrieve` action.
3. You should not choose the `adapt` action twice in a row, because they can be simplified into a
↪  single action.
4. If you choose the `adapt` or `retrieve` action, make sure the prompt is concise and contains
↪  all the necessary information.
5. You should choose the `finish` action before the remaining steps count down to 0.

Now, provide your thought, plan and action with the required format.
```

Figure 4. Prompt template for PlanAgent.

CombineAgent Prompt

```
## Task

ComfyUI uses workflows to create and execute Stable Diffusion pipelines so that users can design
↪  their own workflows to generate highly customized artworks. ComfyUI workflows can be
↪  formulated into the equivalent Python code, where each statement represents the execution of
↪  a single node. You are an expert in ComfyUI, helping users to design workflows according to
↪  their requirements.

Now you are required to create a ComfyUI workflow to finish the following task:

{instruction}

The core requirements and the expected paradigm are analyzed as follows:

{analysis}

## Reference

The code and annotation of the current workflow you are referring to are presented as follows:

{reference}

## Workspace

The code and annotation of the current workflow you are working on are presented as follows:

{workspace}

## Combination

Based on the current working progress, your schedule is presented as follows:

{schedule}

You are working on the first step of your schedule. In other words, you should combine the
↪  reference workflow with the current workflow according to your schedule.

First, you should provide your Python code to formulate the updated workflow. Each line of code
↪  should correspond to a single node, so you should avoid nested calls in a single statement.
↪  You should also avoid reusing the same variable name, even if the variable is temporary.
↪  Your code should be enclosed with "<code>" tag. For example: <code>output =
↪  node(input)</code>.

After that, you should provide an annotation as in the reference, including the function and
↪  principle of the updated workflow. The function should be enclosed with "<function>" tag.
↪  For example: <function>This workflow generates a high-resolution image of a running
↪  horse.</function>. The principle should be enclosed with "<principle>" tag. For example:
↪  <principle>The workflow first generates a low-resolution image using the text-to-image
↪  pipeline and then applies an upscaling module to improve the resolution.</principle>.

Now, provide your code and annotation with the required format.
```

Figure 5. Prompt template for CombineAgent.

AdaptAgent Prompt

```
## Task

ComfyUI uses workflows to create and execute Stable Diffusion pipelines so that users can design their own workflows to generate highly customized artworks. ComfyUI workflows can be formulated into the equivalent Python code, where each statement represents the execution of a single node. You are an expert in ComfyUI, helping users to design workflows according to their requirements.

Now you are required to create a ComfyUI workflow to finish the following task:

{instruction}

The core requirements and the expected paradigm are analyzed as follows:

{analysis}

## Workspace

The code and annotation of the current workflow you are working on are presented as follows:

{workspace}

## Adaptation

Based on the current working progress, your schedule is presented as follows:

{schedule}

You are working on the first step of your schedule. In other words, you should modify the parameters in the current workflow according to your schedule. The adaptation you want to make is specified as follows:

{adaptation}

First, you should provide your Python code to formulate the updated workflow. Each line of code should correspond to a single node, so you should avoid nested calls in a single statement. You should also avoid reusing the same variable name, even if the variable is temporary. Your code should be enclosed with "<code>" tag. For example: <code>output = node(input)</code>.

After that, you should provide an annotation as in the reference, including the function and principle of the updated workflow. The function should be enclosed with "<function>" tag. For example: <function>This workflow generates a high-resolution image of a running horse.</function>. The principle should be enclosed with "<principle>" tag. For example: <principle>The workflow first generates a low-resolution image using the text-to-image pipeline and then applies an upscaling module to improve the resolution.</principle>.

Now, provide your code and annotation with the required format.
```

Figure 6. Prompt template for AdaptAgent.

RefineAgent Prompt

```
## Task

ComfyUI uses workflows to create and execute Stable Diffusion pipelines so that users can design
↪  their own workflows to generate highly customized artworks. ComfyUI workflows can be
↪  formulated into the equivalent Python code, where each statement represents the execution of
↪  a single node. You are an expert in ComfyUI, helping users to design workflows according to
↪  their requirements.

Now you are required to create a ComfyUI workflow to finish the following task:

{instruction}

The core requirements and the expected paradigm are analyzed as follows:

{analysis}

## Reference

According to the requirements, we have retrieved some relevant workflows which may be helpful:

{reference}

## Workspace

The code and annotation of the current workflow you are working on are presented as follows:

{workspace}

## Refinement

An error is detected in the current workflow, which is caused by some bugs in the Python code,
↪  such as nested calls and missing parameters. The specific error message is presented as
↪  follows:

{refinement}

First, you should explain the reason of the error. Your explanation should be enclosed with
↪  "<explanation>" tag. For example: <explanation>The error is caused by the missing
↪  input.</explanation>.

After that, you should provide the corrected Python code to formulate the updated workflow. Each
↪  line of code should correspond to a single node, so you should avoid nested calls in a
↪  single statement. You should also avoid reusing the same variable name, even if the variable
↪  is temporary. Your code should be enclosed with "<code>" tag. For example: <code>output =
↪  node(input)</code>.

Finally, you should provide an annotation as in the reference, including the function and
↪  principle of the updated workflow. The function should be enclosed with "<function>" tag.
↪  For example: <function>This workflow generates a high-resolution image of a running
↪  horse.</function>. The principle should be enclosed with "<principle>" tag. For example:
↪  <principle>The workflow first generates a low-resolution image using the text-to-image
↪  pipeline and then applies an upscaling module to improve the resolution.</principle>.

Now, provide your explanation, code, and annotation with the required format.
```

Figure 7. Prompt template for RefineAgent.

Task Instruction

```
First generate a 2-second video of colorful fish swimming in an aquarium with bubbles rising to
↪  the surface. Then interpolate the video to increase the frame rate by 3x. The result should
↪  be a smoother video of the swimming fish.
```

PlanAgent Prompt

```
## Task

ComfyUI uses workflows to create and execute Stable Diffusion pipelines so that users can design
↪  their own workflows to generate highly customized artworks. ComfyUI workflows can be
↪  formulated into the equivalent Python code, where each statement represents the execution of
↪  a single node. You are an expert in ComfyUI, helping users to design workflows according to
↪  their requirements.

Now you are required to create a ComfyUI workflow to finish the following task:

First generate a 2-second video of colorful fish swimming in an aquarium with bubbles rising to
↪  the surface. Then interpolate the video to increase the frame rate by 3x. The result should
↪  be a smoother video of the swimming fish.

The core requirements and the expected paradigm are analyzed as follows:

The task requires generating a short 2-second video of fish swimming in an aquarium with bubbles
↪  rising using a text-to-video approach, as the primary input is textual. The video needs to
↪  depict dynamic, colorful fish movements with the additional visual element of rising
↪  bubbles. After generating this video, the frame rate needs to be increased by 3x using a
↪  video interpolation process to achieve smoother motion. Thus, the workflow core involves two
↪  stages, first, synthesizing the video from a text prompt using a text-to-video node or
↪  series of nodes, followed by applying video interpolation to enhance frame smoothness. There
↪  are no specific quality-oriented steps like upscaling or refinement beyond the interpolation
↪  requirement.

Improving the quality of the generation result with additional steps, such as upscaling and
↪  interpolation, is not recommended, unless specified in the requirements.

## Reference

According to the requirements, we have retrieved some relevant workflows which may be helpful:

- Example: text_to_video

<function>
This workflow follows a text-to-video paradigm, where it generates a video from a given text
↪  prompt. It first creates an initial image based on the text description and then uses that
↪  image as the starting frame to generate a video with motion and transformation using a
↪  Stable Video Diffusion model. The output is a 3-second video at 8 frames per second,
↪  depicting beautiful scenery with mountains, rivers, and clouds.
</function>

<principle>
The workflow first loads two models: a Stable Diffusion model for generating the initial image
↪  from the text ("sd_xl_base_1.0.safetensors") and a Stable Video Diffusion model
↪  ("svd_xt_1_1.safetensors") for video generation. It uses the text description to create
↪  conditioning, generating an initial 1024x576 image. The image is decoded from latent space
↪  via a VAE. The video-specific node ("SVD_img2vid_Conditioning") then applies continuity and
↪  motion to this image, producing conditioned latent representations for video generation. The
↪  final video is created by sampling the latent space over multiple frames and combining them
↪  into an MP4 video using the specified frame rate and format.
</principle>

- Example: video_frame_interpolation
```

```
<function>
This workflow performs video frame interpolation using the RIFE VFI model. It takes an input video such as "play_guitar.gif", increases the frame rate by generating intermediate frames (interpolating) with a multiplier (in this case, 3x), and produces a smoother video with a higher frame rate (from 8 to 24 frames per second). The final output is saved as a new video or animated GIF.
</function>

<principle>
The workflow first loads the input video using "VHS_LoadVideo", which extracts the individual frames. The "RIFE VFI" node is then used to interpolate the frames by generating additional frames between the existing ones. In this scenario, the multiplier is set to 3x, effectively tripling the frame count and enabling a smoother video playback at 24 frames per second. Finally, the interpolated frames are combined into a video or GIF format using "VHS_VideoCombine".
</principle>

- Example: image_to_video

<function>
This workflow follows an image-to-video paradigm. It requires an input image (in this case, "play_guitar.jpg") and generates a 4-second video at 6 frames per second (24 video frames in total) based on that image. The workflow outputs the generated video.
</function>

<principle>
The workflow uses the "svd_xt_1_1.safetensors" Stable Video Diffusion model to generate a video from the input image "play_guitar.jpg". The "SVD_img2vid_Conditioning" node creates the necessary conditioning for video generation, including the number of frames, resolution, and motion characteristics. A KSamplerAdvanced node adds noise and performs generative sampling over multiple steps to create diverse video frames. These frames are then decoded back into images via a VAE, and finally, the "VHS_VideoCombine" node compiles these images into a 4-second video at 8 frames per second.
</principle>

- Example: text_to_image

<function>
This workflow implements a basic text-to-image generation pipeline using Stable Diffusion. It requires both positive (desired traits) and negative (undesired traits) text prompts to generate an image. In this specific case, the workflow will output a high-resolution photo of a cat wearing a spacesuit inside a spaceship, avoiding blurry or illustration-like effects.
</function>

<principle>
The workflow begins by loading the "dreamshaper_8.safetensors" Stable Diffusion model. It generates a blank latent space as the starting point for the image generation. The positive prompt ("a photo of a cat wearing a spacesuit inside a spaceship") and negative prompt ("blurry, illustration") are encoded into conditioning by the CLIPTextEncode node. The KSampler node then uses these conditionings to guide the generative process over 20 sampling steps, applying denoising to the latent space. The resulting latent code is subsequently decoded into an image using the VAE and saved to disk.
</principle>

- Example: scribble_image_repaint

<function>
This workflow follows a sketch-to-image paradigm, where it takes a scribble image ("simple_graffiti.png") along with a text prompt ("a bird, open wings") and generates a detailed, high-quality image based on both the scribble and the prompt.
</function>

<principle>
```

```
The workflow first loads and inverts the input scribble image. The inverted image is then used
↪  by a "control_v11p_sd15_scribble_fp16" ControlNet model to extract and applied to the
↪  scribble, controlling and guiding the image generation process. A pre-trained model
↪  ("dreamshaper_8.safetensors") processes the positive conditioning generated by the
↪  combination of the text prompt and the ControlNet's output, along with negative conditioning
↪  from undesired traits such as "horror" and "lowres." The latent space image is generated
↪  using a KSampler and further decoded by the VAE to form the final image. The model
↪  synthesizes a highly detailed image while adhering closely to both the shape of the scribble
↪  and the desired features implied by the text prompt.
</principle>

## History

Here is a recent history of your thought, plan and action in the previous steps. The most recent
↪  record is at the bottom.

- The history is empty.

## Workspace

The code and annotation of the current workflow you are working on are presented as follows:

- The workspace is empty.

## Action

Based on the history and workspace, you should first think about what functions have been
↪  implemented and what modules remain to be added. Your thought should be enclosed with
↪  "<thought>" tag. For example: <thought>The basic pipeline has been implemented, but a module
↪  is needed to improve the quality.</thought>.

After that, you should update your step-by-step plan to further modify your workflow. There are
↪  5 steps remaining, so your plan should contain at most 5 steps. Make sure that each step is
↪  feasible to be converted into a single action. Your plan should be enclosed with "<plan>"
↪  tag. For example: <plan>Step 1: I will refer to "reference_name" to add a module. Step 2: I
↪  will finish the task since the expected effects are realized.</plan>.

Finally, you should choose one of the following actions and specify the arguments (if required),
↪  so that the updated workflow can realize the first step in your plan. You should provide
↪  your action with the format of function calls in Python. Your action should be enclosed with
↪  "<action>" tag. For example: <action>combine(name="reference_name")</action>,
↪  <action>adapt(prompt="Change the factor to 0.5 and rewrite the prompt.")</action>, and
↪  <action>finish()</action>.

- `load`: Load a reference workflow into the workspace to replace the current workflow, so that
↪  you can start over. Arguments:
  - `name`: The name of the reference workflow you want to load.
- `combine`: Combine the current workflow with a reference workflow, so that necessary modules
↪  can be added. Arguments:
  - `name`: The name of the reference workflow you want to combine.
- `adapt`: Adapt some parameters in the current workflow, so that the expected effects can be
↪  realized. Arguments:
  - `prompt`: The prompt to specify the adaptation you want to make.
- `retrieve`: Retrieve a new batch of reference workflows, so that more useful references can be
↪  found. Arguments:
  - `prompt`: The prompt to describe the reference you want to retrieve.
- `finish`: Finish the task since the current workflow can realize the expected effects.

Refer to the history before making a decision. Here are some general rules you should follow:

1. You should choose the `load` action if and only if the history is empty.
2. If you choose the `load` or `combine` action, make sure the name exists in the reference.
↪  Otherwise, try to update the reference with the `retrieve` action.
3. You should not choose the `adapt` action twice in a row, because they can be simplified into a
↪  single action.
```

```
4. If you choose the `adapt` or `retrieve` action, make sure the prompt is concise and contains all the necessary information.
5. You should choose the `finish` action before the remaining steps count down to 0.

Now, provide your thought, plan and action with the required format.
```

PlanAgent Answer

```
<thought>Since the history is empty, I need to start by loading a relevant reference workflow that can be used as a base for generating the 2-second video of colorful fish in an aquarium. The "text_to_video" reference seems appropriate to use as it involves generating video from text prompts, which aligns with the first part of the task.</thought>

<plan>Step 1: I will load the "text_to_video" reference workflow to use it as the base.
Step 2: I will adapt the workflow to modify the prompt to match the requirements ("colorful fish swimming in an aquarium with bubbles rising").
Step 3: I will combine it with the "video_frame_interpolation" reference to enable interpolation for smoother video.
Step 4: I will adapt the interpolation multiplier to 3x to achieve the expected frame rate increase.
Step 5: I will finish the task by confirming the generated output meets the required specifications.</plan>

<action>load(name="text_to_video")</action>
```

PlanAgent Prompt

```
## Task

ComfyUI uses workflows to create and execute Stable Diffusion pipelines so that users can design their own workflows to generate highly customized artworks. ComfyUI workflows can be formulated into the equivalent Python code, where each statement represents the execution of a single node. You are an expert in ComfyUI, helping users to design workflows according to their requirements.

Now you are required to create a ComfyUI workflow to finish the following task:

First generate a 2-second video of colorful fish swimming in an aquarium with bubbles rising to the surface. Then interpolate the video to increase the frame rate by 3x. The result should be a smoother video of the swimming fish.

The core requirements and the expected paradigm are analyzed as follows:

The task requires generating a short 2-second video of fish swimming in an aquarium with bubbles rising using a text-to-video approach, as the primary input is textual. The video needs to depict dynamic, colorful fish movements with the additional visual element of rising bubbles. After generating this video, the frame rate needs to be increased by 3x using a video interpolation process to achieve smoother motion. Thus, the workflow core involves two stages, first, synthesizing the video from a text prompt using a text-to-video node or series of nodes, followed by applying video interpolation to enhance frame smoothness. There are no specific quality-oriented steps like upscaling or refinement beyond the interpolation requirement.

Improving the quality of the generation result with additional steps, such as upscaling and interpolation, is not recommended, unless specified in the requirements.

## Reference

According to the requirements, we have retrieved some relevant workflows which may be helpful:
```

```
- Example: text_to_video

<function>
This workflow follows a text-to-video paradigm, where it generates a video from a given text
↪  prompt. It first creates an initial image based on the text description and then uses that
↪  image as the starting frame to generate a video with motion and transformation using a
↪  Stable Video Diffusion model. The output is a 3-second video at 8 frames per second,
↪  depicting beautiful scenery with mountains, rivers, and clouds.
</function>

<principle>
The workflow first loads two models: a Stable Diffusion model for generating the initial image
↪  from the text ("sd_xl_base_1.0.safetensors") and a Stable Video Diffusion model
↪  ("svd_xt_1_1.safetensors") for video generation. It uses the text description to create
↪  conditioning, generating an initial 1024x576 image. The image is decoded from latent space
↪  via a VAE. The video-specific node ("SVD_img2vid_Conditioning") then applies continuity and
↪  motion to this image, producing conditioned latent representations for video generation. The
↪  final video is created by sampling the latent space over multiple frames and combining them
↪  into an MP4 video using the specified frame rate and format.
</principle>

- Example: video_frame_interpolation

<function>
This workflow performs video frame interpolation using the RIFE VFI model. It takes an input
↪  video such as "play_guitar.gif", increases the frame rate by generating intermediate frames
↪  (interpolating) with a multiplier (in this case, 3x), and produces a smoother video with a
↪  higher frame rate (from 8 to 24 frames per second). The final output is saved as a new video
↪  or animated GIF.
</function>

<principle>
The workflow first loads the input video using "VHS_LoadVideo", which extracts the individual
↪  frames. The "RIFE VFI" node is then used to interpolate the frames by generating additional
↪  frames between the existing ones. In this scenario, the multiplier is set to 3x, effectively
↪  tripling the frame count and enabling a smoother video playback at 24 frames per second.
↪  Finally, the interpolated frames are combined into a video or GIF format using
↪  "VHS_VideoCombine".
</principle>

- Example: image_to_video

<function>
This workflow follows an image-to-video paradigm. It requires an input image (in this case,
↪  "play_guitar.jpg") and generates a 4-second video at 6 frames per second (24 video frames in
↪  total) based on that image. The workflow outputs the generated video.
</function>

<principle>
The workflow uses the "svd_xt_1_1.safetensors" Stable Video Diffusion model to generate a video
↪  from the input image "play_guitar.jpg". The "SVD_img2vid_Conditioning" node creates the
↪  necessary conditioning for video generation, including the number of frames, resolution, and
↪  motion characteristics. A KSamplerAdvanced node adds noise and performs generative sampling
↪  over multiple steps to create diverse video frames. These frames are then decoded back into
↪  images via a VAE, and finally, the "VHS_VideoCombine" node compiles these images into a
↪  4-second video at 8 frames per second.
</principle>

- Example: text_to_image

<function>
```

```
This workflow implements a basic text-to-image generation pipeline using Stable Diffusion. It requires both positive (desired traits) and negative (undesired traits) text prompts to generate an image. In this specific case, the workflow will output a high-resolution photo of a cat wearing a spacesuit inside a spaceship, avoiding blurry or illustration-like effects.
</function>

<principle>
The workflow begins by loading the "dreamshaper_8.safetensors" Stable Diffusion model. It generates a blank latent space as the starting point for the image generation. The positive prompt ("a photo of a cat wearing a spacesuit inside a spaceship") and negative prompt ("blurry, illustration") are encoded into conditioning by the CLIPTextEncode node. The KSampler node then uses these conditionings to guide the generative process over 20 sampling steps, applying denoising to the latent space. The resulting latent code is subsequently decoded into an image using the VAE and saved to disk.
</principle>

- Example: scribble_image_repaint

<function>
This workflow follows a sketch-to-image paradigm, where it takes a scribble image ("simple_graffiti.png") along with a text prompt ("a bird, open wings") and generates a detailed, high-quality image based on both the scribble and the prompt.
</function>

<principle>
The workflow first loads and inverts the input scribble image. The inverted image is then used by a "control_v11p_sd15_scribble_fp16" ControlNet model to extract and applied to the scribble, controlling and guiding the image generation process. A pre-trained model ("dreamshaper_8.safetensors") processes the positive conditioning generated by the combination of the text prompt and the ControlNet's output, along with negative conditioning from undesired traits such as "horror" and "lowres." The latent space image is generated using a KSampler and further decoded by the VAE to form the final image. The model synthesizes a highly detailed image while adhering closely to both the shape of the scribble and the desired features implied by the text prompt.
</principle>

## History

Here is a recent history of your thought, plan and action in the previous steps. The most recent record is at the bottom.

- Step: 1

<thought>
Since the history is empty, I need to start by loading a relevant reference workflow that can be used as a base for generating the 2-second video of colorful fish in an aquarium. The "text_to_video" reference seems appropriate to use as it involves generating video from text prompts, which aligns with the first part of the task.
</thought>

<plan>
Step 1: I will load the "text_to_video" reference workflow to use it as the base.
Step 2: I will adapt the workflow to modify the prompt to match the requirements ("colorful fish swimming in an aquarium with bubbles rising").
Step 3: I will combine it with the "video_frame_interpolation" reference to enable interpolation for smoother video.
Step 4: I will adapt the interpolation multiplier to 3x to achieve the expected frame rate increase.
Step 5: I will finish the task by confirming the generated output meets the required specifications.
</plan>

<action>
load(name="text_to_video")
```

```
</action>

## Workspace

The code and annotation of the current workflow you are working on are presented as follows:

<code>
model_15, clip_vision_15, vae_15 =
↪  ImageOnlyCheckpointLoader(ckpt_name="""svd_xt_1_1.safetensors""")
model_16, clip_16, vae_16 = CheckpointLoaderSimple(ckpt_name="""sd_xl_base_1.0.safetensors""")
conditioning_18 = CLIPTextEncode(text="""photograph beautiful scenery nature mountains alps
↪  river rapids snow sky cumulus clouds""", speak_and_recognation=True, clip=clip_16)
conditioning_19 = CLIPTextEncode(text="""text, watermark""", speak_and_recognation=True,
↪  clip=clip_16)
latent_22 = EmptyLatentImage(width=1024, height=576, batch_size=1)
model_14 = VideoLinearCFGGuidance(min_cfg=1, model=model_15)
latent_17 = KSampler(seed=307393744025667, steps=15, cfg=8, sampler_name="""uni_pc_bh2""",
↪  scheduler="""normal""", denoise=1, model=model_16, positive=conditioning_18,
↪  negative=conditioning_19, latent_image=latent_22)
image_20 = VAEDecode(samples=latent_17, vae=vae_16)
_ = PreviewImage(images=image_20)
positive_12, negative_12, latent_12 = SVD_img2vid_Conditioning(width=1024, height=576,
↪  video_frames=24, motion_bucket_id=127, fps=8, augmentation_level=0,
↪  clip_vision=clip_vision_15, init_image=image_20, vae=vae_15)
latent_3 = KSampler(seed=1103641334004632, steps=20, cfg=2.5, sampler_name="""euler""",
↪  scheduler="""karras""", denoise=1, model=model_14, positive=positive_12,
↪  negative=negative_12, latent_image=latent_12)
image_8 = VAEDecode(samples=latent_3, vae=vae_15)
filenames_23 = VHS_VideoCombine(frame_rate=6, loop_count=0, filename_prefix="""SVD_txt2vid""",
↪  format="""video/h264-mp4""", pix_fmt="""yuv420p""", crf=19, save_metadata=True,
↪  pingpong=False, save_output=True, images=image_8)

</code>

<function>
This workflow follows a text-to-video paradigm, where it generates a video from a given text
↪  prompt. It first creates an initial image based on the text description and then uses that
↪  image as the starting frame to generate a video with motion and transformation using a
↪  Stable Video Diffusion model. The output is a 3-second video at 8 frames per second,
↪  depicting beautiful scenery with mountains, rivers, and clouds.
</function>

<principle>
The workflow first loads two models: a Stable Diffusion model for generating the initial image
↪  from the text ("sd_xl_base_1.0.safetensors") and a Stable Video Diffusion model
↪  ("svd_xt_1_1.safetensors") for video generation. It uses the text description to create
↪  conditioning, generating an initial 1024x576 image. The image is decoded from latent space
↪  via a VAE. The video-specific node ("SVD_img2vid_Conditioning") then applies continuity and
↪  motion to this image, producing conditioned latent representations for video generation. The
↪  final video is created by sampling the latent space over multiple frames and combining them
↪  into an MP4 video using the specified frame rate and format.
</principle>

## Action

Based on the history and workspace, you should first think about what functions have been
↪  implemented and what modules remain to be added. Your thought should be enclosed with
↪  "<thought>" tag. For example: <thought>The basic pipeline has been implemented, but a module
↪  is needed to improve the quality.</thought>.

After that, you should update your step-by-step plan to further modify your workflow. There are
↪  4 steps remaining, so your plan should contain at most 4 steps. Make sure that each step is
↪  feasible to be converted into a single action. Your plan should be enclosed with "<plan>"
↪  tag. For example: <plan>Step 1: I will refer to "reference_name" to add a module. Step 2: I
↪  will finish the task since the expected effects are realized.</plan>.
```

```
Finally, you should choose one of the following actions and specify the arguments (if required),
↪  so that the updated workflow can realize the first step in your plan. You should provide
↪  your action with the format of function calls in Python. Your action should be enclosed with
↪  "<action>" tag. For example: <action>combine(name="reference_name")</action>,
↪  <action>adapt(prompt="Change the factor to 0.5 and rewrite the prompt.")</action>, and
↪  <action>finish()</action>.

- `load`: Load a reference workflow into the workspace to replace the current workflow, so that
↪  you can start over. Arguments:
  - `name`: The name of the reference workflow you want to load.
- `combine`: Combine the current workflow with a reference workflow, so that necessary modules
↪  can be added. Arguments:
  - `name`: The name of the reference workflow you want to combine.
- `adapt`: Adapt some parameters in the current workflow, so that the expected effects can be
↪  realized. Arguments:
  - `prompt`: The prompt to specify the adaptation you want to make.
- `retrieve`: Retrieve a new batch of reference workflows, so that more useful references can be
↪  found. Arguments:
  - `prompt`: The prompt to describe the reference you want to retrieve.
- `finish`: Finish the task since the current workflow can realize the expected effects.

Refer to the history before making a decision. Here are some general rules you should follow:

1. You should choose the `load` action if and only if the history is empty.
2. If you choose the `load` or `combine` action, make sure the name exists in the reference.
↪  Otherwise, try to update the reference with the `retrieve` action.
3. You should not choose the `adapt` action twice in a row, because they can be simplified into a
↪  single action.
4. If you choose the `adapt` or `retrieve` action, make sure the prompt is concise and contains
↪  all the necessary information.
5. You should choose the `finish` action before the remaining steps count down to 0.

Now, provide your thought, plan and action with the required format.
```

PlanAgent Answer

```
<thought>
The current workflow generates a 3-second video of a landscape scene, but this is not aligned
↪  with the task of generating a 2-second video of colorful fish swimming in an aquarium with
↪  bubbles rising. I need to adapt the text prompt accordingly to match the requirements.
↪  Additionally, the frame rate is currently 8 frames per second, but after generating the
↪  video, we will need to interpolate it to increase the frame rate by 3x to make the motion
↪  smoother.

The basic pipeline for generating videos from a text prompt is already functional, but the
↪  correct text needs to be provided, and the next step would be to apply video frame
↪  interpolation. I will combine this workflow with the "video_frame_interpolation" to achieve
↪  smoother motion.
</thought>

<plan>
Step 1: Adapt the text prompt to describe the required scene of "colorful fish swimming in an
↪  aquarium with bubbles rising".
Step 2: Combine the current workflow with the "video_frame_interpolation" reference workflow to
↪  add video interpolation functionality.
Step 3: Adapt the interpolation multiplier to 3x for smoother motion.
Step 4: Finish the task after confirming that the workflow meets the requirements.
</plan>

<action>
adapt(prompt="Change the text prompt to 'colorful fish swimming in an aquarium with bubbles
↪  rising to the surface' and reduce the video duration to 2 seconds.")
```

```
</action>
```

AdaptAgent Prompt

```
## Task

ComfyUI uses workflows to create and execute Stable Diffusion pipelines so that users can design
↪  their own workflows to generate highly customized artworks. ComfyUI workflows can be
↪  formulated into the equivalent Python code, where each statement represents the execution of
↪  a single node. You are an expert in ComfyUI, helping users to design workflows according to
↪  their requirements.

Now you are required to create a ComfyUI workflow to finish the following task:

First generate a 2-second video of colorful fish swimming in an aquarium with bubbles rising to
↪  the surface. Then interpolate the video to increase the frame rate by 3x. The result should
↪  be a smoother video of the swimming fish.

The core requirements and the expected paradigm are analyzed as follows:

The task requires generating a short 2-second video of fish swimming in an aquarium with bubbles
↪  rising using a text-to-video approach, as the primary input is textual. The video needs to
↪  depict dynamic, colorful fish movements with the additional visual element of rising
↪  bubbles. After generating this video, the frame rate needs to be increased by 3x using a
↪  video interpolation process to achieve smoother motion. Thus, the workflow core involves two
↪  stages, first, synthesizing the video from a text prompt using a text-to-video node or
↪  series of nodes, followed by applying video interpolation to enhance frame smoothness. There
↪  are no specific quality-oriented steps like upscaling or refinement beyond the interpolation
↪  requirement.

## Workspace

The code and annotation of the current workflow you are working on are presented as follows:

<code>
model_15, clip_vision_15, vae_15 =
↪  ImageOnlyCheckpointLoader(ckpt_name="""svd_xt_1_1.safetensors""")
model_16, clip_16, vae_16 = CheckpointLoaderSimple(ckpt_name="""sd_xl_base_1.0.safetensors""")
conditioning_18 = CLIPTextEncode(text="""photograph beautiful scenery nature mountains alps
↪  river rapids snow sky cumulus clouds""", speak_and_recognation=True, clip=clip_16)
conditioning_19 = CLIPTextEncode(text="""text, watermark""", speak_and_recognation=True,
↪  clip=clip_16)
latent_22 = EmptyLatentImage(width=1024, height=576, batch_size=1)
model_14 = VideoLinearCFGGuidance(min_cfg=1, model=model_15)
latent_17 = KSampler(seed=307393744025667, steps=15, cfg=8, sampler_name="""uni_pc_bh2""",
↪  scheduler="""normal""", denoise=1, model=model_16, positive=conditioning_18,
↪  negative=conditioning_19, latent_image=latent_22)
image_20 = VAEDecode(samples=latent_17, vae=vae_16)
_ = PreviewImage(images=image_20)
positive_12, negative_12, latent_12 = SVD_img2vid_Conditioning(width=1024, height=576,
↪  video_frames=24, motion_bucket_id=127, fps=8, augmentation_level=0,
↪  clip_vision=clip_vision_15, init_image=image_20, vae=vae_15)
latent_3 = KSampler(seed=1103641334004632, steps=20, cfg=2.5, sampler_name="""euler""",
↪  scheduler="""karras""", denoise=1, model=model_14, positive=positive_12,
↪  negative=negative_12, latent_image=latent_12)
image_8 = VAEDecode(samples=latent_3, vae=vae_15)
filenames_23 = VHS_VideoCombine(frame_rate=6, loop_count=0, filename_prefix="""SVD_txt2vid""",
↪  format="""video/h264-mp4""", pix_fmt="""yuv420p""", crf=19, save_metadata=True,
↪  pingpong=False, save_output=True, images=image_8)

</code>

<function>
```

```
This workflow follows a text-to-video paradigm, where it generates a video from a given text
↪  prompt. It first creates an initial image based on the text description and then uses that
↪  image as the starting frame to generate a video with motion and transformation using a
↪  Stable Video Diffusion model. The output is a 3-second video at 8 frames per second,
↪  depicting beautiful scenery with mountains, rivers, and clouds.
</function>

<principle>
The workflow first loads two models: a Stable Diffusion model for generating the initial image
↪  from the text ("sd_xl_base_1.0.safetensors") and a Stable Video Diffusion model
↪  ("svd_xt_1_1.safetensors") for video generation. It uses the text description to create
↪  conditioning, generating an initial 1024x576 image. The image is decoded from latent space
↪  via a VAE. The video-specific node ("SVD_img2vid_Conditioning") then applies continuity and
↪  motion to this image, producing conditioned latent representations for video generation. The
↪  final video is created by sampling the latent space over multiple frames and combining them
↪  into an MP4 video using the specified frame rate and format.
</principle>

## Adaptation

Based on the current working progress, your schedule is presented as follows:

Step 1: Adapt the text prompt to describe the required scene of "colorful fish swimming in an
↪  aquarium with bubbles rising".
Step 2: Combine the current workflow with the "video_frame_interpolation" reference workflow to
↪  add video interpolation functionality.
Step 3: Adapt the interpolation multiplier to 3x for smoother motion.
Step 4: Finish the task after confirming that the workflow meets the requirements.

You are working on the first step of your schedule. In other words, you should modify the
↪  parameters in the current workflow according to your schedule. The adaptation you want to
↪  make is specified as follows:

Change the text prompt to 'colorful fish swimming in an aquarium with bubbles rising to the
↪  surface' and reduce the video duration to 2 seconds.

First, you should provide your Python code to formulate the updated workflow. Each line of code
↪  should correspond to a single node, so you should avoid nested calls in a single statement.
↪  You should also avoid reusing the same variable name, even if the variable is temporary.
↪  Your code should be enclosed with "<code>" tag. For example: <code>output =
↪  node(input)</code>.

After that, you should provide an annotation as in the reference, including the function and
↪  principle of the updated workflow. The function should be enclosed with "<function>" tag.
↪  For example: <function>This workflow generates a high-resolution image of a running
↪  horse.</function>. The principle should be enclosed with "<principle>" tag. For example:
↪  <principle>The workflow first generates a low-resolution image using the text-to-image
↪  pipeline and then applies an upscaling module to improve the resolution.</principle>.

Now, provide your code and annotation with the required format.
```

AdaptAgent Answer

```
<code>
model_15, clip_vision_15, vae_15 =
↪  ImageOnlyCheckpointLoader(ckpt_name="""svd_xt_1_1.safetensors""")
model_16, clip_16, vae_16 = CheckpointLoaderSimple(ckpt_name="""sd_xl_base_1.0.safetensors""")
conditioning_18 = CLIPTextEncode(text="""colorful fish swimming in an aquarium with bubbles
↪  rising to the surface""", speak_and_recognation=True, clip=clip_16)
conditioning_19 = CLIPTextEncode(text="""text, watermark""", speak_and_recognation=True,
↪  clip=clip_16)
latent_22 = EmptyLatentImage(width=1024, height=576, batch_size=1)
model_14 = VideoLinearCFGGuidance(min_cfg=1, model=model_15)
```

```
latent_17 = KSampler(seed=307393744025667, steps=15, cfg=8, sampler_name="""uni_pc_bh2""",
↪  scheduler="""normal""", denoise=1, model=model_16, positive=conditioning_18,
↪  negative=conditioning_19, latent_image=latent_22)
image_20 = VAEDecode(samples=latent_17, vae=vae_16)
_ = PreviewImage(images=image_20)
positive_12, negative_12, latent_12 = SVD_img2vid_Conditioning(width=1024, height=576,
↪  video_frames=16, motion_bucket_id=127, fps=8, augmentation_level=0,
↪  clip_vision=clip_vision_15, init_image=image_20, vae=vae_15)
latent_3 = KSampler(seed=1103641334004632, steps=20, cfg=2.5, sampler_name="""euler""",
↪  scheduler="""karras""", denoise=1, model=model_14, positive=positive_12,
↪  negative=negative_12, latent_image=latent_12)
image_8 = VAEDecode(samples=latent_3, vae=vae_15)
filenames_23 = VHS_VideoCombine(frame_rate=8, loop_count=0, filename_prefix="""SVD_txt2vid""",
↪  format="""video/h264-mp4""", pix_fmt="""yuv420p""", crf=19, save_metadata=True,
↪  pingpong=False, save_output=True, images=image_8)
</code>

<function>
This workflow generates a 2-second video of colorful fish swimming in an aquarium with bubbles
↪  rising to the surface from a given text description. The video is first synthesized using a
↪  text-to-video diffusion process.
</function>

<principle>
The workflow first loads two models: a Stable Diffusion model ("sd_xl_base_1.0.safetensors") for
↪  generating the initial image from the text description, and then a Stable Video Diffusion
↪  model ("svd_xt_1_1.safetensors") to translate that image into a dynamic video over multiple
↪  frames. Following the text input describing colorful fish and bubbles, the image is
↪  generated and expanded across 16 frames, which corresponds to a duration of 2 seconds at 8
↪  FPS (frames per second). The final MP4 video is constructed and saved along with metadata.
</principle>
```

PlanAgent Prompt

```
## Task

ComfyUI uses workflows to create and execute Stable Diffusion pipelines so that users can design
↪  their own workflows to generate highly customized artworks. ComfyUI workflows can be
↪  formulated into the equivalent Python code, where each statement represents the execution of
↪  a single node. You are an expert in ComfyUI, helping users to design workflows according to
↪  their requirements.

Now you are required to create a ComfyUI workflow to finish the following task:

First generate a 2-second video of colorful fish swimming in an aquarium with bubbles rising to
↪  the surface. Then interpolate the video to increase the frame rate by 3x. The result should
↪  be a smoother video of the swimming fish.

The core requirements and the expected paradigm are analyzed as follows:

The task requires generating a short 2-second video of fish swimming in an aquarium with bubbles
↪  rising using a text-to-video approach, as the primary input is textual. The video needs to
↪  depict dynamic, colorful fish movements with the additional visual element of rising
↪  bubbles. After generating this video, the frame rate needs to be increased by 3x using a
↪  video interpolation process to achieve smoother motion. Thus, the workflow core involves two
↪  stages, first, synthesizing the video from a text prompt using a text-to-video node or
↪  series of nodes, followed by applying video interpolation to enhance frame smoothness. There
↪  are no specific quality-oriented steps like upscaling or refinement beyond the interpolation
↪  requirement.

Improving the quality of the generation result with additional steps, such as upscaling and
↪  interpolation, is not recommended, unless specified in the requirements.
```

```
## Reference

According to the requirements, we have retrieved some relevant workflows which may be helpful:

- Example: text_to_video

<function>
This workflow follows a text-to-video paradigm, where it generates a video from a given text prompt. It first creates an initial image based on the text description and then uses that image as the starting frame to generate a video with motion and transformation using a Stable Video Diffusion model. The output is a 3-second video at 8 frames per second, depicting beautiful scenery with mountains, rivers, and clouds.
</function>

<principle>
The workflow first loads two models: a Stable Diffusion model for generating the initial image from the text ("sd_xl_base_1.0.safetensors") and a Stable Video Diffusion model ("svd_xt_1_1.safetensors") for video generation. It uses the text description to create conditioning, generating an initial 1024x576 image. The image is decoded from latent space via a VAE. The video-specific node ("SVD_img2vid_Conditioning") then applies continuity and motion to this image, producing conditioned latent representations for video generation. The final video is created by sampling the latent space over multiple frames and combining them into an MP4 video using the specified frame rate and format.
</principle>

- Example: video_frame_interpolation

<function>
This workflow performs video frame interpolation using the RIFE VFI model. It takes an input video such as "play_guitar.gif", increases the frame rate by generating intermediate frames (interpolating) with a multiplier (in this case, 3x), and produces a smoother video with a higher frame rate (from 8 to 24 frames per second). The final output is saved as a new video or animated GIF.
</function>

<principle>
The workflow first loads the input video using "VHS_LoadVideo", which extracts the individual frames. The "RIFE VFI" node is then used to interpolate the frames by generating additional frames between the existing ones. In this scenario, the multiplier is set to 3x, effectively tripling the frame count and enabling a smoother video playback at 24 frames per second. Finally, the interpolated frames are combined into a video or GIF format using "VHS_VideoCombine".
</principle>

- Example: image_to_video

<function>
This workflow follows an image-to-video paradigm. It requires an input image (in this case, "play_guitar.jpg") and generates a 4-second video at 6 frames per second (24 video frames in total) based on that image. The workflow outputs the generated video.
</function>

<principle>
The workflow uses the "svd_xt_1_1.safetensors" Stable Video Diffusion model to generate a video from the input image "play_guitar.jpg". The "SVD_img2vid_Conditioning" node creates the necessary conditioning for video generation, including the number of frames, resolution, and motion characteristics. A KSamplerAdvanced node adds noise and performs generative sampling over multiple steps to create diverse video frames. These frames are then decoded back into images via a VAE, and finally, the "VHS_VideoCombine" node compiles these images into a 4-second video at 8 frames per second.
</principle>

- Example: text_to_image

<function>
```

```
This workflow implements a basic text-to-image generation pipeline using Stable Diffusion. It
↪  requires both positive (desired traits) and negative (undesired traits) text prompts to
↪  generate an image. In this specific case, the workflow will output a high-resolution photo
↪  of a cat wearing a spacesuit inside a spaceship, avoiding blurry or illustration-like
↪  effects.
</function>

<principle>
The workflow begins by loading the "dreamshaper_8.safetensors" Stable Diffusion model. It
↪  generates a blank latent space as the starting point for the image generation. The positive
↪  prompt ("a photo of a cat wearing a spacesuit inside a spaceship") and negative prompt
↪  ("blurry, illustration") are encoded into conditioning by the CLIPTextEncode node. The
↪  KSampler node then uses these conditionings to guide the generative process over 20 sampling
↪  steps, applying denoising to the latent space. The resulting latent code is subsequently
↪  decoded into an image using the VAE and saved to disk.
</principle>

- Example: scribble_image_repaint

<function>
This workflow follows a sketch-to-image paradigm, where it takes a scribble image
↪  ("simple_graffiti.png") along with a text prompt ("a bird, open wings") and generates a
↪  detailed, high-quality image based on both the scribble and the prompt.
</function>

<principle>
The workflow first loads and inverts the input scribble image. The inverted image is then used
↪  by a "control_v11p_sd15_scribble_fp16" ControlNet model to extract and applied to the
↪  scribble, controlling and guiding the image generation process. A pre-trained model
↪  ("dreamshaper_8.safetensors") processes the positive conditioning generated by the
↪  combination of the text prompt and the ControlNet's output, along with negative conditioning
↪  from undesired traits such as "horror" and "lowres." The latent space image is generated
↪  using a KSampler and further decoded by the VAE to form the final image. The model
↪  synthesizes a highly detailed image while adhering closely to both the shape of the scribble
↪  and the desired features implied by the text prompt.
</principle>

## History

Here is a recent history of your thought, plan and action in the previous steps. The most recent
↪  record is at the bottom.

- Step: 1

<thought>
Since the history is empty, I need to start by loading a relevant reference workflow that can be
↪  used as a base for generating the 2-second video of colorful fish in an aquarium. The
↪  "text_to_video" reference seems appropriate to use as it involves generating video from text
↪  prompts, which aligns with the first part of the task.
</thought>

<plan>
Step 1: I will load the "text_to_video" reference workflow to use it as the base.
Step 2: I will adapt the workflow to modify the prompt to match the requirements ("colorful fish
↪  swimming in an aquarium with bubbles rising").
Step 3: I will combine it with the "video_frame_interpolation" reference to enable interpolation
↪  for smoother video.
Step 4: I will adapt the interpolation multiplier to 3x to achieve the expected frame rate
↪  increase.
Step 5: I will finish the task by confirming the generated output meets the required
↪  specifications.
</plan>

<action>
load(name="text_to_video")
```

```
</action>

- Step: 2

<thought>
The current workflow generates a 3-second video of a landscape scene, but this is not aligned
↪  with the task of generating a 2-second video of colorful fish swimming in an aquarium with
↪  bubbles rising. I need to adapt the text prompt accordingly to match the requirements.
↪  Additionally, the frame rate is currently 8 frames per second, but after generating the
↪  video, we will need to interpolate it to increase the frame rate by 3x to make the motion
↪  smoother.

The basic pipeline for generating videos from a text prompt is already functional, but the
↪  correct text needs to be provided, and the next step would be to apply video frame
↪  interpolation. I will combine this workflow with the "video_frame_interpolation" to achieve
↪  smoother motion.
</thought>

<plan>
Step 1: Adapt the text prompt to describe the required scene of "colorful fish swimming in an
↪  aquarium with bubbles rising".
Step 2: Combine the current workflow with the "video_frame_interpolation" reference workflow to
↪  add video interpolation functionality.
Step 3: Adapt the interpolation multiplier to 3x for smoother motion.
Step 4: Finish the task after confirming that the workflow meets the requirements.
</plan>

<action>
adapt(prompt="Change the text prompt to 'colorful fish swimming in an aquarium with bubbles
↪  rising to the surface' and reduce the video duration to 2 seconds.")
</action>

## Workspace

The code and annotation of the current workflow you are working on are presented as follows:

<code>
model_15, clip_vision_15, vae_15 =
↪  ImageOnlyCheckpointLoader(ckpt_name="""svd_xt_1_1.safetensors""")
model_16, clip_16, vae_16 = CheckpointLoaderSimple(ckpt_name="""sd_xl_base_1.0.safetensors""")
conditioning_18 = CLIPTextEncode(text="""colorful fish swimming in an aquarium with bubbles
↪  rising to the surface""", speak_and_recognation=True, clip=clip_16)
conditioning_19 = CLIPTextEncode(text="""text, watermark""", speak_and_recognation=True,
↪  clip=clip_16)
latent_22 = EmptyLatentImage(width=1024, height=576, batch_size=1)
model_14 = VideoLinearCFGGuidance(min_cfg=1, model=model_15)
latent_17 = KSampler(seed=307393744025667, steps=15, cfg=8, sampler_name="""uni_pc_bh2""",
↪  scheduler="""normal""", denoise=1, model=model_16, positive=conditioning_18,
↪  negative=conditioning_19, latent_image=latent_22)
image_20 = VAEDecode(samples=latent_17, vae=vae_16)
_ = PreviewImage(images=image_20)
positive_12, negative_12, latent_12 = SVD_img2vid_Conditioning(width=1024, height=576,
↪  video_frames=16, motion_bucket_id=127, fps=8, augmentation_level=0,
↪  clip_vision=clip_vision_15, init_image=image_20, vae=vae_15)
latent_3 = KSampler(seed=1103641334004632, steps=20, cfg=2.5, sampler_name="""euler""",
↪  scheduler="""karras""", denoise=1, model=model_14, positive=positive_12,
↪  negative=negative_12, latent_image=latent_12)
image_8 = VAEDecode(samples=latent_3, vae=vae_15)
filenames_23 = VHS_VideoCombine(frame_rate=8, loop_count=0, filename_prefix="""SVD_txt2vid""",
↪  format="""video/h264-mp4""", pix_fmt="""yuv420p""", crf=19, save_metadata=True,
↪  pingpong=False, save_output=True, images=image_8)
</code>

<function>
```

```
This workflow generates a 2-second video of colorful fish swimming in an aquarium with bubbles
↪  rising to the surface from a given text description. The video is first synthesized using a
↪  text-to-video diffusion process.
</function>

<principle>
The workflow first loads two models: a Stable Diffusion model ("sd_xl_base_1.0.safetensors") for
↪  generating the initial image from the text description, and then a Stable Video Diffusion
↪  model ("svd_xt_1_1.safetensors") to translate that image into a dynamic video over multiple
↪  frames. Following the text input describing colorful fish and bubbles, the image is
↪  generated and expanded across 16 frames, which corresponds to a duration of 2 seconds at 8
↪  FPS (frames per second). The final MP4 video is constructed and saved along with metadata.
</principle>

## Action

Based on the history and workspace, you should first think about what functions have been
↪  implemented and what modules remain to be added. Your thought should be enclosed with
↪  "<thought>" tag. For example: <thought>The basic pipeline has been implemented, but a module
↪  is needed to improve the quality.</thought>.

After that, you should update your step-by-step plan to further modify your workflow. There are
↪  3 steps remaining, so your plan should contain at most 3 steps. Make sure that each step is
↪  feasible to be converted into a single action. Your plan should be enclosed with "<plan>"
↪  tag. For example: <plan>Step 1: I will refer to "reference_name" to add a module. Step 2: I
↪  will finish the task since the expected effects are realized.</plan>.

Finally, you should choose one of the following actions and specify the arguments (if required),
↪  so that the updated workflow can realize the first step in your plan. You should provide
↪  your action with the format of function calls in Python. Your action should be enclosed with
↪  "<action>" tag. For example: <action>combine(name="reference_name")</action>,
↪  <action>adapt(prompt="Change the factor to 0.5 and rewrite the prompt.")</action>, and
↪  <action>finish()</action>.

- `load`: Load a reference workflow into the workspace to replace the current workflow, so that
↪  you can start over. Arguments:
  - `name`: The name of the reference workflow you want to load.
- `combine`: Combine the current workflow with a reference workflow, so that necessary modules
↪  can be added. Arguments:
  - `name`: The name of the reference workflow you want to combine.
- `adapt`: Adapt some parameters in the current workflow, so that the expected effects can be
↪  realized. Arguments:
  - `prompt`: The prompt to specify the adaptation you want to make.
- `retrieve`: Retrieve a new batch of reference workflows, so that more useful references can be
↪  found. Arguments:
  - `prompt`: The prompt to describe the reference you want to retrieve.
- `finish`: Finish the task since the current workflow can realize the expected effects.

Refer to the history before making a decision. Here are some general rules you should follow:

1. You should choose the `load` action if and only if the history is empty.
2. If you choose the `load` or `combine` action, make sure the name exists in the reference.
↪  Otherwise, try to update the reference with the `retrieve` action.
3. You should not choose the `adapt` action twice in a row, because they can be simplified into a
↪  single action.
4. If you choose the `adapt` or `retrieve` action, make sure the prompt is concise and contains
↪  all the necessary information.
5. You should choose the `finish` action before the remaining steps count down to 0.

Now, provide your thought, plan and action with the required format.
```

PlanAgent Answer

```
<thought>
The video generation part is complete, wherein a 2-second video of colorful fish swimming in an
↪  aquarium with bubbles rising is generated at 8 FPS. The next part of the task is to increase
↪  the frame rate by 3x using frame interpolation to achieve smoother motion. I need to combine
↪  the current workflow with the "video_frame_interpolation" reference to add the interpolation
↪  capability and then adjust the frame rate multiplier to 3x as per the requirements.
</thought>

<plan>
Step 1: Combine the current workflow with the "video_frame_interpolation" reference to add the
↪  interpolation functionality.
Step 2: Adapt the multiplier for frame interpolation to 3x to ensure the frame rate increase.
Step 3: Finish the task since this will meet the stated requirements.
</plan>

<action>combine(name="video_frame_interpolation")</action>
```

CombineAgent Prompt

```
## Task

ComfyUI uses workflows to create and execute Stable Diffusion pipelines so that users can design
↪  their own workflows to generate highly customized artworks. ComfyUI workflows can be
↪  formulated into the equivalent Python code, where each statement represents the execution of
↪  a single node. You are an expert in ComfyUI, helping users to design workflows according to
↪  their requirements.

Now you are required to create a ComfyUI workflow to finish the following task:

First generate a 2-second video of colorful fish swimming in an aquarium with bubbles rising to
↪  the surface. Then interpolate the video to increase the frame rate by 3x. The result should
↪  be a smoother video of the swimming fish.

The core requirements and the expected paradigm are analyzed as follows:

The task requires generating a short 2-second video of fish swimming in an aquarium with bubbles
↪  rising using a text-to-video approach, as the primary input is textual. The video needs to
↪  depict dynamic, colorful fish movements with the additional visual element of rising
↪  bubbles. After generating this video, the frame rate needs to be increased by 3x using a
↪  video interpolation process to achieve smoother motion. Thus, the workflow core involves two
↪  stages, first, synthesizing the video from a text prompt using a text-to-video node or
↪  series of nodes, followed by applying video interpolation to enhance frame smoothness. There
↪  are no specific quality-oriented steps like upscaling or refinement beyond the interpolation
↪  requirement.

## Reference

The code and annotation of the current workflow you are referring to are presented as follows:

<code>
image_7, frame_count_7, audio_7, video_info_7, latent_7 = VHS_LoadVideo(custom_height=512,
↪  custom_width=512, force_rate=0, force_size="""Disabled""", frame_load_cap=0,
↪  select_every_nth=1, skip_first_frames=0, video="""play_guitar.gif""")
image_10 = RIFE_VFI(ckpt_name="""rife47.pth""", clear_cache_after_n_frames=10, ensemble=True,
↪  fast_mode=True, frames=image_7, multiplier=3, scale_factor=1)
filenames_3 = VHS_VideoCombine(filename_prefix="""AnimateDiff""", format="""image/gif""",
↪  frame_rate=24, images=image_10, loop_count=0, pingpong=False, save_output=True)
</code>

<function>
```

```
This workflow performs video frame interpolation using the RIFE VFI model. It takes an input
↪  video such as "play_guitar.gif", increases the frame rate by generating intermediate frames
↪  (interpolating) with a multiplier (in this case, 3x), and produces a smoother video with a
↪  higher frame rate (from 8 to 24 frames per second). The final output is saved as a new video
↪  or animated GIF.
</function>

<principle>
The workflow first loads the input video using "VHS_LoadVideo", which extracts the individual
↪  frames. The "RIFE VFI" node is then used to interpolate the frames by generating additional
↪  frames between the existing ones. In this scenario, the multiplier is set to 3x, effectively
↪  tripling the frame count and enabling a smoother video playback at 24 frames per second.
↪  Finally, the interpolated frames are combined into a video or GIF format using
↪  "VHS_VideoCombine".
</principle>

## Workspace

The code and annotation of the current workflow you are working on are presented as follows:

<code>
model_15, clip_vision_15, vae_15 =
↪  ImageOnlyCheckpointLoader(ckpt_name="""svd_xt_1_1.safetensors""")
model_16, clip_16, vae_16 = CheckpointLoaderSimple(ckpt_name="""sd_xl_base_1.0.safetensors""")
conditioning_18 = CLIPTextEncode(text="""colorful fish swimming in an aquarium with bubbles
↪  rising to the surface""", speak_and_recognation=True, clip=clip_16)
conditioning_19 = CLIPTextEncode(text="""text, watermark""", speak_and_recognation=True,
↪  clip=clip_16)
latent_22 = EmptyLatentImage(width=1024, height=576, batch_size=1)
model_14 = VideoLinearCFGGuidance(min_cfg=1, model=model_15)
latent_17 = KSampler(seed=307393744025667, steps=15, cfg=8, sampler_name="""uni_pc_bh2""",
↪  scheduler="""normal""", denoise=1, model=model_16, positive=conditioning_18,
↪  negative=conditioning_19, latent_image=latent_22)
image_20 = VAEDecode(samples=latent_17, vae=vae_16)
_ = PreviewImage(images=image_20)
positive_12, negative_12, latent_12 = SVD_img2vid_Conditioning(width=1024, height=576,
↪  video_frames=16, motion_bucket_id=127, fps=8, augmentation_level=0,
↪  clip_vision=clip_vision_15, init_image=image_20, vae=vae_15)
latent_3 = KSampler(seed=1103641334004632, steps=20, cfg=2.5, sampler_name="""euler""",
↪  scheduler="""karras""", denoise=1, model=model_14, positive=positive_12,
↪  negative=negative_12, latent_image=latent_12)
image_8 = VAEDecode(samples=latent_3, vae=vae_15)
filenames_23 = VHS_VideoCombine(frame_rate=8, loop_count=0, filename_prefix="""SVD_txt2vid""",
↪  format="""video/h264-mp4""", pix_fmt="""yuv420p""", crf=19, save_metadata=True,
↪  pingpong=False, save_output=True, images=image_8)
</code>

<function>
This workflow generates a 2-second video of colorful fish swimming in an aquarium with bubbles
↪  rising to the surface from a given text description. The video is first synthesized using a
↪  text-to-video diffusion process.
</function>

<principle>
The workflow first loads two models: a Stable Diffusion model ("sd_xl_base_1.0.safetensors") for
↪  generating the initial image from the text description, and then a Stable Video Diffusion
↪  model ("svd_xt_1_1.safetensors") to translate that image into a dynamic video over multiple
↪  frames. Following the text input describing colorful fish and bubbles, the image is
↪  generated and expanded across 16 frames, which corresponds to a duration of 2 seconds at 8
↪  FPS (frames per second). The final MP4 video is constructed and saved along with metadata.
</principle>

## Combination

Based on the current working progress, your schedule is presented as follows:
```

```
Step 1: Combine the current workflow with the "video_frame_interpolation" reference to add the
↪ interpolation functionality.
Step 2: Adapt the multiplier for frame interpolation to 3x to ensure the frame rate increase.
Step 3: Finish the task since this will meet the stated requirements.

You are working on the first step of your schedule. In other words, you should combine the
↪ reference workflow with the current workflow according to your schedule.

First, you should provide your Python code to formulate the updated workflow. Each line of code
↪ should correspond to a single node, so you should avoid nested calls in a single statement.
↪ You should also avoid reusing the same variable name, even if the variable is temporary.
↪ Your code should be enclosed with "<code>" tag. For example: <code>output =
↪ node(input)</code>.

After that, you should provide an annotation as in the reference, including the function and
↪ principle of the updated workflow. The function should be enclosed with "<function>" tag.
↪ For example: <function>This workflow generates a high-resolution image of a running
↪ horse.</function>. The principle should be enclosed with "<principle>" tag. For example:
↪ <principle>The workflow first generates a low-resolution image using the text-to-image
↪ pipeline and then applies an upscaling module to improve the resolution.</principle>.

Now, provide your code and annotation with the required format.
```

CombineAgent Answer

```
<code>
model_15, clip_vision_15, vae_15 =
↪ ImageOnlyCheckpointLoader(ckpt_name="""svd_xt_1_1.safetensors""")
model_16, clip_16, vae_16 = CheckpointLoaderSimple(ckpt_name="""sd_xl_base_1.0.safetensors""")
conditioning_18 = CLIPTextEncode(text="""colorful fish swimming in an aquarium with bubbles
↪ rising to the surface""", speak_and_recognation=True, clip=clip_16)
conditioning_19 = CLIPTextEncode(text="""text, watermark""", speak_and_recognation=True,
↪ clip=clip_16)
latent_22 = EmptyLatentImage(width=1024, height=576, batch_size=1)
model_14 = VideoLinearCFGGuidance(min_cfg=1, model=model_15)
latent_17 = KSampler(seed=307393744025667, steps=15, cfg=8, sampler_name="""uni_pc_bh2""",
↪ scheduler="""normal""", denoise=1, model=model_16, positive=conditioning_18,
↪ negative=conditioning_19, latent_image=latent_22)
image_20 = VAEDecode(samples=latent_17, vae=vae_16)
_ = PreviewImage(images=image_20)
positive_12, negative_12, latent_12 = SVD_img2vid_Conditioning(width=1024, height=576,
↪ video_frames=16, motion_bucket_id=127, fps=8, augmentation_level=0,
↪ clip_vision=clip_vision_15, init_image=image_20, vae=vae_15)
latent_3 = KSampler(seed=1103641334004632, steps=20, cfg=2.5, sampler_name="""euler""",
↪ scheduler="""karras""", denoise=1, model=model_14, positive=positive_12,
↪ negative=negative_12, latent_image=latent_12)
image_8 = VAEDecode(samples=latent_3, vae=vae_15)
filenames_23 = VHS_VideoCombine(frame_rate=8, loop_count=0, filename_prefix="""SVD_txt2vid""",
↪ format="""video/h264-mp4""", pix_fmt="""yuv420p""", crf=19, save_metadata=True,
↪ pingpong=False, save_output=True, images=image_8)

image_10 = RIFE_VFI(ckpt_name="""rife47.pth""", clear_cache_after_n_frames=10, ensemble=True,
↪ fast_mode=True, frames=image_8, multiplier=3, scale_factor=1)
filenames_24 = VHS_VideoCombine(frame_rate=24, loop_count=0,
↪ filename_prefix="""SVD_txt2vid_Interpolated""", format="""video/h264-mp4""",
↪ pix_fmt="""yuv420p""", crf=19, save_metadata=True, pingpong=False, save_output=True,
↪ images=image_10)
</code>

<function>
```

```
This workflow first generates a 2-second video of colorful fish swimming in an aquarium with bubbles rising to the surface from a text description using a text-to-video approach. It then performs video interpolation to increase the frame rate by 3x, resulting in a smoother video.
</function>

<principle>
The workflow begins by loading the necessary models, including the Stable Diffusion model and the Stable Video Diffusion model. These are used to encode the input text to create a video showcasing colorful fish and bubbles. Initially, the video is generated at 8 frames per second (FPS) over 16 frames (~2 seconds). After the video is synthesized, it is passed through a frame interpolation process using the RIFE VFI model, which increases the frame rate by generating additional frames, multiplying the frame rate by 3x (from 8 FPS to 24 FPS). Finally, the interpolated frames are combined into a smoother MP4 video output.
</principle>
```

PlanAgent Prompt

```
## Task

ComfyUI uses workflows to create and execute Stable Diffusion pipelines so that users can design their own workflows to generate highly customized artworks. ComfyUI workflows can be formulated into the equivalent Python code, where each statement represents the execution of a single node. You are an expert in ComfyUI, helping users to design workflows according to their requirements.

Now you are required to create a ComfyUI workflow to finish the following task:

First generate a 2-second video of colorful fish swimming in an aquarium with bubbles rising to the surface. Then interpolate the video to increase the frame rate by 3x. The result should be a smoother video of the swimming fish.

The core requirements and the expected paradigm are analyzed as follows:

The task requires generating a short 2-second video of fish swimming in an aquarium with bubbles rising using a text-to-video approach, as the primary input is textual. The video needs to depict dynamic, colorful fish movements with the additional visual element of rising bubbles. After generating this video, the frame rate needs to be increased by 3x using a video interpolation process to achieve smoother motion. Thus, the workflow core involves two stages, first, synthesizing the video from a text prompt using a text-to-video node or series of nodes, followed by applying video interpolation to enhance frame smoothness. There are no specific quality-oriented steps like upscaling or refinement beyond the interpolation requirement.

Improving the quality of the generation result with additional steps, such as upscaling and interpolation, is not recommended, unless specified in the requirements.

## Reference

According to the requirements, we have retrieved some relevant workflows which may be helpful:

- Example: text_to_video

<function>
This workflow follows a text-to-video paradigm, where it generates a video from a given text prompt. It first creates an initial image based on the text description and then uses that image as the starting frame to generate a video with motion and transformation using a Stable Video Diffusion model. The output is a 3-second video at 8 frames per second, depicting beautiful scenery with mountains, rivers, and clouds.
</function>

<principle>
```

```
The workflow first loads two models: a Stable Diffusion model for generating the initial image
↪  from the text ("sd_xl_base_1.0.safetensors") and a Stable Video Diffusion model
↪  ("svd_xt_1_1.safetensors") for video generation. It uses the text description to create
↪  conditioning, generating an initial 1024x576 image. The image is decoded from latent space
↪  via a VAE. The video-specific node ("SVD_img2vid_Conditioning") then applies continuity and
↪  motion to this image, producing conditioned latent representations for video generation. The
↪  final video is created by sampling the latent space over multiple frames and combining them
↪  into an MP4 video using the specified frame rate and format.
</principle>

- Example: video_frame_interpolation

<function>
This workflow performs video frame interpolation using the RIFE VFI model. It takes an input
↪  video such as "play_guitar.gif", increases the frame rate by generating intermediate frames
↪  (interpolating) with a multiplier (in this case, 3x), and produces a smoother video with a
↪  higher frame rate (from 8 to 24 frames per second). The final output is saved as a new video
↪  or animated GIF.
</function>

<principle>
The workflow first loads the input video using "VHS_LoadVideo", which extracts the individual
↪  frames. The "RIFE VFI" node is then used to interpolate the frames by generating additional
↪  frames between the existing ones. In this scenario, the multiplier is set to 3x, effectively
↪  tripling the frame count and enabling a smoother video playback at 24 frames per second.
↪  Finally, the interpolated frames are combined into a video or GIF format using
↪  "VHS_VideoCombine".
</principle>

- Example: image_to_video

<function>
This workflow follows an image-to-video paradigm. It requires an input image (in this case,
↪  "play_guitar.jpg") and generates a 4-second video at 6 frames per second (24 video frames in
↪  total) based on that image. The workflow outputs the generated video.
</function>

<principle>
The workflow uses the "svd_xt_1_1.safetensors" Stable Video Diffusion model to generate a video
↪  from the input image "play_guitar.jpg". The "SVD_img2vid_Conditioning" node creates the
↪  necessary conditioning for video generation, including the number of frames, resolution, and
↪  motion characteristics. A KSamplerAdvanced node adds noise and performs generative sampling
↪  over multiple steps to create diverse video frames. These frames are then decoded back into
↪  images via a VAE, and finally, the "VHS_VideoCombine" node compiles these images into a
↪  4-second video at 8 frames per second.
</principle>

- Example: text_to_image

<function>
This workflow implements a basic text-to-image generation pipeline using Stable Diffusion. It
↪  requires both positive (desired traits) and negative (undesired traits) text prompts to
↪  generate an image. In this specific case, the workflow will output a high-resolution photo
↪  of a cat wearing a spacesuit inside a spaceship, avoiding blurry or illustration-like
↪  effects.
</function>

<principle>
The workflow begins by loading the "dreamshaper_8.safetensors" Stable Diffusion model. It
↪  generates a blank latent space as the starting point for the image generation. The positive
↪  prompt ("a photo of a cat wearing a spacesuit inside a spaceship") and negative prompt
↪  ("blurry, illustration") are encoded into conditioning by the CLIPTextEncode node. The
↪  KSampler node then uses these conditionings to guide the generative process over 20 sampling
↪  steps, applying denoising to the latent space. The resulting latent code is subsequently
↪  decoded into an image using the VAE and saved to disk.
```

```
</principle>

- Example: scribble_image_repaint

<function>
This workflow follows a sketch-to-image paradigm, where it takes a scribble image
↪  ("simple_graffiti.png") along with a text prompt ("a bird, open wings") and generates a
↪  detailed, high-quality image based on both the scribble and the prompt.
</function>

<principle>
The workflow first loads and inverts the input scribble image. The inverted image is then used
↪  by a "control_v11p_sd15_scribble_fp16" ControlNet model to extract and applied to the
↪  scribble, controlling and guiding the image generation process. A pre-trained model
↪  ("dreamshaper_8.safetensors") processes the positive conditioning generated by the
↪  combination of the text prompt and the ControlNet's output, along with negative conditioning
↪  from undesired traits such as "horror" and "lowres." The latent space image is generated
↪  using a KSampler and further decoded by the VAE to form the final image. The model
↪  synthesizes a highly detailed image while adhering closely to both the shape of the scribble
↪  and the desired features implied by the text prompt.
</principle>

## History

Here is a recent history of your thought, plan and action in the previous steps. The most recent
↪  record is at the bottom.

- Step: 1

<thought>
Since the history is empty, I need to start by loading a relevant reference workflow that can be
↪  used as a base for generating the 2-second video of colorful fish in an aquarium. The
↪  "text_to_video" reference seems appropriate to use as it involves generating video from text
↪  prompts, which aligns with the first part of the task.
</thought>

<plan>
Step 1: I will load the "text_to_video" reference workflow to use it as the base.
Step 2: I will adapt the workflow to modify the prompt to match the requirements ("colorful fish
↪  swimming in an aquarium with bubbles rising").
Step 3: I will combine it with the "video_frame_interpolation" reference to enable interpolation
↪  for smoother video.
Step 4: I will adapt the interpolation multiplier to 3x to achieve the expected frame rate
↪  increase.
Step 5: I will finish the task by confirming the generated output meets the required
↪  specifications.
</plan>

<action>
load(name="text_to_video")
</action>

- Step: 2

<thought>
The current workflow generates a 3-second video of a landscape scene, but this is not aligned
↪  with the task of generating a 2-second video of colorful fish swimming in an aquarium with
↪  bubbles rising. I need to adapt the text prompt accordingly to match the requirements.
↪  Additionally, the frame rate is currently 8 frames per second, but after generating the
↪  video, we will need to interpolate it to increase the frame rate by 3x to make the motion
↪  smoother.
```

```
The basic pipeline for generating videos from a text prompt is already functional, but the correct text needs to be provided, and the next step would be to apply video frame interpolation. I will combine this workflow with the "video_frame_interpolation" to achieve smoother motion.
</thought>

<plan>
Step 1: Adapt the text prompt to describe the required scene of "colorful fish swimming in an aquarium with bubbles rising".
Step 2: Combine the current workflow with the "video_frame_interpolation" reference workflow to add video interpolation functionality.
Step 3: Adapt the interpolation multiplier to 3x for smoother motion.
Step 4: Finish the task after confirming that the workflow meets the requirements.
</plan>

<action>
adapt(prompt="Change the text prompt to 'colorful fish swimming in an aquarium with bubbles rising to the surface' and reduce the video duration to 2 seconds.")
</action>

- Step: 3

<thought>
The video generation part is complete, wherein a 2-second video of colorful fish swimming in an aquarium with bubbles rising is generated at 8 FPS. The next part of the task is to increase the frame rate by 3x using frame interpolation to achieve smoother motion. I need to combine the current workflow with the "video_frame_interpolation" reference to add the interpolation capability and then adjust the frame rate multiplier to 3x as per the requirements.
</thought>

<plan>
Step 1: Combine the current workflow with the "video_frame_interpolation" reference to add the interpolation functionality.
Step 2: Adapt the multiplier for frame interpolation to 3x to ensure the frame rate increase.
Step 3: Finish the task since this will meet the stated requirements.
</plan>

<action>
combine(name="video_frame_interpolation")
</action>

## Workspace

The code and annotation of the current workflow you are working on are presented as follows:

<code>
model_15, clip_vision_15, vae_15 = ImageOnlyCheckpointLoader(ckpt_name="""svd_xt_1_1.safetensors""")
model_16, clip_16, vae_16 = CheckpointLoaderSimple(ckpt_name="""sd_xl_base_1.0.safetensors""")
conditioning_18 = CLIPTextEncode(text="""colorful fish swimming in an aquarium with bubbles rising to the surface""", speak_and_recognation=True, clip=clip_16)
conditioning_19 = CLIPTextEncode(text="""text, watermark""", speak_and_recognation=True, clip=clip_16)
latent_22 = EmptyLatentImage(width=1024, height=576, batch_size=1)
model_14 = VideoLinearCFGGuidance(min_cfg=1, model=model_15)
latent_17 = KSampler(seed=307393744025667, steps=15, cfg=8, sampler_name="""uni_pc_bh2""", scheduler="""normal""", denoise=1, model=model_16, positive=conditioning_18, negative=conditioning_19, latent_image=latent_22)
image_20 = VAEDecode(samples=latent_17, vae=vae_16)
_ = PreviewImage(images=image_20)
positive_12, negative_12, latent_12 = SVD_img2vid_Conditioning(width=1024, height=576, video_frames=16, motion_bucket_id=127, fps=8, augmentation_level=0, clip_vision=clip_vision_15, init_image=image_20, vae=vae_15)
```

```
latent_3 = KSampler(seed=1103641334004632, steps=20, cfg=2.5, sampler_name="""euler""",
↪  scheduler="""karras""", denoise=1, model=model_14, positive=positive_12,
↪  negative=negative_12, latent_image=latent_12)
image_8 = VAEDecode(samples=latent_3, vae=vae_15)
filenames_23 = VHS_VideoCombine(frame_rate=8, loop_count=0, filename_prefix="""SVD_txt2vid""",
↪  format="""video/h264-mp4""", pix_fmt="""yuv420p""", crf=19, save_metadata=True,
↪  pingpong=False, save_output=True, images=image_8)

image_10 = RIFE_VFI(ckpt_name="""rife47.pth""", clear_cache_after_n_frames=10, ensemble=True,
↪  fast_mode=True, frames=image_8, multiplier=3, scale_factor=1)
filenames_24 = VHS_VideoCombine(frame_rate=24, loop_count=0,
↪  filename_prefix="""SVD_txt2vid_Interpolated""", format="""video/h264-mp4""",
↪  pix_fmt="""yuv420p""", crf=19, save_metadata=True, pingpong=False, save_output=True,
↪  images=image_10)
</code>

<function>
This workflow first generates a 2-second video of colorful fish swimming in an aquarium with
↪  bubbles rising to the surface from a text description using a text-to-video approach. It
↪  then performs video interpolation to increase the frame rate by 3x, resulting in a smoother
↪  video.
</function>

<principle>
The workflow begins by loading the necessary models, including the Stable Diffusion model and
↪  the Stable Video Diffusion model. These are used to encode the input text to create a video
↪  showcasing colorful fish and bubbles. Initially, the video is generated at 8 frames per
↪  second (FPS) over 16 frames (~2 seconds). After the video is synthesized, it is passed
↪  through a frame interpolation process using the RIFE VFI model, which increases the frame
↪  rate by generating additional frames, multiplying the frame rate by 3x (from 8 FPS to 24
↪  FPS). Finally, the interpolated frames are combined into a smoother MP4 video output.
</principle>

## Action

Based on the history and workspace, you should first think about what functions have been
↪  implemented and what modules remain to be added. Your thought should be enclosed with
↪  "<thought>" tag. For example: <thought>The basic pipeline has been implemented, but a module
↪  is needed to improve the quality.</thought>.

After that, you should update your step-by-step plan to further modify your workflow. There are
↪  2 steps remaining, so your plan should contain at most 2 steps. Make sure that each step is
↪  feasible to be converted into a single action. Your plan should be enclosed with "<plan>"
↪  tag. For example: <plan>Step 1: I will refer to "reference_name" to add a module. Step 2: I
↪  will finish the task since the expected effects are realized.</plan>.

Finally, you should choose one of the following actions and specify the arguments (if required),
↪  so that the updated workflow can realize the first step in your plan. You should provide
↪  your action with the format of function calls in Python. Your action should be enclosed with
↪  "<action>" tag. For example: <action>combine(name="reference_name")</action>,
↪  <action>adapt(prompt="Change the factor to 0.5 and rewrite the prompt.")</action>, and
↪  <action>finish()</action>.

- `load`: Load a reference workflow into the workspace to replace the current workflow, so that
↪  you can start over. Arguments:
  - `name`: The name of the reference workflow you want to load.
- `combine`: Combine the current workflow with a reference workflow, so that necessary modules
↪  can be added. Arguments:
  - `name`: The name of the reference workflow you want to combine.
- `adapt`: Adapt some parameters in the current workflow, so that the expected effects can be
↪  realized. Arguments:
  - `prompt`: The prompt to specify the adaptation you want to make.
- `retrieve`: Retrieve a new batch of reference workflows, so that more useful references can be
↪  found. Arguments:
  - `prompt`: The prompt to describe the reference you want to retrieve.
```

```
- `finish`: Finish the task since the current workflow can realize the expected effects.

Refer to the history before making a decision. Here are some general rules you should follow:

1. You should choose the `load` action if and only if the history is empty.
2. If you choose the `load` or `combine` action, make sure the name exists in the reference.
↪  Otherwise, try to update the reference with the `retrieve` action.
3. You should not choose the `adapt` action twice in a row, because they can be simplified into a
↪  single action.
4. If you choose the `adapt` or `retrieve` action, make sure the prompt is concise and contains
↪  all the necessary information.
5. You should choose the `finish` action before the remaining steps count down to 0.

Now, provide your thought, plan and action with the required format.
```

PlanAgent Answer

```
<thought>The current workflow successfully generates a 2-second video of colorful fish swimming
↪  in an aquarium with bubbles rising to the surface and then applies video interpolation to
↪  enhance the frame rate by 3x. The workflow seems to fulfill both the generation and
↪  interpolation requirements. No additional modifications appear to be necessary, as the video
↪  generation part is functional and the interpolation process has been properly set to
↪  increase the frame rate from 8 FPS to 24 FPS.</thought>

<plan>
Step 1: I will finish the task since the expected effects have been realized.
</plan>

<action>finish()</action>
```

Figure 8. A typical example trajectory of ComfyAgent.