Student Name: Steven Tran

Student ID number: 01698762

Lab # 4

Date of completion: April 30, 2025

# Lab Focus

1. Main Topics Covered in this Lab  
   This lab focuses on controlled generation with image models. The first part emphasizes the use of ControlNet with different structural inputs (such as Canny edge maps, depth maps, and pose detection) to guide the generative output of image models. For part 2, the lab utilizes **Streamlit** for the user interface and connects with large language models (LLMs) running locally. Using the provided app.py, By using **Streamlit's st.session\_state**, the chatbot is able to preserve conversational context across user interactions.
2. Software Design
   1. The software implementation used the OpenPose-based conditioning method to control image generation. A fashion model image was first loaded and processed using the OpenposeDetector. This produced a pose map highlighting body keypoints. This pose map, along with a text prompt was passed to the StableDiffusionControlNetPipeline. The pipeline output preserved the subject’s pose while generating a new image aligned with the prompt.
   2. The Canny-based implementation explored structural control using edge detection to guide the diffusion process. An input image of a bird was first loaded and converted into a NumPy array. Using the Canny edge detector, an edge map was generated to highlight the outlines of the subject. This map was passed to a StableDiffusionControlNetPipeline loaded with the sd-controlnet-canny model. Image generation was performed using prompts resulting in outputs that conformed to the detected edge structure.
   3. For the depth-based conditioning setup, a depth estimation pipeline was used to extract spatial depth information from an image. The output depth map was processed and then used with the sd-controlnet-depth model integrated into a StableDiffusionControlNetPipeline. By pairing the depth map with prompts, the pipeline generated images that preserved depth-based scenes.
   4. In part 2 of the lab, the the python program app.py creates a local chatbot using **Ollama's LLM** and **LangChain** for interacting with uploaded PDF documents and external data sources. **Streamlit** is used for the user interface. The key features include PDF upload and processing, tool integration, memory management, and response generation. The app allows users to query documents and external data through a local chatbot that maintains conversation context and integrates multiple tools for enhanced responses.
3. Results
   1. Part 1

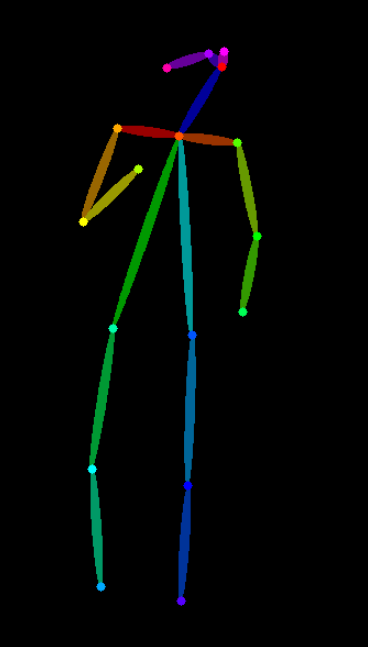
  

Image 1: Progression from model Image to pose map to new image with preserved pose

Image 2: Progression from model image to edge map to new image with preserved edges

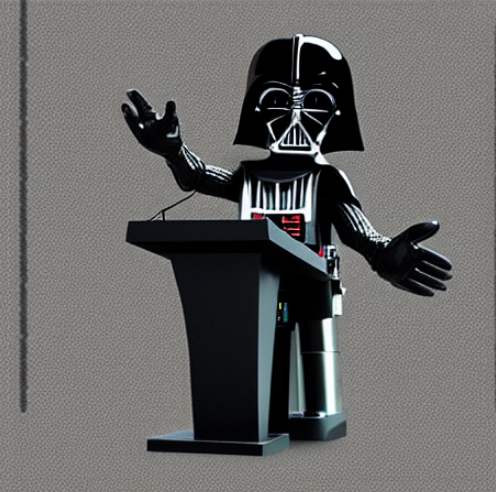
  

Image 3: Progression from model image to depth map to new images with preserved depths

* 1. Part 2

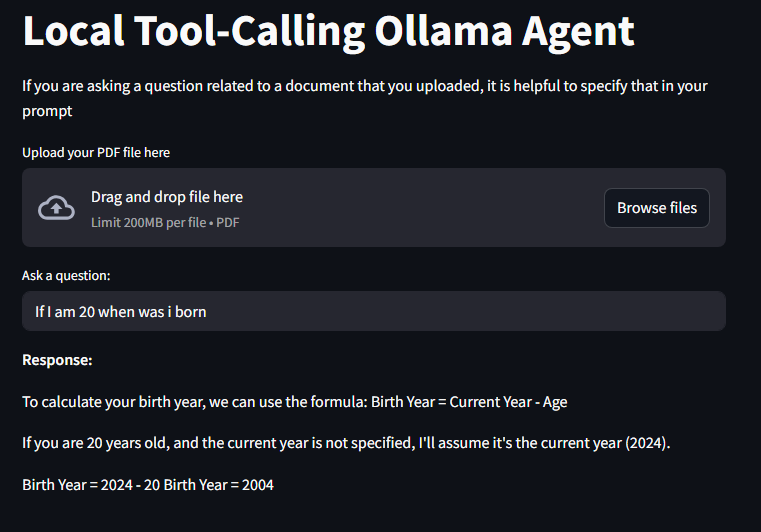
 

Image 4: Chatbot generation and memory preservation across user prompts

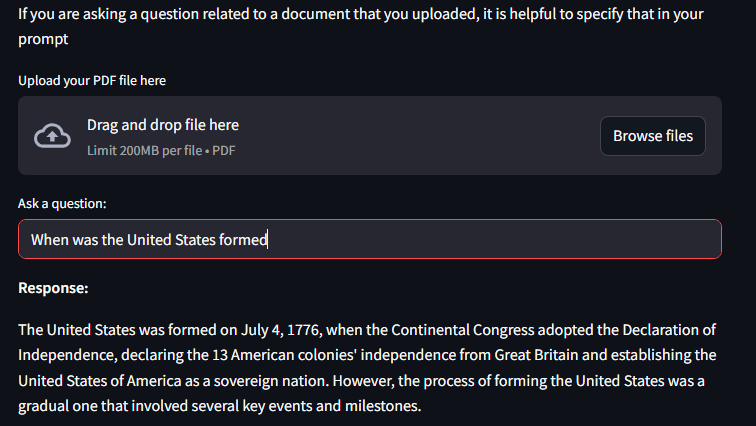


Image 5: Chatbot using tools to acquire information

# Problems Encountered and Solved

I encountered a **PermissionError: [Errno 13] Permission denied** while trying to open PDFs. This error occurred due to insufficient permissions when attempting to write the uploaded file to a temporary directory. The issue was not fixed, likely because the temporary file path did not have proper access permissions. I was not able to open the PDF successfully after multiple tries.

# Questions

* 1. Why is it necessary to use a pretrained pose detector like `OpenposeDetector` to preprocess the conditioning input, and what format does this model output to make it compatible with the ControlNet pipeline?  
     Using a pretrained pose detector like OpenposeDetector is necessary because it extracts the skeletal structure of the human body from the input image which serves as an input to ControlNet. This models outputs a pose map.
  2. How does integrating a specific ControlNet model into the `StableDiffusionControlNetPipeline` alter the generative process compared to using vanilla Stable Diffusion, and what are the technical implications of specifying `torch\_dtype=torch.float16` and targeting the `xpu` device?  
     Integrating a specific ControlNet model into the `StableDiffusionControlNetPipeline`alters the generative process by introducing additional inputs that guides image generation. Specifying torch\_dtype=torch.float16 reduces memory usage. Targeting the xpu device ensures that the model runs on Intel's specialized hardware.
  3. What is the role of the `image` argument in conjunction with the `prompt` for `pipe()`, and how does ControlNet internally combine these two conditioning signals during inference to produce a coherent output?  
     The image argument serves as a structure in the generative process. The prompt provides instructions for the content and style of the generated image. ControlNet combines these two conditioning signals by making sure the generated image respects the map structure and integrates it with the information from the prompt.
  4. Streamlit reruns the script on every user interaction. How does storing objects like the `MemorySaver` checkpoint in `st.session\_state` preserve chatbot memory across user prompts, and what potential pitfalls might arise if this step is skipped or incorrectly implemented?  
     Storing objects like the MemorySaver checkpoint in st.session\_state ensures that the memory persists across interactions, even though the script reruns. Potential pitfalls include improperly initializing or managing the session state, which could cause inconsistent behavior, such as losing memory or failing to restore it correctly.
  5. How does LangChain abstract the use of external APIs into tools, and how might you extend this setup to incorporate a new tool (e.g., querying arXiv or GitHub repositories)?  
     LangChain abstracts external APIs into tools by creating wrapper functions that can be invoked within the chatbot’s workflow. These integrated tools allow the chatbot to call external APIs or perform complex tasks. To extend this setup with a new tool, define a new tool function, integrate it into the toolset, and add it to the LangChain agent. Refer to image 6 and 7.

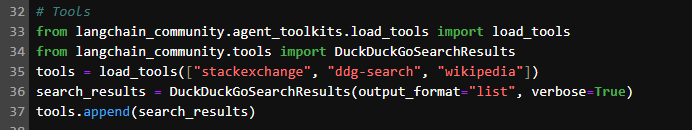


Image 6: Tools defining

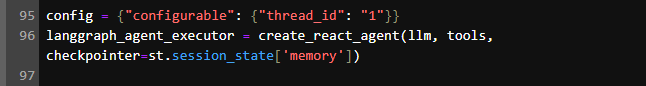


Image 7: Adding tools to LangChain agent

* 1. Ollama allows running large language models locally through a REST API. What are the advantages and trade-offs of running an LLM locally with Ollama compared to using cloud-hosted models, particularly in terms of latency, privacy, resource usage, and scalability?  
     Running the LLM locally has a few advantages. Notable advantages are the control over model configurations and privacy. However, the tradeoffs include lower computational resources and limited model size.

# Lessons Learnt

In Part 1: Controlled Generation, I learned how ControlNet uses various conditioning structures (like Canny, Depth, and Pose) to guide image generation. In Part 2: Local LLM Chatbot, I gained insight into integrating Ollama with LangChain and Streamlit for building a local chatbot. These experiences enhanced my understanding of working with both text and image generation models, and highlighted the importance of tool integration, and memory management in real-world AI applications.