Application for Question Answering on Job Postings Dataset

1. Overview

This project aims to develop a Question Answering (QA) platform by implementing a Decoder-Based Transformer using the T5 model. The objective is to create an interactive application using Streamlit, leveraging advanced AI techniques such as fine-tuning and deployment within a Colab environment using ngrok.

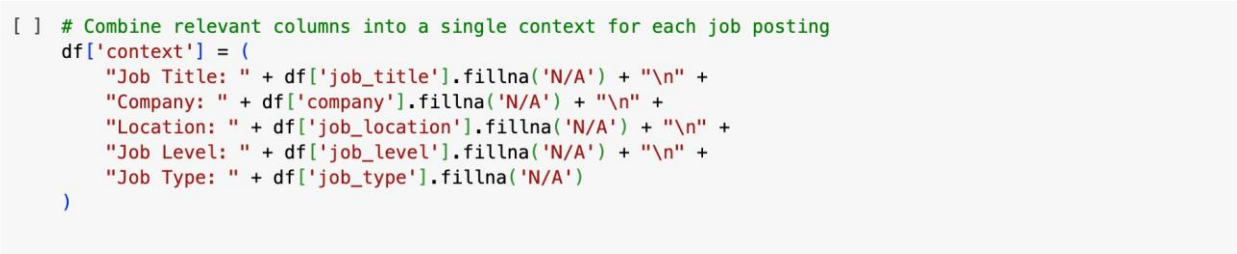
2. Project Goals and Scope

- **Use Case**: The application is designed as a QA system that provides contextual answers to user queries about job postings.
- **Main Objectives**:
  - Develop a decoder-based transformer model using T5.
  - Fine-tune the model on a custom dataset (job_postings.csv).
  - Deploy the model as an interactive web application using Streamlit and ngrok for public access.

3. Process of Implementation

Step 1: Preparing the Dataset

- **Dataset Used**: job_postings.csv, containing job titles, companies, locations, job levels, and job types.
- **Data Preprocessing**:
  - Combined relevant columns into a single context for each job posting:

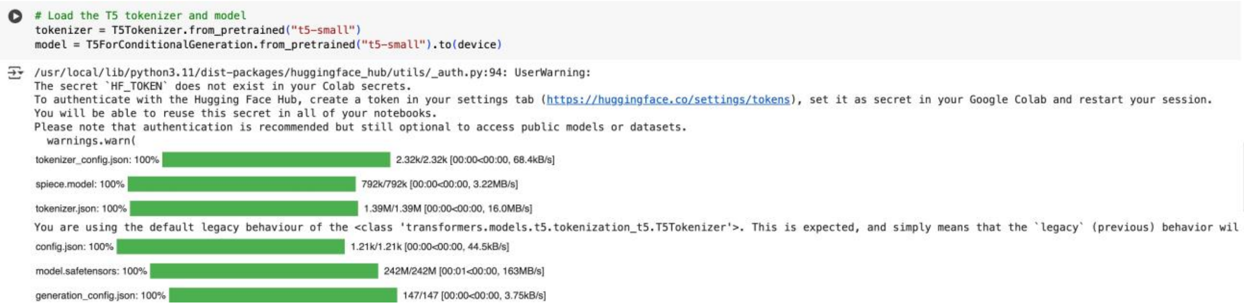


```
[ ]  # Combine relevant columns into a single context for each job posting
     df['context'] = (
         "Job Title: " + df['job_title'].fillna('N/A') + "\n" +
         "Company: " + df['company'].fillna('N/A') + "\n" +
         "Location: " + df['job_location'].fillna('N/A') + "\n" +
         "Job Level: " + df['job_level'].fillna('N/A') + "\n" +
         "Job Type: " + df['job_type'].fillna('N/A')
     )
```

  - Generated Question-Answer pairs for training:
    - What is the job title?
    - Which company is hiring?
    - Where is the job located?
    - What is the job level?
    - Is it a remote or onsite job?

Step 2: Model Training and Selection

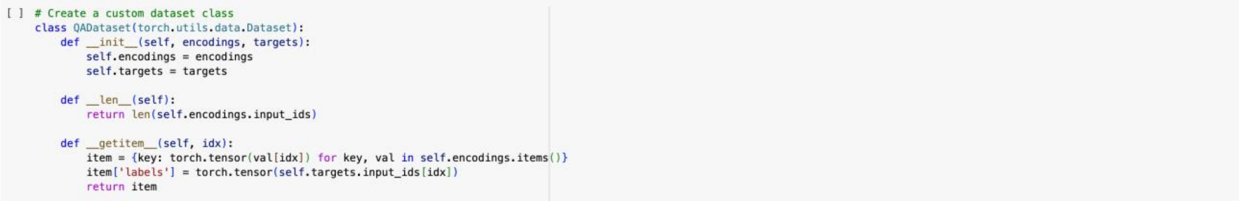
- **Base Model**: T5-small (Text-to-Text Transfer Transformer).
- **Model Setup**:
  - Loaded pre-trained T5 tokenizer and model:



```
# Load the T5 tokenizer and model
tokenizer = T5Tokenizer.from_pretrained("t5-small")
model = T5ForConditionalGeneration.from_pretrained("t5-small").to(device)
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 100% 2.32k/2.32k [00:00<00:00, 68.4kB/s]
spiece.model: 100% 792k/792k [00:00<00:00, 3.22MB/s]
tokenizer.json: 100% 1.39M/1.39M [00:00<00:00, 16.0MB/s]
You are using the default legacy behaviour of the <class 'transformers.models.t5.tokenization_t5.T5Tokenizer'>. This is expected, and simply means that the `legacy` (previous) behavior wil
config.json: 100% 1.21k/1.21k [00:00<00:00, 44.5kB/s]
model.safetensors: 100% 242M/242M [00:01<00:00, 163MB/s]
generation_config.json: 100% 147/147 [00:00<00:00, 3.75kB/s]
```

- Prepared input and output sequences using the context and questions.
- Custom Dataset Class was created for tokenized inputs:

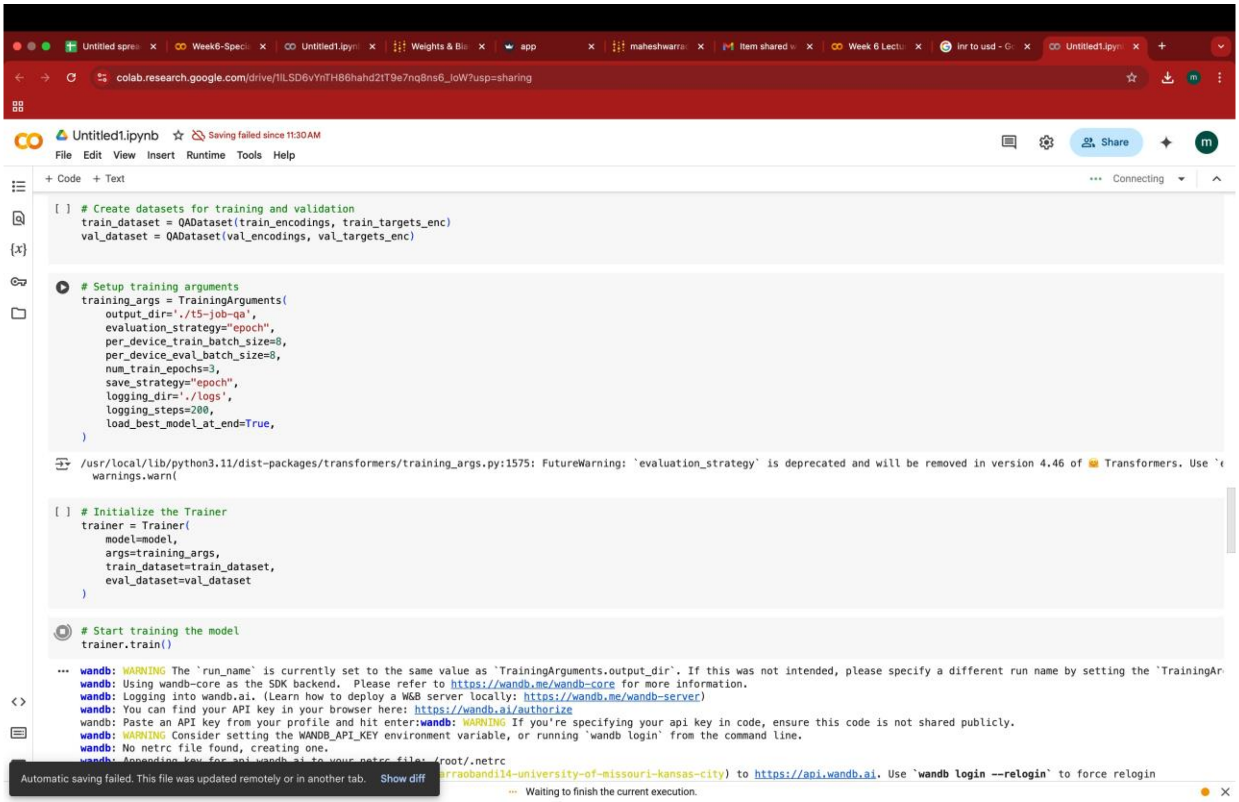


```
[ ] # Create a custom dataset class
    class QADataset(torch.utils.data.Dataset):
        def __init__(self, encodings, targets):
            self.encodings = encodings
            self.targets = targets

        def __len__(self):
            return len(self.encodings.input_ids)

        def __getitem__(self, idx):
            item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
            item['labels'] = torch.tensor(self.targets.input_ids[idx])
            return item
```

- Fine-tuned the model using Hugging Face Trainer API:



```
[ ] # Create datasets for training and validation
    train_dataset = QADataset(train_encodings, train_targets_enc)
    val_dataset = QADataset(val_encodings, val_targets_enc)
```

```
# Setup training arguments
training_args = TrainingArguments(
    output_dir='./t5-job-qa',
    evaluation_strategy="epoch",
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=3,
    save_strategy="epoch",
    logging_dir='./logs',
    logging_steps=200,
    load_best_model_at_end=True,
)
```

```
/usr/local/lib/python3.11/dist-packages/transformers/training_args.py:1575: FutureWarning: `evaluation_strategy` is deprecated and will be removed in version 4.46 of 🤗 Transformers. Use `e
  warnings.warn(
```

```
[ ] # Initialize the Trainer
    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=train_dataset,
        eval_dataset=val_dataset
    )
```

```
# Start training the model
trainer.train()
```

```
wandb: WARNING The `run_name` is currently set to the same value as `TrainingArguments.output_dir`. If this was not intended, please specify a different run name by setting the `TrainingAr
wandb: Using wandb-core as the SDK backend.  Please refer to https://wandb.me/wandb-core for more information.
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter:wandb: WARNING If you're specifying your api key in code, ensure this code is not shared publicly.
wandb: WARNING Consider setting the WANDB_API_KEY environment variable, or running `wandb login` from the command line.
wandb: No netrc file found, creating one.
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
arraobandi14-university-of-missouri-kansas-city) to https://api.wandb.ai. Use `wandb login --relogin` to force relogin
```

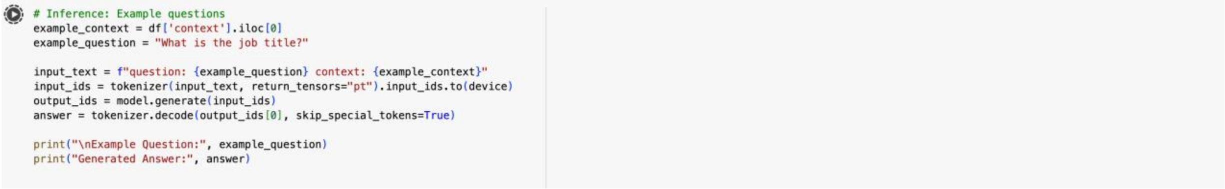
Automatic saving failed. This file was updated remotely or in another tab. Show diff

Waiting to finish the current execution.

Step 3: Development of the Application

- **Web Framework Used**: Streamlit
- **Key Features**:
  - Interactive UI to select a job posting and ask questions.

- Generated answers displayed in real-time using T5.
- Streamlit components used:
  - selectbox: Select a job posting.
  - text_input: Input for user questions.
  - success: Display generated answers.



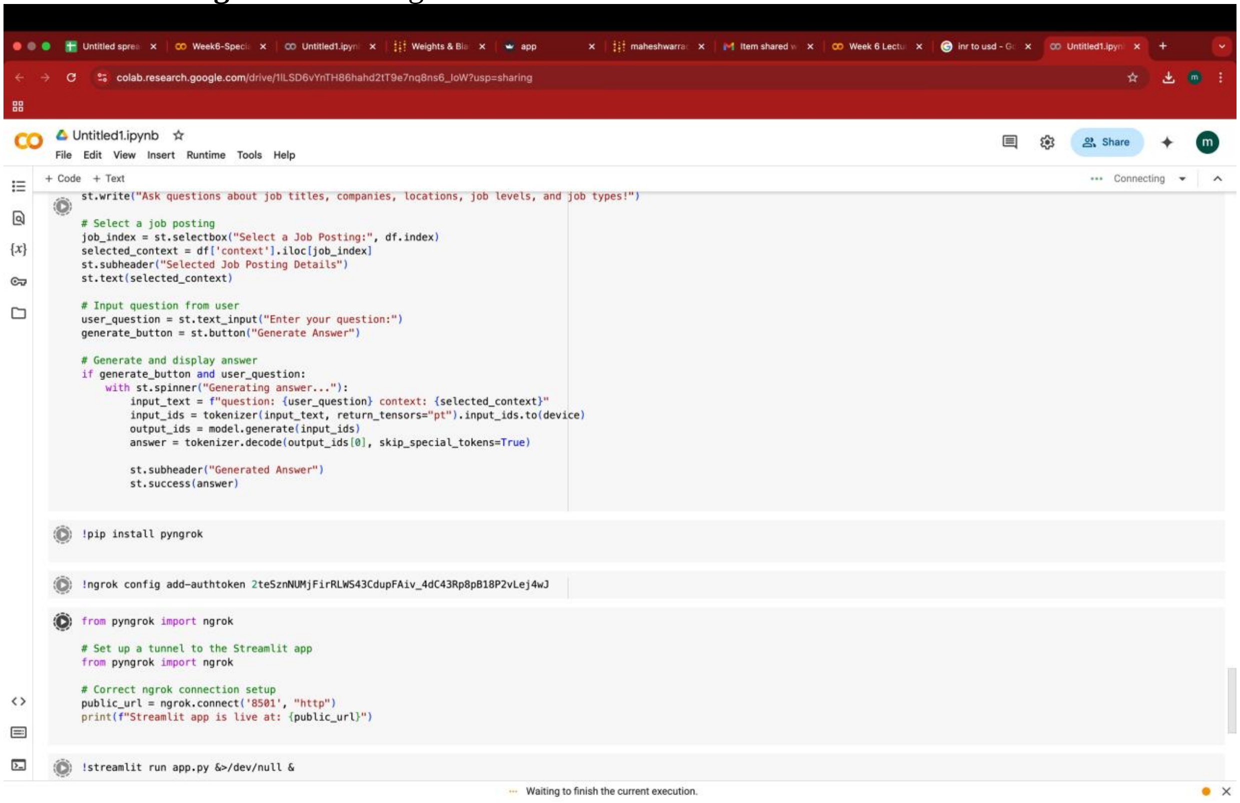
```
# Inference: Example questions
example_context = df['context'].iloc[0]
example_question = "What is the job title?"

input_text = f"question: {example_question} context: {example_context}"
input_ids = tokenizer(input_text, return_tensors="pt").input_ids.to(device)
output_ids = model.generate(input_ids)
answer = tokenizer.decode(output_ids[0], skip_special_tokens=True)

print("\nExample Question:", example_question)
print("Generated Answer:", answer)
```

Step 4: Testing and Deployment

- **Colab Deployment**:
  - Streamlit was used within Google Colab.
  - **ngrok** was configured for external access:



```
st.write("Ask questions about job titles, companies, locations, job levels, and job types!")

# Select a job posting
job_index = st.selectbox("Select a Job Posting:", df.index)
selected_context = df['context'].iloc[job_index]
st.subheader("Selected Job Posting Details")
st.text(selected_context)

# Input question from user
user_question = st.text_input("Enter your question:")
generate_button = st.button("Generate Answer")

# Generate and display answer
if generate_button and user_question:
    with st.spinner("Generating answer..."):
        input_text = f"question: {user_question} context: {selected_context}"
        input_ids = tokenizer(input_text, return_tensors="pt").input_ids.to(device)
        output_ids = model.generate(input_ids)
        answer = tokenizer.decode(output_ids[0], skip_special_tokens=True)

        st.subheader("Generated Answer")
        st.success(answer)
```

```
!pip install pyngrok
```

```
!ngrok config add-authtoken 2teSznNUMjFirRLWS43CdupFAiv_4dC43Rp8pB18P2vLej4wJ
```

```
from pyngrok import ngrok

# Set up a tunnel to the Streamlit app
from pyngrok import ngrok

# Correct ngrok connection setup
public_url = ngrok.connect('8501', "http")
print(f"Streamlit app is live at: {public_url}")
```

```
!streamlit run app.py &>/dev/null &
```

Waiting to finish the current execution.

4. Challenges and Resolutions

Key Challenges:

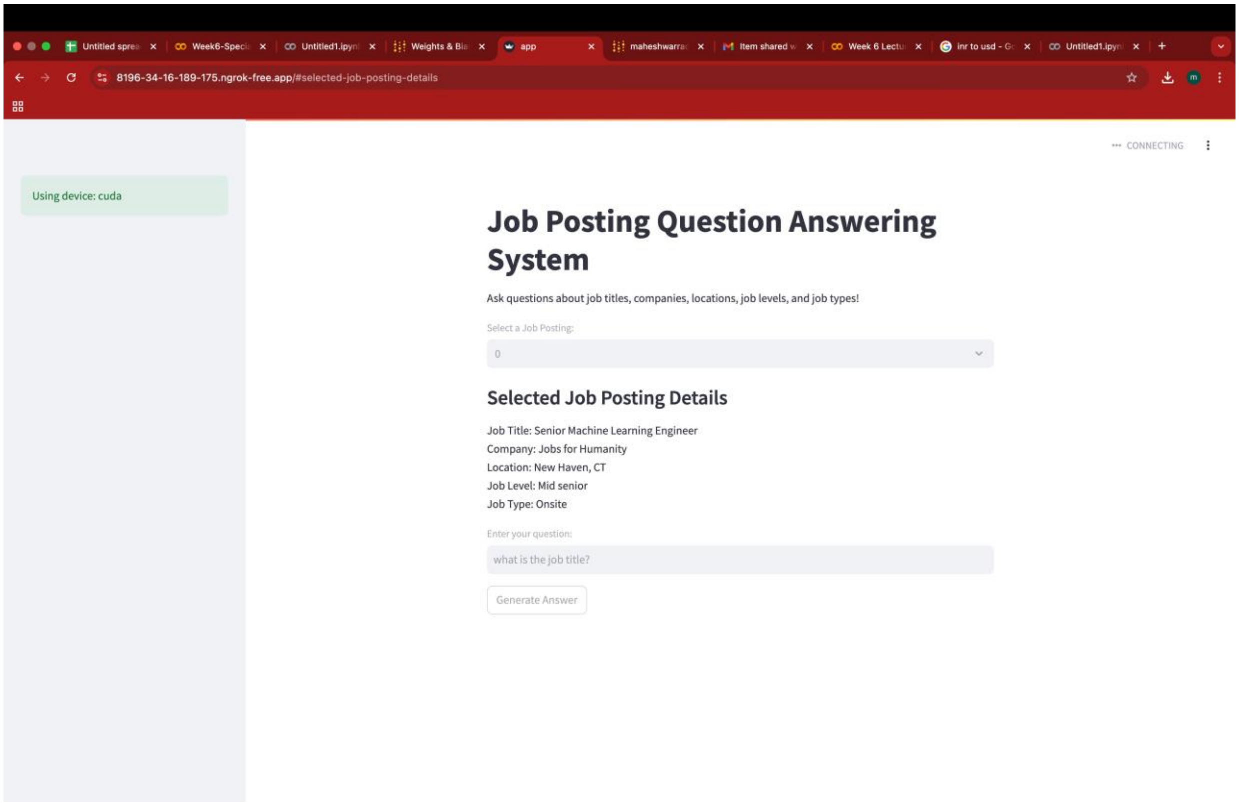
- **KeyError during Dataset Processing**:
  - Issue: Column mismatch in the dataset.
  - Resolution: Adjusted column names to match the expected format.
- **CUDA Out of Memory Error**:

    - Issue: GPU memory limitations.
    - Resolution: Switched to CPU-based inference when GPU was unavailable.
- **Streamlit Installation Problem**:
    - Issue: ModuleNotFoundError for Streamlit.
    - Resolution: Installed Streamlit manually using:

---

5. Conclusion and Results

- The project successfully implemented a Question Answering System using a Decoder-Based Transformer (T5).
- Achievements:
    - Fine-tuned the T5 model for contextual QA on job postings.
    - Developed an interactive Streamlit application for user interaction.
    - Successfully deployed within Colab using ngrok.
- Outcome:
    - Demonstrated the power of transformer-based models in contextual question answering.
    - Showcased the deployment potential using Streamlit and ngrok.



Using device: cuda

Job Posting Question Answering System

Ask questions about job titles, companies, locations, job levels, and job types!

Select a Job Posting:

0

Selected Job Posting Details

Job Title: Senior Machine Learning Engineer
Company: Jobs for Humanity
Location: New Haven, CT
Job Level: Mid senior
Job Type: Onsite

Enter your question:

what is the job title?

Generate Answer

7. Recommendations for Future Work

1. **Model Enhancement**:
    - Experiment with larger T5 models (e.g., t5-base, t5-large) for improved accuracy.
    - Fine-tune with more diverse datasets to enhance generalization.
2. **Feature Improvements**:
    - Add filters for job type, location, and company in the Streamlit UI.
    - Include a feedback system for user satisfaction with generated answers.
3. **Deployment Enhancements**:
    - Deploy on Streamlit Cloud or Heroku for a more stable public URL.
    - Implement CI/CD pipelines for continuous integration and deployment.

8. Conclusion

This project effectively demonstrates the application of Decoder-Based Transformers for building an intelligent QA system. It showcases the integration of T5 models with Streamlit and ngrok for real-time deployment. The system can be further expanded for other domains, including customer support, educational tutoring, and career guidance systems.