Narrative Quest Generation Term Project

Wednesday, April 16, 2025 12:55 PM

For using LLM on CPU
- Use the Distilled version:
  - DistilBERT, DistilGPT2

GPT2
- Create story line and generate adventure

BERT
- Player input understanding

Maybe don't necessarily need to fine tune?
- When to skip fine-tuning
  - Small project, no GPU
  - Model already works well with clever prompts
  - Limited dataset/time
- When to fine-tune
  - Want high relevance
  - Input/output too domain-specific
  - Want consistent accuracy

How to remember player context
- Retrieval System using FAISS (Don't know if this would work well???)
  - Store player history as an embedding in FAISS
    - Sentence Transformers (e.g. all-MiniLM or BERT): embed input + output
    - Makes each sentence a vector
  - For each turn
    - Retrieve similar past events
    - Embed the new player input/output and store into FAISS
    - Use the retrieved similar events in the story generation prompt

BERT extractions:
- Intention
- Direction
- Object
- Target
- Character

We would need to keep track of inventory

https://medium.com/@shaikhrayyan123/a-comprehensive-guide-to-understanding-bert-from-beginners-to-advanced-2379699e2b51

We can use BERT to extract the most important parts of the sentence
- We can also fine tune it to find specific parts of the sentence we are looking for -- Not necessarily JerichoDataset, but other datasets where BERT can find the specific parts we need
- We can maybe find a pretrained/fine-tuned BERT model

Use the output from BERT, and add to vdb (via embedding)
- Query and get the most relevant information of what has happened in the history of the player
- Use relevant information in prompt back to GPT

```python
from transformers import pipeline

# Load the question-answering pipeline with a BERT model
qa = pipeline('question-answering', model='bert-large-uncased-whole-word-masking-finetuned-squad')

# Example paragraph
context = """
Agent Blake infiltrated the facility to retrieve stolen files.
He moved stealthily through the hallways, avoiding guards.
His final objective was the server room on the top floor.
"""

# Questions for each element you want to extract
questions = {
    "Intention": "What was the agent's intention?",
    "Direction": "Which direction did the agent move?",
    "Object": "What object was the agent trying to retrieve?",
    "Target": "What was the agent's final target?",
    "Character": "Who is the character in the paragraph?"
}

# Ask questions using the QA pipeline
for key, question in questions.items():
    result = qa(question=question, context=context)
    print(f"{key}: {result['answer']}")
```

Output:

```
Intention: to retrieve stolen files
Direction: through the hallways
Object: stolen files
Target: the server room
Character: Agent Blake
```

Pipeline:

Player input: Player will type a command
- Extract and summarize element of players response
  - ex. Intent, object, Direction, Target/Character
- Embed context into FAISS
  - Using sentence transformers to embed the Player Input and Models generated output
- Context Retrieval
  - For each turn query FAISS for most similar past interactions
  - Return top-K similar interactions
- Use the context retrieved from FAISS into the prompt of the model

1. Player input
2. Sentence intent extraction (BERT)
3. Embed current state (sentence transformer)
4. FAISS query (player input sentence embedding) -> Similar history
5. FAISS add (player input sentence embedding)
6. Use similar history in prompt - Use last few events + similar history
7. Generate output from prompt
8. Display to player
9. Summarize model output -> embed and input into FAISS

Keep the last k history events
Include most relevant history from FAISS

Should we include player input + language model output into FAISS, or just language model output?

How to keep track of inventory:
Option 1. Using Python array
- Whenever BERT recongizes that a player has the intent to use an object, search the Inventory list to see if it is in the players inventory
- We may also be able to do that with surroundings? i.e. player want to throw rock on ground, so environment = [rock, leaf, …]
  - Use BERT to create a list of what is in the environment around them (?)

Option 2. Using FAISS
- Using BERT we can find out what the player picks up
  - 'Player picked up rock'
- Embed that into FAISS for future reference
- When the player wants to use something, check FAISS to see if it is in their inventory
  - This is probably less dynamic. Might be hard to keep track when the world is evolving so quickly
  - Might be hard to separate player history and player inventory, unless we use two FAISS vdb

Combine these
- Embed the one work item into FAISS, and then query to see if you have anything similar
- Ex. If you have a rock, and the player asks to throw a stone, we know that rock and stone are pretty much the same thing

What do we need
- Find out how to prompt a good language model
  - Maybe decide what the best model to use
- What to include in FAISS vdb
  - Player input + language model output, or just language model output
- Flesh out the idea of inventory using FAISS

- How do we want to use BERT to summarize and give intentions of the text
    - Intention
    - Direction
    - Object
    - Target
-

Timeline
1. Get command line working so the user can prompt language model
2. Interpret intention of player and have language model act accordingly (BERT)
3. FAISS in memory - Remembering player context
4. FAISS for inventory

while:

input -> player inputs
pass into language model
get language model output
print output