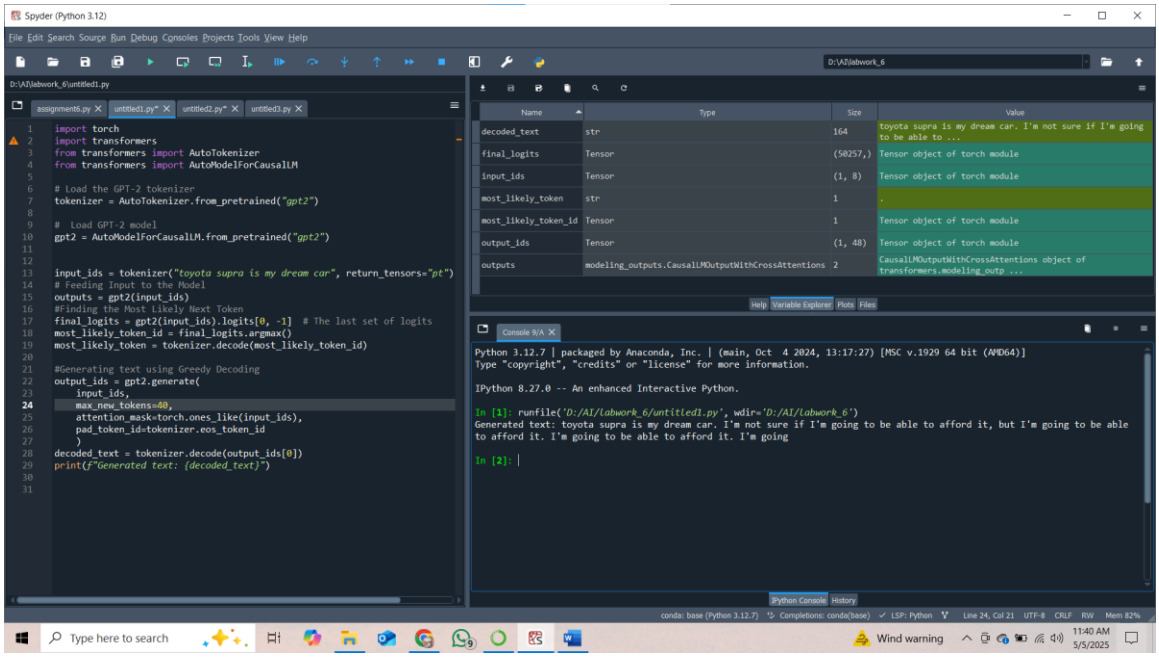
Text Generation Techniques Using GPT-2

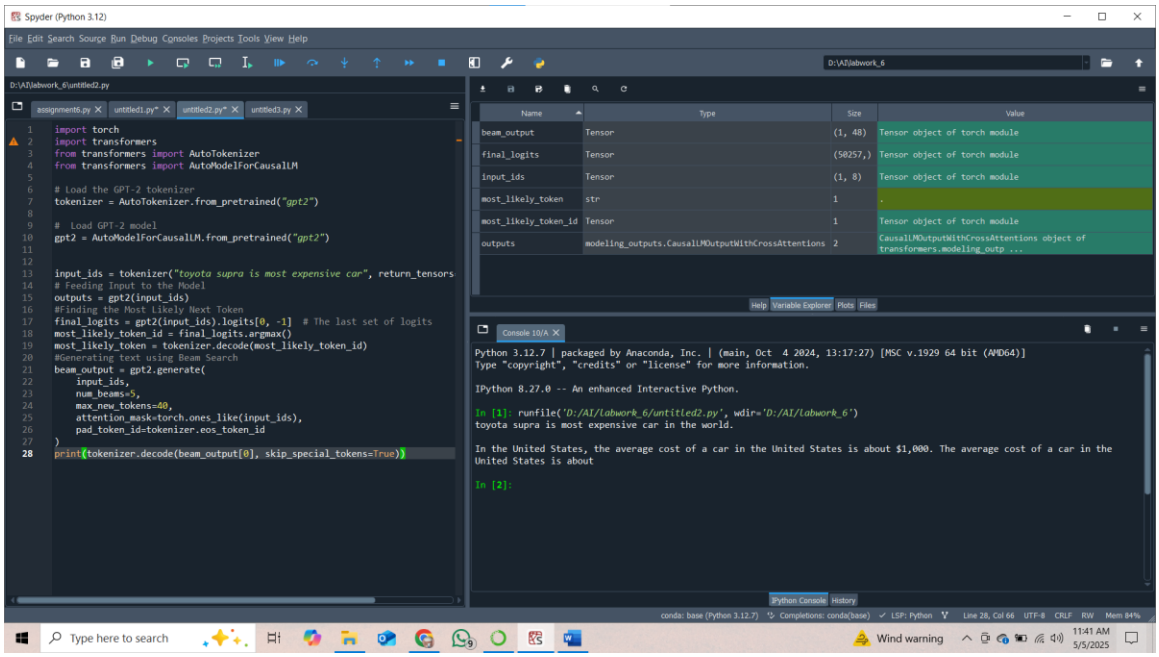
1. Obtained Results

Multiple text generation methods were applied to the phrase "toyota supra is my dream car" using the GPT-2 model. The following decoding strategies and parameters were tested:

1. Greedy Decoding:
  Output: "toyota supra is my dream car. I'm not sure if I'm going to be able to afford it, but I'm going to be able to afford it. I'm going"
  - Observation: Repetitive and overly deterministic. It loops awkwardly.



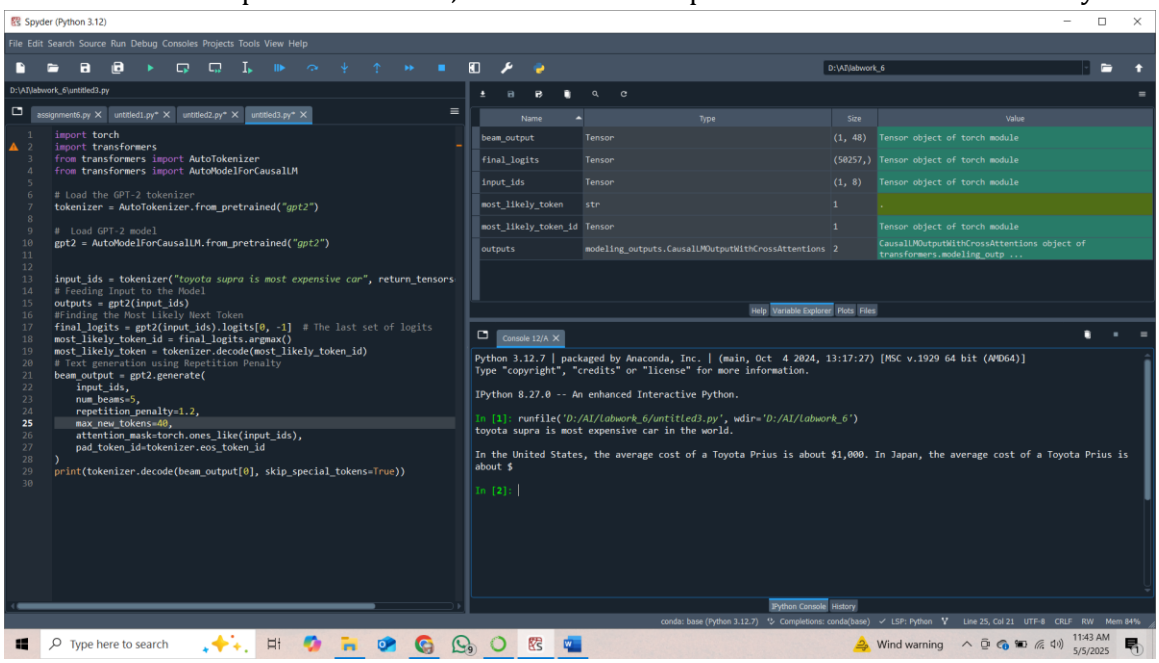
2. Beam Search (num_beams=5):
  Output: "toyota supra is most expensive car in the world. In the United States, the average cost of a car in the United States is about"
  - Observation: Better structure and informativeness, but includes redundancy.



3. Beam Search + Repetition Penalty (penalty=1.2):

Output: "toyota supra is most expensive car. In the United States, the average cost of a Toyota Prius is about $1,000. In Japan, the average cost of a Toyota Prius is"

- Observation: Improved variation; avoids extreme repetition but still has redundancy.

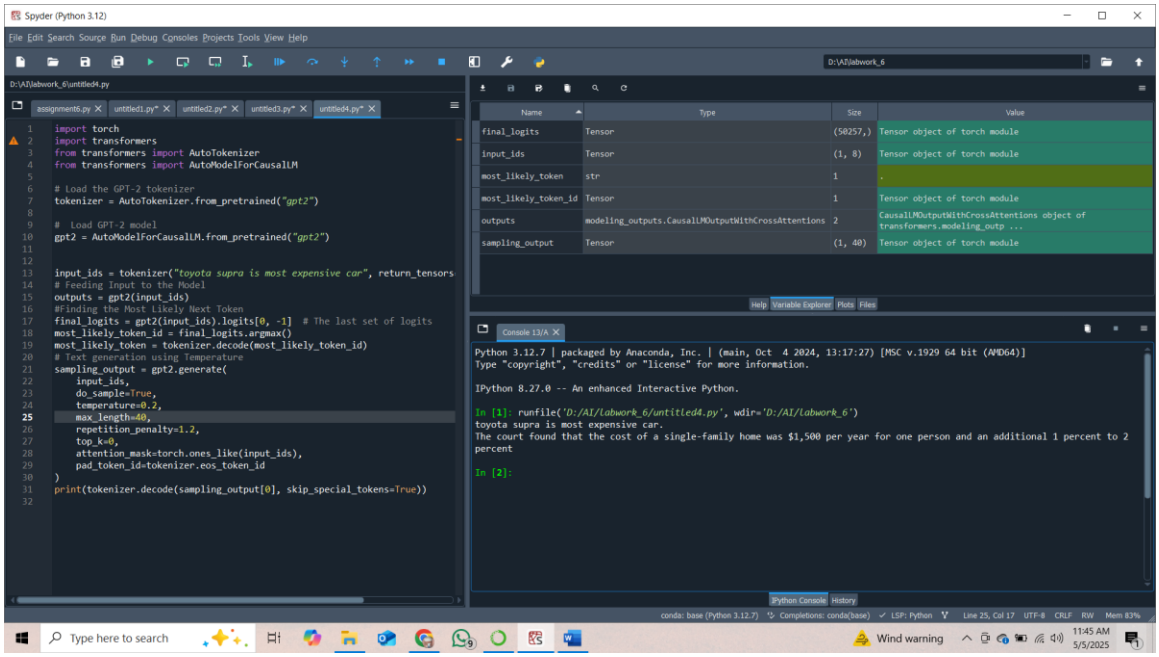


4. Sampling with Temperature (temperature=0.2):

Output: "toyota supra is most expensive car. The court found that the cost of a single-family home was $1,582 per year for one person and an additional 1 percent to 2"

- Observation: Limited randomness and some coherence. Output turns to unrelated

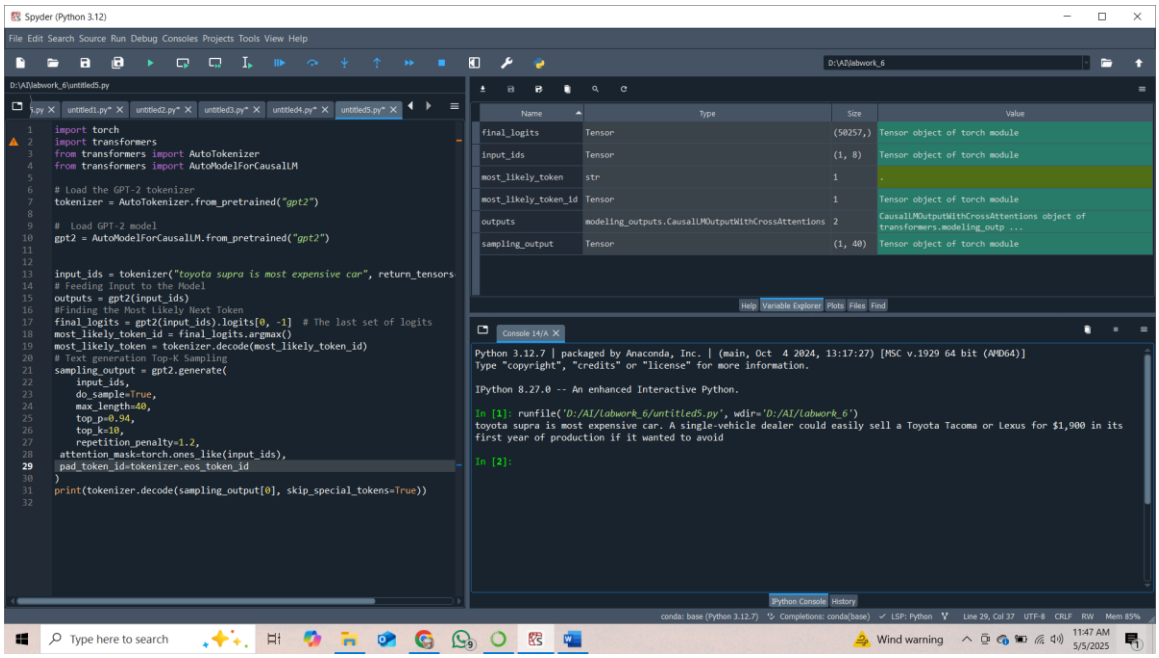
content.



5. Top-K Sampling (top_k=10, top_p=0.94):

Output: "toyota supra is most expensive car. A single-vehicle dealer could easily sell a Toyota Tacoma or Lexus for $1,900 in its first year of production if it wanted to avoid"

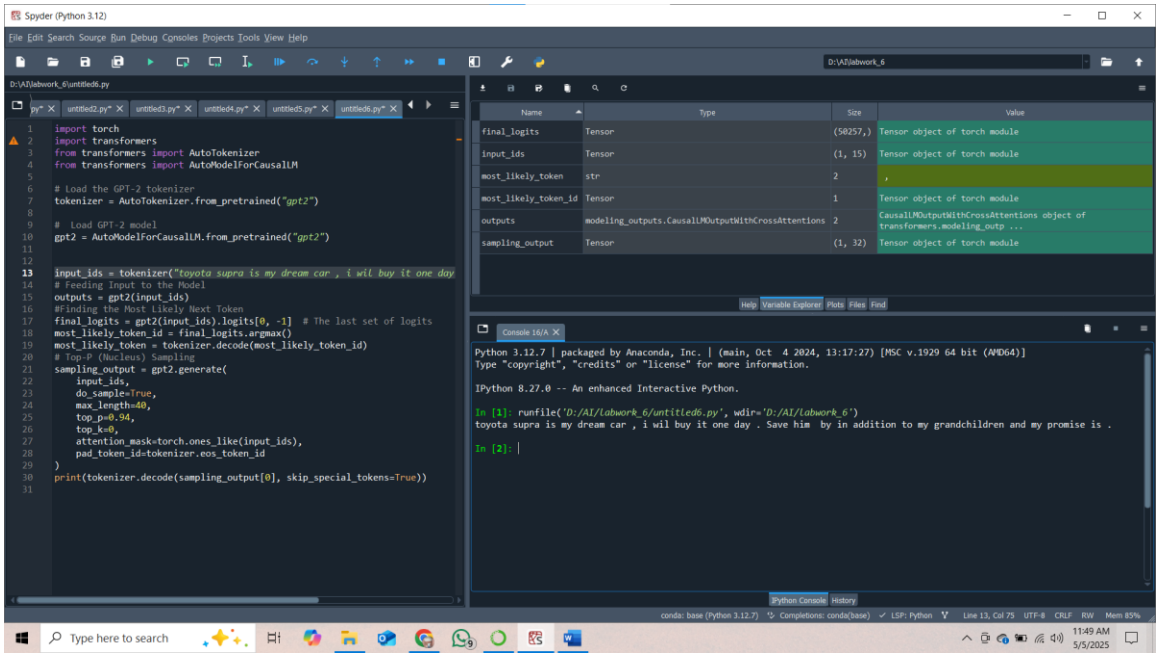
- Observation: Balanced randomness with more specific content.



6. Top-P (Nucleus) Sampling (top_p=0.94, top_k=0):

Output: "toyota supra is my dream car , i wil buy it one day . Save him by in addition to my grandchildren and my promise is ."

- Observation: Highly creative but grammatically incorrect and lacks clarity.



2. Discussion of Results

The outputs reveal key strengths and weaknesses of each decoding strategy:

- Greedy Decoding is fast but repetitive and lacks depth or diversity.
- Beam Search improves fluency but can be redundant without further tuning.
- Adding a Repetition Penalty helps reduce looping patterns and boosts lexical variety.
- Temperature Sampling controls randomness. Lower values like 0.2 give more deterministic outputs but risk monotony and topic drift.
- Top-K Sampling filters vocabulary to the top-k likely words, promoting diversity without losing relevance.
- Top-P (Nucleus) Sampling selects words from a dynamic probability mass, encouraging creativity. However, it can generate odd or incoherent sentences.

3. Conclusions

Each generation method offers trade-offs between structure, diversity, and creativity. For reliable, informative text, Beam Search with repetition penalty is preferred. For exploratory or imaginative tasks, Top-K or Top-P sampling is ideal. The choice depends on the use case and desired output style.

Appendix: Code Used

```
import torch
import transformers
from transformers import AutoTokenizer
from transformers import AutoModelForCausalLM

# Load the GPT-2 tokenizer
tokenizer = AutoTokenizer.from_pretrained("gpt2")

# Load GPT-2 model
gpt2 = AutoModelForCausalLM.from_pretrained("gpt2")

input_ids = tokenizer("toyota supra is my dream car",
return_tensors="pt").input_ids

# Feeding Input to the Model
outputs = gpt2(input_ids)

# Finding the Most Likely Next Token
final_logits = gpt2(input_ids).logits[0, -1]  # The last set of logits
most_likely_token_id = final_logits.argmax()
most_likely_token = tokenizer.decode(most_likely_token_id)

# Generating text using Greedy Decoding
output_ids = gpt2.generate(
    input_ids,
    max_new_tokens=40,
    attention_mask=torch.ones_like(input_ids),
    pad_token_id=tokenizer.eos_token_id
)
decoded_text = tokenizer.decode(output_ids[0])
print(f"Generated text: {decoded_text}")



#i used it step by step as you can see in the screenshot……

#Generating text using Beam Search

beam_output = gpt2.generate(

    input_ids,

    num_beams=5,

    max_new_tokens=40,

    attention_mask=torch.ones_like(input_ids),

    pad_token_id=tokenizer.eos_token_id
```

```
)
print(tokenizer.decode(beam_output[0], skip_special_tokens=True))
# Text generation using Repetition Penalty
beam_output = gpt2.generate(
    input_ids,
    num_beams=5,
    repetition_penalty=1.2,
    max_new_tokens=40,
    attention_mask=torch.ones_like(input_ids),
    pad_token_id=tokenizer.eos_token_id
)
print(tokenizer.decode(beam_output[0], skip_special_tokens=True))
# Text generation using Temperature
sampling_output = gpt2.generate(
    input_ids,
    do_sample=True,
    temperature=0.2,
    max_length=40,
    repetition_penalty=1.2,
    top_k=0,
    attention_mask=torch.ones_like(input_ids),
    pad_token_id=tokenizer.eos_token_id
)
print(tokenizer.decode(sampling_output[0], skip_special_tokens=True))
# Text generation Top-K Sampling
sampling_output = gpt2.generate(
    input_ids,
    do_sample=True,
```

```
    max_length=40,
    top_p=0.94,
    top_k=10,
    repetition_penalty=1.2,
 attention_mask=torch.ones_like(input_ids),
 pad_token_id=tokenizer.eos_token_id
)
print(tokenizer.decode(sampling_output[0], skip_special_tokens=True))
# Top-P (Nucleus) Sampling
sampling_output = gpt2.generate(
    input_ids,
    do_sample=True,
    max_length=40,
    top_p=0.94,
    top_k=0,
    attention_mask=torch.ones_like(input_ids),
    pad_token_id=tokenizer.eos_token_id
)
print(tokenizer.decode(sampling_output[0], skip_special_tokens=True))
```