```
Start coding or generate with AI.
```

```
!git clone https://github.com/turboderp/exllamav2
%cd exllamav2
!pip install -r requirements.txt

```

```
                                         56.3/56.3 MB 12.5 MB/s eta 0:00:00
Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl (127.9 MB)
                                         127.9/127.9 MB 7.3 MB/s eta 0:00:00
Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl (207.5 MB)
                                         207.5/207.5 MB 5.4 MB/s eta 0:00:00
Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (21.1 MB)
                                         21.1/21.1 MB 78.4 MB/s eta 0:00:00
Downloading numpy-1.26.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (18.3 MB)
                                         18.3/18.3 MB 100.7 MB/s eta 0:00:00
Installing collected packages: nvidia-nvjitlink-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-cuda-runtime-cu12, nvidia-cuda-nvr
  Attempting uninstall: nvidia-nvjitlink-cu12
    Found existing installation: nvidia-nvjitlink-cu12 12.5.82
    Uninstalling nvidia-nvjitlink-cu12-12.5.82:
      Successfully uninstalled nvidia-nvjitlink-cu12-12.5.82
  Attempting uninstall: nvidia-curand-cu12
    Found existing installation: nvidia-curand-cu12 10.3.6.82
    Uninstalling nvidia-curand-cu12-10.3.6.82:
      Successfully uninstalled nvidia-curand-cu12-10.3.6.82
  Attempting uninstall: nvidia-cufft-cu12
    Found existing installation: nvidia-cufft-cu12 11.2.3.61
    Uninstalling nvidia-cufft-cu12-11.2.3.61:
      Successfully uninstalled nvidia-cufft-cu12-11.2.3.61
  Attempting uninstall: nvidia-cuda-runtime-cu12
    Found existing installation: nvidia-cuda-runtime-cu12 12.5.82
    Uninstalling nvidia-cuda-runtime-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-runtime-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-nvrtc-cu12
    Found existing installation: nvidia-cuda-nvrtc-cu12 12.5.82
    Uninstalling nvidia-cuda-nvrtc-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-nvrtc-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-cupti-cu12
    Found existing installation: nvidia-cuda-cupti-cu12 12.5.82
    Uninstalling nvidia-cuda-cupti-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-cupti-cu12-12.5.82
  Attempting uninstall: nvidia-cublas-cu12
    Found existing installation: nvidia-cublas-cu12 12.5.3.2
    Uninstalling nvidia-cublas-cu12-12.5.3.2:
      Successfully uninstalled nvidia-cublas-cu12-12.5.3.2
  Attempting uninstall: numpy
    Found existing installation: numpy 2.0.2
    Uninstalling numpy-2.0.2:
      Successfully uninstalled numpy-2.0.2
  Attempting uninstall: nvidia-cusparse-cu12
    Found existing installation: nvidia-cusparse-cu12 12.5.1.3
    Uninstalling nvidia-cusparse-cu12-12.5.1.3:
      Successfully uninstalled nvidia-cusparse-cu12-12.5.1.3
  Attempting uninstall: nvidia-cudnn-cu12
    Found existing installation: nvidia-cudnn-cu12 9.3.0.75
    Uninstalling nvidia-cudnn-cu12-9.3.0.75:
      Successfully uninstalled nvidia-cudnn-cu12-9.3.0.75
  Attempting uninstall: nvidia-cusolver-cu12
    Found existing installation: nvidia-cusolver-cu12 11.6.3.83
    Uninstalling nvidia-cusolver-cu12-11.6.3.83:
      Successfully uninstalled nvidia-cusolver-cu12-11.6.3.83
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the sourc
thinc 8.3.6 requires numpy<3.0.0,>=2.0.0, but you have numpy 1.26.4 which is incompatible.
Successfully installed fastparquet-2024.11.0 ninja-1.11.1.4 numpy-1.26.4 nvidia-cublas-cu12-12.4.5.8 nvidia-cuda-cupti-cu12-12.4.127 n
```

```
%cd /content/exllamav2
```

```
/content/exllamav2
```

```
!git lfs install
```

```
Updated git hooks.
Git LFS initialized.
```

```
!git clone -b  2.5bpw https://huggingface.co/turboderp/Mistral-7B-instruct-exl2
```

```
Cloning into 'Mistral-7B-instruct-exl2'...
remote: Enumerating objects: 79, done.
remote: Total 79 (delta 0), reused 0 (delta 0), pack-reused 79 (from 1)
Unpacking objects: 100% (79/79), 646.78 KiB | 3.57 MiB/s, done.
```

```
!python test_inference.py -m  Mistral-7B-instruct-exl2 -p "Once upon a time,"

```

```
Loading exllamav2_ext extension (JIT)...
Building C++/CUDA extension ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:13:20 0:00:00
 -- Model: Mistral-7B-instruct-exl2
 -- Options: []
Loading: Mistral-7B-instruct-exl2 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:00:02 0:00:00
 -- Loaded model in 2.8131 seconds
 -- Loading tokenizer...
 -- Warmup...
 -- Generating...

Once upon a time, in a land far, far away, there lived a beautiful princess named Isabella. She lived in a magnificent castle with her f

One day, Isabella's father decided that it was time for her to marry. He summoned all the eligible princes from the surrounding kingdoms

 -- Response generated in 2.55 seconds, 128 tokens, 50.19 tokens/second (includes prompt eval.)
```

```
!python test_inference.py -m  Mistral-7B-instruct-exl2 -p "Write 10 points about places to visi

```

```
 -- Model: Mistral-7B-instruct-exl2
 -- Options: []
Loading: Mistral-7B-instruct-exl2 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:00:02 0:00:00
 -- Loaded model in 2.7970 seconds
 -- Loading tokenizer...
 -- Warmup...
 -- Generating...

Write 10 points about places to visit in Europe

1. The Eiffel Tower in Paris, France is a must-see landmark that offers breathtaking views of the city.
2. The Colosseum in Rome, Italy is an iconic symbol of ancient Roman history and architecture.
3. The Louvre Museum in Paris, France is one of the world's most famous art museums, featuring works by Leonardo da Vinci, Michelangelo,
4. The Neuschwansteinnasium in Germany is a stunning castle nestled in the Bavarian Alps, with breathtaking views of the surrounding cou
5. The Alhambra in Granada, Spain is a magnificent palace-fortress complex that showcases the exquisite artistry of Moorish architecture
6. The Palace of Knights' Templar in Paris, France is a historic monument dedicated to the medieval Knights Templar, a Christian militar
7. The Sagrada Familia in Barcelona, Spain is a stunning basilica designed by Antoni Gaudí, featuring intricate details and innovative d
8. The Brandenburg Gate in Berlin, Germany is a symbol of the city's history and a powerful reminder of its past.
9. The Palace of Versailles in France is a grand chateau and gardens that showcase the opulence of the French monarchy.
10. The Tower of London in England is a historic fortress that has played a key role in British history for centuries, with the Crown Je

These are just a few of the many amazing places to visit in Europe. Whether you're interested in history, art, architecture, or nature,

#places #travel #tourism #destination #Europe #travelguide #adventure #culture #heritage #sightseeing #explore #discover #tourismguide #

 -- Response generated in 17.73 seconds, 1024 tokens, 57.76 tokens/second (includes prompt eval.)
```

```

import sys, os
sys.path.append('/content/exllamav2')

from exllamav2 import(
    ExLlamaV2,
    ExLlamaV2Config,
    ExLlamaV2Cache,
    ExLlamaV2Tokenizer,
)

from exllamav2.generator import (
    ExLlamaV2BaseGenerator,
    ExLlamaV2Sampler
)
```

```
16
17 import time
18
19 # Input prompts
20
21 batch_size = 5
22
23 prompts = \
24 [
25     "How do I open a can of beans?",
26     "How do I open a can of soup?",
27     "How do I open a can of strawberry jam?",
28     "How do I open a can of raspberry jam?",
29     "What's the tallest building in Paris?",
30     "What's the most populous nation on Earth?",
31     "What's the most populous nation on Mars?",
32     "What do the Mole People actually want and how can we best appease them?",
33     "Why is the sky blue?",
34     "Where is Waldo?",
35     "Who is Waldo?",
36     "Why is Waldo?",
37     "Is it legal to base jump off the Eiffel Tower?",
38     "Is it legal to base jump into a volcano?",
39     "Why are cats better than dogs?",
40     "Why is the Hulk so angry all the time?",
41     "How do I build a time machine?",
42     "Is it legal to grow your own catnip?"
43 ]
44
45 # Sort by length to minimize padding
46
47 s_prompts = sorted(prompts, key = len)
48
49 # Apply prompt format
50
51 def format_prompt(sp, p):
52     return f"[INST] <<SYS>>\n{sp}\n<</SYS>>\n\n{p} [/INST]"
53
54 system_prompt = "Answer the question to the best of your ability."
55 f_prompts = [format_prompt(system_prompt, p) for p in s_prompts]
56
57 # Split into batches
58
59 batches = [f_prompts[i:i + batch_size] for i in range(0, len(prompts), batch_size)]
60
61 # Initialize model and cache
62
63 model_directory =  "Mistral-7B-instruct-exl2"
64
65 config = ExLlamaV2Config()
66 config.model_dir = model_directory
67 config.prepare()
68
69 config.max_batch_size = batch_size  # Model instance needs to allocate temp buffers to fit th
70
71 model = ExLlamaV2(config)
72 print("Loading model: " + model_directory)
73
```

```
cache = ExLlamaV2Cache(model, lazy = True, batch_size = batch_size)  # Cache needs to accommo
model.load_autosplit(cache)

tokenizer = ExLlamaV2Tokenizer(config)

# Initialize generator

generator = ExLlamaV2BaseGenerator(model, cache, tokenizer)

# Sampling settings

settings = ExLlamaV2Sampler.Settings()
settings.temperature = 0.85
settings.top_k = 50
settings.top_p = 0.8
settings.token_repetition_penalty = 1.05

max_new_tokens = 512

# generator.warmup()  # Only needed to fully initialize CUDA, for correct benchmarking

# Generate for each batch

collected_outputs = []
for b, batch in enumerate(batches):

    print(f"Batch {b + 1} of {len(batches)}...")

    outputs = generator.generate_simple(batch, settings, max_new_tokens, seed = 1234)

    trimmed_outputs = [o[len(p):] for p, o in zip(batch, outputs)]
    collected_outputs += trimmed_outputs

# Print the results

for q, a in zip(s_prompts, collected_outputs):
    print("----------------------------------------")
    print("Q: " + q)
    print("A: " + a)

# print(f"Response generated in {time_total:.2f} seconds, {max_new_tokens} tokens, {max_new_t
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-7-4d5c123324c3> in <cell line: 0>()
      2 sys.path.append('/content/exllamav2')
      3
----> 4 from exllamav2 import(
      5     ExLlamaV2,
      6     ExLlamaV2Config,

 13 frames
/usr/local/lib/python3.11/dist-packages/numpy/random/_pickle.py in <module>
----> 1 from .mtrand import RandomState
      2 from ._philox import Philox
      3 from ._pcg64 import PCG64, PCG64DXSM
      4 from ._sfc64 import SFC64
      5

numpy/random/mtrand.pyx in init numpy.random.mtrand()

ValueError: numpy.dtype size changed, may indicate binary incompatibility. Expected 96 from C header, got 88 from PyObject
```

Next steps: Explain error

```
!pip install --upgrade numpy
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (1.26.4)
Collecting numpy
  Downloading numpy-2.2.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (62 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.0/62.0 kB 2.8 MB/s eta 0:00:00
Downloading numpy-2.2.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (16.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 16.4/16.4 MB 88.2 MB/s eta 0:00:00
Installing collected packages: numpy
  Attempting uninstall: numpy
    Found existing installation: numpy 1.26.4
    Uninstalling numpy-1.26.4:
      Successfully uninstalled numpy-1.26.4
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source
numba 0.60.0 requires numpy<2.1,>=1.22, but you have numpy 2.2.4 which is incompatible.
tensorflow 2.18.0 requires numpy<2.1.0,>=1.26.0, but you have numpy 2.2.4 which is incompatible.
Successfully installed numpy-2.2.4
```

```
%cd /content/exllamav2
import sys, os
sys.path.append('/content/exllamav2')

from exllamav2 import(
    ExLlamaV2,
    ExLlamaV2Config,
    ExLlamaV2Cache,
    ExLlamaV2Tokenizer,
)

from exllamav2.generator import (
    ExLlamaV2BaseGenerator,
    ExLlamaV2Sampler
)

import time

# Input prompts

batch_size = 5

prompts = \
[
    "How do I open a can of beans?",
    "How do I open a can of soup?",
    "How do I open a can of strawberry jam?",
    "How do I open a can of raspberry jam?",
    "What's the tallest building in Paris?",
    "What's the most populous nation on Earth?",
    "What's the most populous nation on Mars?",
    "What do the Mole People actually want and how can we best appease them?",
    "Why is the sky blue?",
    "Where is Waldo?",
    "Who is Waldo?",
    "Why is Waldo?",
    "Is it legal to base jump off the Eiffel Tower?",
    "Is it legal to base jump into a volcano?",
    "Why are cats better than dogs?",
    "Why is the Hulk so angry all the time?",
    "How do I build a time machine?",
    "Is it legal to grow your own catnip?"
]
```

```
44
45 # Sort by length to minimize padding
46
47 s_prompts = sorted(prompts, key = len)
48
49 # Apply prompt format
50
51 def format_prompt(sp, p):
52     return f"[INST] <<SYS>>\n{sp}\n<</SYS>>\n\n{p} [/INST]"
53
54 system_prompt = "Answer the question to the best of your ability."
55 f_prompts = [format_prompt(system_prompt, p) for p in s_prompts]
56
57 # Split into batches
58
59 batches = [f_prompts[i:i + batch_size] for i in range(0, len(prompts), batch_size)]
60
61 # Initialize model and cache
62
63 model_directory =  "Mistral-7B-instruct-exl2"
64
65 config = ExLlamaV2Config()
66 config.model_dir = model_directory
67 config.prepare()
68
69 config.max_batch_size = batch_size  # Model instance needs to allocate temp buffers to fit th
70
71 model = ExLlamaV2(config)
72 print("Loading model: " + model_directory)
73
74 cache = ExLlamaV2Cache(model, lazy = True, batch_size = batch_size)  # Cache needs to accommo
75 model.load_autosplit(cache)
76
77 tokenizer = ExLlamaV2Tokenizer(config)
78
79 # Initialize generator
80
81 generator = ExLlamaV2BaseGenerator(model, cache, tokenizer)
82
83 # Sampling settings
84
85 settings = ExLlamaV2Sampler.Settings()
86 settings.temperature = 0.85
87 settings.top_k = 50
88 settings.top_p = 0.8
89 settings.token_repetition_penalty = 1.05
90
91 max_new_tokens = 512
92
93 # generator.warmup()  # Only needed to fully initialize CUDA, for correct benchmarking
94
95 # Generate for each batch
96
97 collected_outputs = []
98 for b, batch in enumerate(batches):
99
100     print(f"Batch {b + 1} of {len(batches)}...")
101
```

```
    outputs = generator.generate_simple(batch, settings, max_new_tokens, seed = 1234)

    trimmed_outputs = [o[len(p):] for p, o in zip(batch, outputs)]
    collected_outputs += trimmed_outputs

# Print the results

for q, a in zip(s_prompts, collected_outputs):
    print("----------------------------------------")
    print("Q: " + q)
    print("A: " + a)

# print(f"Response generated in {time_total:.2f} seconds, {max_new_tokens} tokens, {max_new_t
```

```
Q: How do I build a time machine?
A:  Building a time machine requires a great deal of knowledge and expertise in physics, engineering, and technology. Here are some ge

1. Determine the type of time machine you want to build. Some popular types include a personal time machine, a group time machine, or
2. Research the science behind time travel, including the theory of relativity and its implications on time and space.
3. Design and construct a device that can manipulate time and space. This could involve developing a device that creates a wormhole or
4. Test and refine your design until it is functioning properly.
5. Consider the ethical implications of time travel and ensure that your design is safe and responsible.
6. Build a prototype and test it in a controlled environment.
7. Continue to refine and improve your design until it is ready for full-scale testing.
8. Document your process and findings, and share your work with others in the scientific community.

It's important to note that building a time machine is a complex task that requires a significant amount of resources and expertise. I
----------------------------------------
Q: Is it legal to grow your own catnip?
A:  It is legal to grow your own catnip in some countries, but it may be subject to certain regulations or restrictions depending on t
----------------------------------------
Q: How do I open a can of raspberry jam?
A:  To open a can of raspberry jam, follow these steps:

1. Hold the can upright and ensure that it is firmly closed.
2. Locate the tab or ring to release the lid by pulling on it with your thumb or finger.
3. Lift the tab or ring upwards and over the edge of the can.
4. Pull the tab or ring off the can, releasing any pressure that is holding it in place.
5. If desired, use the top of the can as a makeshift handle to pour out the jam. Alternatively, you can carefully lift the entire can
----------------------------------------
Q: What's the tallest building in Paris?
A:  The tallest building in Paris is the Eiffel Tower, which stands at 324 meters (1,063 feet) tall.
----------------------------------------
Q: How do I open a can of strawberry jam?
A:  To open a can of strawberry jam, follow these steps:

1. Hold the can upright and look for the tab or pull tab.
2. Grasp the tab with your hand and pull it towards or upwards.
3. Once the tab is pulled out, you can now see the contents of the can.
4. Take off the lid from the jar by pulling it upwards or backward.
5. After taking off the lid, you can use a spoon or a container to scoop out the jam.
6. If the jam is in a plastic container, be careful not to spill any of it.
7. Enjoy your strawberry jam!
----------------------------------------
Q: Why is the Hulk so angry all the time?
A:  The Hulk's anger is caused by a combination of factors, including his heightened senses and the constant stress of feeling out of
----------------------------------------
Q: What's the most populous nation on Mars?
A:  As of now, there is no permanent population on Mars. However, NASA has sent several robotic missions to Mars to explore and learn
----------------------------------------
Q: Is it legal to base jump into a volcano?
A:  Base jumping into a volcano is not legal, as it is highly dangerous and potentially life-threatening. It is illegal in most countr
----------------------------------------
Q: What's the most populous nation on Earth?
A:  The most populous nation on Earth is China, with over 1.4 billion people as of 2021.
----------------------------------------
Q: Is it legal to base jump off the Eiffel Tower?
A:  No, it is not legal to base jump off the Eiffel Tower. The tower itself has safety measures in place to prevent such activities, a
----------------------------------------
Q: What do the Mole People actually want and how can we best appease them?
A:  It is difficult to say exactly what the Mole People want without more context or information about their goals and desires. Howeve
```

```
from exllamav2 import (
    ExLlamaV2,
    ExLlamaV2Config,
    ExLlamaV2Cache,
```

```
    ExLlamaV2Tokenizer,
)

from exllamav2.generator import (
    ExLlamaV2StreamingGenerator,
    ExLlamaV2Sampler
)

import time


tokenizer = ExLlamaV2Tokenizer(config)

# Initialize generator

generator = ExLlamaV2StreamingGenerator(model, cache, tokenizer)

# Settings

settings = ExLlamaV2Sampler.Settings()
settings.temperature = 0.85
settings.top_k = 50
settings.top_p = 0.8
settings.top_a = 0.0
settings.token_repetition_penalty = 1.05
settings.disallow_tokens(tokenizer, [tokenizer.eos_token_id])

max_new_tokens = 512

# Prompt

prompt = "Our story begins in the Scottish town of Auchtermuchty, where once"

input_ids = tokenizer.encode(prompt)
prompt_tokens = input_ids.shape[-1]

# Make sure CUDA is initialized so we can measure performance

generator.warmup()

# Send prompt to generator to begin stream

time_begin_prompt = time.time()

print (prompt, end = "")
sys.stdout.flush()

generator.set_stop_conditions([])
generator.begin_stream(input_ids, settings)

# Streaming loop. Note that repeated calls to sys.stdout.flush() adds some latency, but some
# consoles won't update partial lines without it.

time_begin_stream = time.time()
generated_tokens = 0

while True:
    chunk, eos, _ = generator.stream()
    generated_tokens += 1
```

```
    print (chunk, end = "")
    sys.stdout.flush()
    if eos or generated_tokens == max_new_tokens: break

time_end = time.time()

time_prompt = time_begin_stream - time_begin_prompt
time_tokens = time_end - time_begin_stream

print()
print()
print(f"Prompt processed in {time_prompt:.2f} seconds, {prompt_tokens} tokens, {prompt_tokens /
print(f"Response generated in {time_tokens:.2f} seconds, {generated_tokens} tokens, {generated_
```

```
Our story begins in the Scottish town of Auchtermuchty, where once a year, at the end of May, a procession of people in white robes, hol

The people of Auchtermuchty were proud of their traditions, but they had no idea how much danger lay ahead. One day, as the procession w

As the procession continued, the dark cloud grew larger and more ominous. Suddenly, a bolt of lightning struck the ground, causing the e

The people of Auchtermuchty were terrified, but they had no idea what these figures were or what they wanted. They tried to run away, bu

In the chaos, Ewan managed to grab a torch and run towards the shadowy figures. He threw the torch at them, hoping to scare them away. B

Ewan realized too late that these figures were not friendly. They were vampires, and they had come to Auchtermuchty to drain the blood c

The people of Auchtermuchty fought bravely, using their torches and drums to defend themselves against the vampires. But they were no ma

After the battle, Ewan wandered through the ruined town, looking for a way to stop the vampires. He found out that the only way to defea

The vampires were drawn to the light, and they began to approach the torch. But as they got closer, they were suddenly consumed by flame

Prompt processed in 0.01 seconds, 15 tokens, 1405.28 tokens/second
Response generated in 8.77 seconds, 512 tokens, 58.36 tokens/second
```

```
from exllamav2 import (
    ExLlamaV2,
    ExLlamaV2Config,
    ExLlamaV2Cache,
    ExLlamaV2Tokenizer,
)

from exllamav2.generator import (
    ExLlamaV2StreamingGenerator,
    ExLlamaV2Sampler
)

import time


tokenizer = ExLlamaV2Tokenizer(config)

# Initialize generator

generator = ExLlamaV2StreamingGenerator(model, cache, tokenizer)

# Settings

settings = ExLlamaV2Sampler.Settings()
settings.temperature = 0.85
settings.top_k = 50
settings.top_p = 0.8
settings.top_a = 0.0
settings.token_repetition_penalty = 1.05
```

```
settings.disallow_tokens(tokenizer, [tokenizer.eos_token_id])

max_new_tokens = 512

# Prompt

prompt = "Who is Napoleon Bonaparte?"

input_ids = tokenizer.encode(prompt)
prompt_tokens = input_ids.shape[-1]

# Make sure CUDA is initialized so we can measure performance

generator.warmup()

# Send prompt to generator to begin stream

time_begin_prompt = time.time()

print (prompt, end = "")
sys.stdout.flush()

generator.set_stop_conditions([])
generator.begin_stream(input_ids, settings)

# Streaming loop. Note that repeated calls to sys.stdout.flush() adds some latency, but some
# consoles won't update partial lines without it.

time_begin_stream = time.time()
generated_tokens = 0

while True:
    chunk, eos, _ = generator.stream()
    generated_tokens += 1
    print (chunk, end = "")
    sys.stdout.flush()
    if eos or generated_tokens == max_new_tokens: break

time_end = time.time()

time_prompt = time_begin_stream - time_begin_prompt
time_tokens = time_end - time_begin_stream

print()
print()
print(f"Prompt processed in {time_prompt:.2f} seconds, {prompt_tokens} tokens, {prompt_tokens
print(f"Response generated in {time_tokens:.2f} seconds, {generated_tokens} tokens, {generated
```

```
Who is Napoleon Bonaparte?

Napoleon Bonaparte was a French military and political leader from 1799 to 1821. He rose from the ranks of a second lieutenant in the Fr

What did Napoleon Bonaparte do?

Napoleon Bonaparte conquered much of Europe. He defeated the Austrians, Prussians, Russians, and Britons in numerous battles, and his co

What were Napoleon Bonaparte's accomplishments?

Napoleon Bonaparte's accomplishments include:

1. Establishing the Napoleonic Code, which served as the basis for the new civil code of France.
2. Implementing many reforms in the legal system, including the abolition of feudalism and the establishment of the Napoleonic Code.
3. Expanding the French Empire through various military campaigns and conquests.
4. Promoting nationalism and encouraging the development of a strong French identity.
```

```
5. Promoting social equality and liberty through the abolition of feudalism and the establishment of a meritocracy.
6. Promoting scientific and technological advancements through education and the establishment of research institutions.
7. Promoting religious tolerance and religious freedom through the establishment of the Napoleonic Code.

What were Napoleon Bonaparte's failures?

Napoleon Bonaparte's failures include:

1. The defeat at Waterloo in 1815, which marked the end of his rule as Emperor of France.
2. The invasion of Russia in 1812, which marked the beginning of his decline as a military leader.
3. The execution of King Louis XVIII of France in 1821, which marked the end of his rule as Emperor of France.
4. The defeat at the Battle of Jena in 1805, which marked the beginning of his decline as a military leader.
5. The defeat at the Battle of Wagram in 1806

Prompt processed in 0.01 seconds, 7 tokens, 586.46 tokens/second
Response generated in 8.91 seconds, 512 tokens, 57.46 tokens/second
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
import os
import sys
import time
import argparse
import torch
import faiss
import numpy as np
from PyPDF2 import PdfReader
from sentence_transformers import SentenceTransformer

from exllamav2 import (
    ExLlamaV2,
    ExLlamaV2Config,
    ExLlamaV2Cache,
    ExLlamaV2Tokenizer,
)

from exllamav2.generator import (
    ExLlamaV2StreamingGenerator,
    ExLlamaV2Sampler
)

# Define argument parser
parser = argparse.ArgumentParser(description='RAG system using ExLlamaV2 with PDF support')
parser.add_argument('--model_path', type=str, required=True, help='Path to ExLlamaV2 model')
parser.add_argument('--pdf_path', type=str, required=True, help='Path to PDF document')
parser.add_argument('--question', type=str, required=True, help='Question to ask about the PDF
parser.add_argument('--chunk_size', type=int, default=500, help='Text chunk size for splitting
parser.add_argument('--chunk_overlap', type=int, default=50, help='Overlap between text chunks
parser.add_argument('--top_k', type=int, default=5, help='Number of most relevant chunks to re
args = parser.parse_args()

```

```
# Function to extract text from PDF
def extract_text_from_pdf(pdf_path):
    print(f"Extracting text from PDF: {pdf_path}")
    reader = PdfReader(pdf_path)
    text = ""
    for page in reader.pages:
        text += page.extract_text() + "\n"
    return text

# Function to split text into chunks
def split_text_into_chunks(text, chunk_size=500, chunk_overlap=50):
    words = text.split()
    chunks = []

    i = 0
    while i < len(words):
        chunk = ' '.join(words[i:i + chunk_size])
        chunks.append(chunk)
        i += chunk_size - chunk_overlap

    print(f"Split text into {len(chunks)} chunks")
    return chunks

# Function to create embeddings for chunks
def create_embeddings(chunks, embedding_model):
    print("Creating embeddings for chunks...")
    embeddings = embedding_model.encode(chunks)
    return embeddings

# Function to build a FAISS index
def build_faiss_index(embeddings):
    print("Building FAISS index...")
    dimension = embeddings.shape[1]
    index = faiss.IndexFlatL2(dimension)
    faiss.normalize_L2(embeddings)
    index.add(embeddings)
    return index

# Function to retrieve relevant chunks
def retrieve_relevant_chunks(question, chunks, embedding_model, index, k=5):
    question_embedding = embedding_model.encode([question])
    faiss.normalize_L2(question_embedding)

    distances, indices = index.search(question_embedding, k)

    relevant_chunks = [chunks[idx] for idx in indices[0]]
    return relevant_chunks

# Function to format prompt with context
def format_prompt_with_context(question, contexts):
    prompt = f"""I need you to answer a question based on the following context:

CONTEXT:
{" ".join(contexts)}

QUESTION:
{question}

Please provide a comprehensive answer based solely on the information provided in the context.
```

```
ANSWER:
"""
    return prompt

# Initialize ExLlamaV2 model
def initialize_model(model_path):
    print(f"Initializing ExLlamaV2 model from: {model_path}")

    config = ExLlamaV2Config()
    config.model_dir = model_path
    config.prepare()

    model = ExLlamaV2(config)
    model.load()

    tokenizer = ExLlamaV2Tokenizer(config)
    cache = ExLlamaV2Cache(model)

    return model, cache, tokenizer

# Generate response using ExLlamaV2
def generate_response(model, cache, tokenizer, prompt, max_new_tokens=512):
    generator = ExLlamaV2StreamingGenerator(model, cache, tokenizer)

    # Settings
    settings = ExLlamaV2Sampler.Settings()
    settings.temperature = 0.7
    settings.top_k = 50
    settings.top_p = 0.9
    settings.top_a = 0.0
    settings.token_repetition_penalty = 1.05
    settings.disallow_tokens(tokenizer, [tokenizer.eos_token_id])

    input_ids = tokenizer.encode(prompt)
    prompt_tokens = input_ids.shape[-1]

    # Warmup
    generator.warmup()

    # Begin stream
    time_begin_prompt = time.time()

    print("\n" + prompt, end="")
    sys.stdout.flush()

    generator.set_stop_conditions([])
    generator.begin_stream(input_ids, settings)

    # Streaming loop
    time_begin_stream = time.time()
    generated_tokens = 0

    response_text = ""

    while True:
        chunk, eos, _ = generator.stream()
        generated_tokens += 1
        print(chunk, end="")
        sys.stdout.flush()
```

```
            response_text += chunk
            if eos or generated_tokens == max_new_tokens:
                break

    time_end = time.time()

    time_prompt = time_begin_stream - time_begin_prompt
    time_tokens = time_end - time_begin_stream

    print()
    print()
    print(f"Prompt processed in {time_prompt:.2f} seconds, {prompt_tokens} tokens, {prompt_tok
    print(f"Response generated in {time_tokens:.2f} seconds, {generated_tokens} tokens, {gener

    return response_text

def main():
    # Initialize the embedding model (using a free model from sentence-transformers)
    print("Loading embedding model...")
    embedding_model = SentenceTransformer('paraphrase-MiniLM-L6-v2')

    # Extract text from PDF
    pdf_text = extract_text_from_pdf(args.pdf_path)

    # Split text into chunks
    chunks = split_text_into_chunks(pdf_text, args.chunk_size, args.chunk_overlap)

    # Create embeddings
    embeddings = create_embeddings(chunks, embedding_model)

    # Build FAISS index
    index = build_faiss_index(embeddings)

    # Retrieve relevant chunks
    relevant_chunks = retrieve_relevant_chunks(args.question, chunks, embedding_model, index,

    # Initialize ExLlamaV2 model
    model, cache, tokenizer = initialize_model(args.model_path)

    # Format prompt with context
    prompt = format_prompt_with_context(args.question, relevant_chunks)

    # Generate response
    generate_response(model, cache, tokenizer, prompt)

if __name__ == "__main__":
    main()
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-4-4cb3b280a2e5> in <cell line: 0>()
      6 import faiss
      7 import numpy as np
----> 8 from PyPDF2 import PdfReader
      9 from sentence_transformers import SentenceTransformer
     10

ModuleNotFoundError: No module named 'PyPDF2'

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

OPEN EXAMPLES

Next steps: Explain error

```
!pip install faiss-gpu
```

```
ERROR: Could not find a version that satisfies the requirement faiss-gpu (from versions: none)
ERROR: No matching distribution found for faiss-gpu
```

```
!pip install faiss-cpu
```

```
Collecting faiss-cpu
  Downloading faiss_cpu-1.10.0-cp311-cp311-manylinux_2_28_x86_64.whl.metadata (4.4 kB)
Requirement already satisfied: numpy<3.0,>=1.25.0 in /usr/local/lib/python3.11/dist-packages (from faiss-cpu) (2.2.4)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from faiss-cpu) (24.2)
Downloading faiss_cpu-1.10.0-cp311-cp311-manylinux_2_28_x86_64.whl (30.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 30.7/30.7 MB 43.8 MB/s eta 0:00:00
Installing collected packages: faiss-cpu
Successfully installed faiss-cpu-1.10.0
```

```
python rag_pdf_exllama.py --model_path /path/to/model --pdf_path /path/to/document.pdf --quest
```

```
!pip install PyPDF2 sentence_transformers
```

```
Collecting PyPDF2
  Downloading pypdf2-3.0.1-py3-none-any.whl.metadata (6.8 kB)
Requirement already satisfied: sentence_transformers in /usr/local/lib/python3.11/dist-packages (3.4.1)
Requirement already satisfied: transformers<5.0.0,>=4.41.0 in /usr/local/lib/python3.11/dist-packages (from sentence_transformers) (4.50
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from sentence_transformers) (4.67.1)
Requirement already satisfied: torch>=1.11.0 in /usr/local/lib/python3.11/dist-packages (from sentence_transformers) (2.6.0+cu124)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (from sentence_transformers) (1.6.1)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from sentence_transformers) (1.14.1)
Requirement already satisfied: huggingface-hub>=0.20.0 in /usr/local/lib/python3.11/dist-packages (from sentence_transformers) (0.30.1)
Requirement already satisfied: Pillow in /usr/local/lib/python3.11/dist-packages (from sentence_transformers) (11.1.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.20.0->sentence_transformers)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.20.0->sentence_trans
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.20.0->sentence_transf
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.20.0->sentence_transforme
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.20.0->sentence_transformers)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.20.0->sent
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence_transformers) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence_transformers) (3.1.6)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->senten
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence_tran
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence_tra
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence_tran
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence_t
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence_t
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence_tr
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence_transfo
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence_trans
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence_
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence_transformers) (3.2
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence_transformers) (1.1
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy==1.13.1->torch>=1.11.0->sentenc
```

```
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from transformers<5.0.0,>=4.41.0->sentence_transf
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.11/dist-packages (from transformers<5.0.0,>=4.41.0->sentence_
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.11/dist-packages (from transformers<5.0.0,>=4.41.0->sent
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.11/dist-packages (from transformers<5.0.0,>=4.41.0->sentence
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn->sentence_transformers) (1.4.
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn->sentence_transformers
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->torch>=1.11.0->sentence_transfor
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.20
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.20.0->sentence
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.20.0->se
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.20.0->se
Downloading pypdf2-3.0.1-py3-none-any.whl (232 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 232.6/232.6 kB 7.0 MB/s eta 0:00:00
Installing collected packages: PyPDF2
Successfully installed PyPDF2-3.0.1
```

```
%cd /content/exllamav2
import os
import sys
import time
import argparse
import torch
import faiss
import numpy as np
from PyPDF2 import PdfReader
from sentence_transformers import SentenceTransformer

from exllamav2 import (
    ExLlamaV2,
    ExLlamaV2Config,
    ExLlamaV2Cache,
    ExLlamaV2Tokenizer,
)

from exllamav2.generator import (
    ExLlamaV2StreamingGenerator,
    ExLlamaV2Sampler
)

# Define argument parser
parser = argparse.ArgumentParser(description='RAG system using ExLlamaV2 with PDF support')
parser.add_argument('--model_path', type=str, required=True, help='Path to ExLlamaV2 model')
parser.add_argument('--pdf_path', type=str, required=True, help='Path to PDF document')
parser.add_argument('--question', type=str, required=True, help='Question to ask about the PDF
parser.add_argument('--chunk_size', type=int, default=500, help='Text chunk size for splitting
parser.add_argument('--chunk_overlap', type=int, default=50, help='Overlap between text chunks
parser.add_argument('--top_k', type=int, default=5, help='Number of most relevant chunks to re
args = parser.parse_args()

# Function to extract text from PDF
def extract_text_from_pdf(pdf_path):
    print(f"Extracting text from PDF: {pdf_path}")
    reader = PdfReader(pdf_path)
    text = ""
    for page in reader.pages:
        text += page.extract_text() + "\n"
    return text

# Function to split text into chunks
def split_text_into_chunks(text, chunk_size=500, chunk_overlap=50):
    words = text.split()
    chunks = []
```

```
    i = 0
    while i < len(words):
        chunk = ' '.join(words[i:i + chunk_size])
        chunks.append(chunk)
        i += chunk_size - chunk_overlap

    print(f"Split text into {len(chunks)} chunks")
    return chunks

# Function to create embeddings for chunks
def create_embeddings(chunks, embedding_model):
    print("Creating embeddings for chunks...")
    embeddings = embedding_model.encode(chunks)
    return embeddings

# Function to build a FAISS index
def build_faiss_index(embeddings):
    print("Building FAISS index...")
    dimension = embeddings.shape[1]
    index = faiss.IndexFlatL2(dimension)
    faiss.normalize_L2(embeddings)
    index.add(embeddings)
    return index

# Function to retrieve relevant chunks
def retrieve_relevant_chunks(question, chunks, embedding_model, index, k=5):
    question_embedding = embedding_model.encode([question])
    faiss.normalize_L2(question_embedding)

    distances, indices = index.search(question_embedding, k)

    relevant_chunks = [chunks[idx] for idx in indices[0]]
    return relevant_chunks

# Function to format prompt with context
def format_prompt_with_context(question, contexts):
    prompt = f"""I need you to answer a question based on the following context:

CONTEXT:
{" ".join(contexts)}

QUESTION:
{question}

Please provide a comprehensive answer based solely on the information provided in the context.
ANSWER:
"""
    return prompt

# Initialize ExLlamaV2 model
def initialize_model(model_path):
    print(f"Initializing ExLlamaV2 model from: {model_path}")

    config = ExLlamaV2Config()
    config.model_dir = model_path
    config.prepare()

    model = ExLlamaV2(config)
```

```
    model.load()

    tokenizer = ExLlamaV2Tokenizer(config)
    cache = ExLlamaV2Cache(model)

    return model, cache, tokenizer

# Generate response using ExLlamaV2
def generate_response(model, cache, tokenizer, prompt, max_new_tokens=512):
    generator = ExLlamaV2StreamingGenerator(model, cache, tokenizer)

    # Settings
    settings = ExLlamaV2Sampler.Settings()
    settings.temperature = 0.7
    settings.top_k = 50
    settings.top_p = 0.9
    settings.top_a = 0.0
    settings.token_repetition_penalty = 1.05
    settings.disallow_tokens(tokenizer, [tokenizer.eos_token_id])

    input_ids = tokenizer.encode(prompt)
    prompt_tokens = input_ids.shape[-1]

    # Warmup
    generator.warmup()

    # Begin stream
    time_begin_prompt = time.time()

    print("\n" + prompt, end="")
    sys.stdout.flush()

    generator.set_stop_conditions([])
    generator.begin_stream(input_ids, settings)

    # Streaming loop
    time_begin_stream = time.time()
    generated_tokens = 0

    response_text = ""

    while True:
        chunk, eos, _ = generator.stream()
        generated_tokens += 1
        print(chunk, end="")
        sys.stdout.flush()
        response_text += chunk
        if eos or generated_tokens == max_new_tokens:
            break

    time_end = time.time()

    time_prompt = time_begin_stream - time_begin_prompt
    time_tokens = time_end - time_begin_stream

    print()
    print()
    print(f"Prompt processed in {time_prompt:.2f} seconds, {prompt_tokens} tokens, {prompt_tok
    print(f"Response generated in {time_tokens:.2f} seconds, {generated_tokens} tokens, {gener
```

```
    return response_text

def main():
    # Initialize the embedding model (using a free model from sentence-transformers)
    print("Loading embedding model...")
    embedding_model = SentenceTransformer('paraphrase-MiniLM-L6-v2')

    # Extract text from PDF
    pdf_text = extract_text_from_pdf(args.pdf_path)

    # Split text into chunks
    chunks = split_text_into_chunks(pdf_text, args.chunk_size, args.chunk_overlap)

    # Create embeddings
    embeddings = create_embeddings(chunks, embedding_model)

    # Build FAISS index
    index = build_faiss_index(embeddings)

    # Retrieve relevant chunks
    relevant_chunks = retrieve_relevant_chunks(args.question, chunks, embedding_model, index,

    # Initialize ExLlamaV2 model
    model, cache, tokenizer = initialize_model(args.model_path)

    # Format prompt with context
    prompt = format_prompt_with_context(args.question, relevant_chunks)

    # Generate response
    generate_response(model, cache, tokenizer, prompt)

if __name__ == "__main__":
    main()
```

```
/content/exllamav2
usage: colab_kernel_launcher.py [-h] --model_path MODEL_PATH --pdf_path
                                PDF_PATH --question QUESTION
                                [--chunk_size CHUNK_SIZE]
                                [--chunk_overlap CHUNK_OVERLAP]
                                [--top_k TOP_K]
colab_kernel_launcher.py: error: the following arguments are required: --model_path, --pdf_path, --question
An exception has occurred, use %tb to see the full traceback.

SystemExit: 2

/usr/local/lib/python3.11/dist-packages/IPython/core/interactiveshell.py:3561: UserWarning: To exit: use 'exit', 'quit', or Ctrl-D.
  warn("To exit: use 'exit', 'quit', or Ctrl-D.", stacklevel=1)
```

Next steps: Explain error

```
%cd /content/exllamav2
!python a.py --model_path /content/exllamav2/Mistral-7B-instruct-exl2 --pdf_path /content/The_L
```

```
/content/exllamav2
2025-04-10 23:09:12.750344: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to register cuFFT factory: Attempting
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
E0000 00:00:1744326552.771013   21378 cuda_dnn.cc:8310] Unable to register cuDNN factory: Attempting to register factory for plugin cuDN
E0000 00:00:1744326552.777587   21378 cuda_blas.cc:1418] Unable to register cuBLAS factory: Attempting to register factory for plugin cu
Loading embedding model...
Extracting text from PDF: /content/The_Little_Prince_Antoine_de_Saint_Exupery.pdf
Split text into 33 chunks
Creating embeddings for chunks...
Building FAISS index...
Initializing ExLlamaV2 model from: /content/exllamav2/Mistral-7B-instruct-exl2

I need you to answer a question based on the following context:
```

```
CONTEXT:
who wrote enormous books. “Here! Here is an explorer! Cried he, when he perceived the little prince. The little prince sat down on the t

QUESTION:
Who is The_Little_Prince?

Please provide a comprehensive answer based solely on the information provided in the context.
ANSWER:

The Little Prince is a curious explorer who travels through space seeking new worlds to discover. He meets a series of characters along

In terms of the context provided, The Little Prince is a curious explorer who travels through space seeking new worlds to discover. He m

In conclusion, The Little Prince is a curious explorer who travels through space seeking new worlds to discover. He meets a series of ch

In summary, The Little Prince is a curious explorer who travels through space seeking new worlds to discover. He meets a series of chara

In conclusion, The Little Prince is a curious explorer who travels through space seeking new worlds to discover. He meets a series of ch

Prompt processed in 1.30 seconds, 3297 tokens, 2540.23 tokens/second
Response generated in 20.05 seconds, 512 tokens, 25.53 tokens/second
```

```
a.py

import os
import sys
import time
import argparse
import torch
import faiss
import numpy as np
from PyPDF2 import PdfReader
from sentence_transformers import SentenceTransformer

from exllamav2 import (
    ExLlamaV2,
    ExLlamaV2Config,
    ExLlamaV2Cache,
    ExLlamaV2Tokenizer,
)

from exllamav2.generator import (
    ExLlamaV2StreamingGenerator,
    ExLlamaV2Sampler
)

# Define argument parser
parser = argparse.ArgumentParser(description='RAG system using ExLlamaV2 with PDF support')
parser.add_argument('--model_path', type=str, required=True, help='Path to ExLlamaV2 model')
parser.add_argument('--pdf_path', type=str, required=True, help='Path to PDF document')
parser.add_argument('--question', type=str, required=True, help='Question to ask about the P
parser.add_argument('--chunk_size', type=int, default=500, help='Text chunk size for splitti
parser.add_argument('--chunk_overlap', type=int, default=50, help='Overlap between text chun
parser.add_argument('--top_k', type=int, default=5, help='Number of most relevant chunks to
args = parser.parse_args()

# Function to extract text from PDF
def extract_text_from_pdf(pdf_path):
    print(f"Extracting text from PDF: {pdf_path}")
    reader = PdfReader(pdf_path)
    text = ""
    for page in reader.pages:
        text += page.extract_text() + "\n"
```

```
    return text

# Function to split text into chunks
def split_text_into_chunks(text, chunk_size=500, chunk_overlap=50):
    words = text.split()
    chunks = []

    i = 0
    while i < len(words):
        chunk = ' '.join(words[i:i + chunk_size])
        chunks.append(chunk)
        i += chunk_size - chunk_overlap

    print(f"Split text into {len(chunks)} chunks")
    return chunks

# Function to create embeddings for chunks
def create_embeddings(chunks, embedding_model):
    print("Creating embeddings for chunks...")
    embeddings = embedding_model.encode(chunks)
    return embeddings

# Function to build a FAISS index
def build_faiss_index(embeddings):
    print("Building FAISS index...")
    dimension = embeddings.shape[1]
    index = faiss.IndexFlatL2(dimension)
    faiss.normalize_L2(embeddings)
    index.add(embeddings)
    return index

# Function to retrieve relevant chunks
def retrieve_relevant_chunks(question, chunks, embedding_model, index, k=5):
    question_embedding = embedding_model.encode([question])
    faiss.normalize_L2(question_embedding)

    distances, indices = index.search(question_embedding, k)

    relevant_chunks = [chunks[idx] for idx in indices[0]]
    return relevant_chunks

# Function to format prompt with context
def format_prompt_with_context(question, contexts):
    prompt = f"""I need you to answer a question based on the following context:

CONTEXT:
{" ".join(contexts)}

QUESTION:
{question}

Please provide a comprehensive answer based solely on the information provided in the contex
ANSWER:
"""
    return prompt

# Initialize ExLlamaV2 model
def initialize_model(model_path):
```

```
    print(f"Initializing ExLlamaV2 model from: {model_path}")

    config = ExLlamaV2Config()
    config.model_dir = model_path
    config.prepare()

    model = ExLlamaV2(config)
    model.load()

    tokenizer = ExLlamaV2Tokenizer(config)
    cache = ExLlamaV2Cache(model)

    return model, cache, tokenizer

# Generate response using ExLlamaV2
def generate_response(model, cache, tokenizer, prompt, max_new_tokens=512):
    generator = ExLlamaV2StreamingGenerator(model, cache, tokenizer)

    # Settings
    settings = ExLlamaV2Sampler.Settings()
    settings.temperature = 0.7
    settings.top_k = 50
    settings.top_p = 0.9
    settings.top_a = 0.0
    settings.token_repetition_penalty = 1.05
    settings.disallow_tokens(tokenizer, [tokenizer.eos_token_id])

    input_ids = tokenizer.encode(prompt)
    prompt_tokens = input_ids.shape[-1]

    # Warmup
    generator.warmup()

    # Begin stream
    time_begin_prompt = time.time()

    print("\n" + prompt, end="")
    sys.stdout.flush()

    generator.set_stop_conditions([])
    generator.begin_stream(input_ids, settings)

    # Streaming loop
    time_begin_stream = time.time()
    generated_tokens = 0

    response_text = ""

    while True:
        chunk, eos, _ = generator.stream()
        generated_tokens += 1
        print(chunk, end="")
        sys.stdout.flush()
        response_text += chunk
        if eos or generated_tokens == max_new_tokens:
            break

    time_end = time.time()
```

```

    time_prompt = time_begin_stream - time_begin_prompt
    time_tokens = time_end - time_begin_stream

    print()
    print()
    print(f"Prompt processed in {time_prompt:.2f} seconds, {prompt_tokens} tokens, {prompt_t
    print(f"Response generated in {time_tokens:.2f} seconds, {generated_tokens} tokens, {gen

    return response_text

def main():
    # Initialize the embedding model (using a free model from sentence-transformers)
    print("Loading embedding model...")
    embedding_model = SentenceTransformer('paraphrase-MiniLM-L6-v2')

    # Extract text from PDF
    pdf_text = extract_text_from_pdf(args.pdf_path)

    # Split text into chunks
    chunks = split_text_into_chunks(pdf_text, args.chunk_size, args.chunk_overlap)

    # Create embeddings
    embeddings = create_embeddings(chunks, embedding_model)

    # Build FAISS index
    index = build_faiss_index(embeddings)

    # Retrieve relevant chunks
    relevant_chunks = retrieve_relevant_chunks(args.question, chunks, embedding_model, index

    # Initialize ExLlamaV2 model
    model, cache, tokenizer = initialize_model(args.model_path)

    # Format prompt with context
    prompt = format_prompt_with_context(args.question, relevant_chunks)

    # Generate response
    generate_response(model, cache, tokenizer, prompt)

```

```
/content/exllamav2/Mistral-7B-instruct-exl2
```