

1 Start coding or generate with AI.

```
1 !git clone https://github.com/turboderp/exllamav2
2 %cd exllamav2
3 !pip install -r requirements.txt
4
5
```

```
→ Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl (127.9 MB)
   56.3/56.3 MB 12.5 MB/s eta 0:00:00
→ Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl (207.5 MB)
   127.9/127.9 MB 7.3 MB/s eta 0:00:00
→ Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (21.1 MB)
   207.5/207.5 MB 5.4 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)
   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-nrv
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 source of the known "Unresolved dependency" errors.
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
```

```
1 %cd /content/exllamav2
```

```
→ /content/exllamav2
```

```
1 !git lfs install
```

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

```
1 !git clone -b 2.5bpw https://huggingface.co/turboderp/Mistral-7B-instruct-ex12
```

```
→ Cloning into 'Mistral-7B-instruct-ex12'...
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.
```

```
1 !python test_inference.py -m Mistral-7B-instruct-ex12 -p "Once upon a time,"
```

```
2
```

```
→ Loading exllamav2_ext extension (JIT)...
Building C++/CUDA extension ━━━━━━━━ 100% 0:13:20 0:00:00
-- Model: Mistral-7B-instruct-ex12
-- Options: []
Loading: Mistral-7B-instruct-ex12 ━━━━━━ 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 father. 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.)

```
1 !python test_inference.py -m Mistral-7B-instruct-ex12 -p "Write 10 points about places to visit in Europe"
```

```
→ -- Model: Mistral-7B-instruct-ex12
-- Options: []
Loading: Mistral-7B-instruct-ex12 ━━━━━━ 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, and Raffaello Sanzio.
4. The Neuschwanstein Castle in Germany is a stunning castle nestled in the Bavarian Alps, with breathtaking views of the surrounding countryside.
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 military order.
7. The Sagrada Família in Barcelona, Spain is a stunning basilica designed by Antoni Gaudí, featuring intricate details and innovative engineering.
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 Jewels still housed within its walls.

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.)
```

```
1
2 import sys, os
3 sys.path.append('/content/exllamav2')
4
5 from exllamav2 import(
6     ExLlamaV2,
7     ExLlamaV2Config,
8     ExLlamaV2Cache,
9     ExLlamaV2Tokenizer,
10 )
11
12 from exllamav2.generator import (
13     ExLlamaV2BaseGenerator,
14     ExLlamaV2Sampler
15 )
```

```
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] <>{sp}<>\n{p}\n[/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-ex12"
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
102    outputs = generator.generate_simple(batch, settings, max_new_tokens, seed = 1234)
103
104    trimmed_outputs = [o[len(p):] for p, o in zip(batch, outputs)]
105    collected_outputs += trimmed_outputs
106
107 # Print the results
108
109 for q, a in zip(s_prompts, collected_outputs):
110    print("-----")
111    print("Q: " + q)
112    print("A: " + a)
113
114 # 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](#)

```
1 !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
```

```
1 %cd /content/exllamav2
2 import sys, os
3 sys.path.append('/content/exllamav2')
4
5 from exllamav2 import(
6     ExLlamaV2,
7     ExLlamaV2Config,
8     ExLlamaV2Cache,
9     ExLlamaV2Tokenizer,
10 )
11
12 from exllamav2.generator import (
13     ExLlamaV2BaseGenerator,
14     ExLlamaV2Sampler
15 )
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] <>{sp}<>\n</INST>\n\n{p}"
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-ex12"
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
```

```
102     outputs = generator.generate_simple(batch, settings, max_new_tokens, seed = 1234)
103
104     trimmed_outputs = [o[len(p):] for p, o in zip(batch, outputs)]
105     collected_outputs += trimmed_outputs
106
107 # Print the results
108
109 for q, a in zip(s_prompts, collected_outputs):
110     print("-----")
111     print("Q: " + q)
112     print("A: " + a)
113
114 # 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 general steps:

1. Determine the type of time machine you want to build. Some popular types include a personal time machine, a group time machine, or a wormhole.
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 a time machine.
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 the country.

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 control.

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 more about the planet.

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 countries.

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, although there have been instances of illegal base jumping.

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. However, they are often depicted as being interested in human flesh and bones.

```
1 from exllamav2 import (
2     ExLlamaV2,
3     ExLlamaV2Config,
4     ExLlamaV2Cache,
```

```
5     ExLlamaV2Tokenizer,
6 )
7
8 from exllamav2.generator import (
9     ExLlamaV2StreamingGenerator,
10    ExLlamaV2Sampler
11 )
12
13 import time
14
15
16 tokenizer = ExLlamaV2Tokenizer(config)
17
18 # Initialize generator
19
20 generator = ExLlamaV2StreamingGenerator(model, cache, tokenizer)
21
22 # Settings
23
24 settings = ExLlamaV2Sampler.Settings()
25 settings.temperature = 0.85
26 settings.top_k = 50
27 settings.top_p = 0.8
28 settings.top_a = 0.0
29 settings.token_repetition_penalty = 1.05
30 settings.disallow_tokens(tokenizer, [tokenizer.eos_token_id])
31
32 max_new_tokens = 512
33
34 # Prompt
35
36 prompt = "Our story begins in the Scottish town of Auchtermuchty, where once"
37
38 input_ids = tokenizer.encode(prompt)
39 prompt_tokens = input_ids.shape[-1]
40
41 # Make sure CUDA is initialized so we can measure performance
42
43 generator.warmup()
44
45 # Send prompt to generator to begin stream
46
47 time_begin_prompt = time.time()
48
49 print(prompt, end="")
50 sys.stdout.flush()
51
52 generator.set_stop_conditions([])
53 generator.begin_stream(input_ids, settings)
54
55 # Streaming loop. Note that repeated calls to sys.stdout.flush() adds some latency, but some
56 # consoles won't update partial lines without it.
57
58 time_begin_stream = time.time()
59 generated_tokens = 0
60
61 while True:
62     chunk, eos, _ = generator.stream()
63     generated_tokens += 1
```

```
64     print(chunk, end = "")  
65     sys.stdout.flush()  
66     if eos or generated_tokens == max_new_tokens: break  
67  
68 time_end = time.time()  
69  
70 time_prompt = time_begin_stream - time_begin_prompt  
71 time_tokens = time_end - time_begin_stream  
72  
73 print()  
74 print()  
75 print(f"Prompt processed in {time_prompt:.2f} seconds, {prompt_tokens} tokens, {prompt_tokens / time_prompt:.2f} tokens/second")  
76 print(f"Response generated in {time_tokens:.2f} seconds, {generated_tokens} tokens, {generated_tokens / time_tokens:.2f} tokens/second")
```

→ 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, hold

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

```
1 from exllamav2 import (  
2     ExLlamaV2,  
3     ExLlamaV2Config,  
4     ExLlamaV2Cache,  
5     ExLlamaV2Tokenizer,  
6 )  
7  
8 from exllamav2.generator import (  
9     ExLlamaV2StreamingGenerator,  
10    ExLlamaV2Sampler  
11 )  
12  
13 import time  
14  
15  
16 tokenizer = ExLlamaV2Tokenizer(config)  
17  
18 # Initialize generator  
19  
20 generator = ExLlamaV2StreamingGenerator(model, cache, tokenizer)  
21  
22 # Settings  
23  
24 settings = ExLlamaV2Sampler.Settings()  
25 settings.temperature = 0.85  
26 settings.top_k = 50  
27 settings.top_p = 0.8  
28 settings.top_a = 0.0  
29 settings.token_repetition_penalty = 1.05
```

```
30 settings.disallow_tokens(tokenizer, [tokenizer.eos_token_id])
31
32 max_new_tokens = 512
33
34 # Prompt
35
36 prompt = "Who is Napoleon Bonaparte?"
37
38 input_ids = tokenizer.encode(prompt)
39 prompt_tokens = input_ids.shape[-1]
40
41 # Make sure CUDA is initialized so we can measure performance
42
43 generator.warmup()
44
45 # Send prompt to generator to begin stream
46
47 time_begin_prompt = time.time()
48
49 print (prompt, end = "")
50 sys.stdout.flush()
51
52 generator.set_stop_conditions([])
53 generator.begin_stream(input_ids, settings)
54
55 # Streaming loop. Note that repeated calls to sys.stdout.flush() adds some latency, but some
56 # consoles won't update partial lines without it.
57
58 time_begin_stream = time.time()
59 generated_tokens = 0
60
61 while True:
62     chunk, eos, _ = generator.stream()
63     generated_tokens += 1
64     print (chunk, end = "")
65     sys.stdout.flush()
66     if eos or generated_tokens == max_new_tokens: break
67
68 time_end = time.time()
69
70 time_prompt = time_begin_stream - time_begin_prompt
71 time_tokens = time_end - time_begin_stream
72
73 print()
74 print()
75 print(f"Prompt processed in {time_prompt:.2f} seconds, {prompt_tokens} tokens, {prompt_tokens}
76 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

1 Start coding or generate with AI.

```
1 import os
2 import sys
3 import time
4 import argparse
5 import torch
6 import faiss
7 import numpy as np
8 from PyPDF2 import PdfReader
9 from sentence_transformers import SentenceTransformer
10
11 from exllamav2 import (
12     ExLlamaV2,
13     ExLlamaV2Config,
14     ExLlamaV2Cache,
15     ExLlamaV2Tokenizer,
16 )
17
18 from exllamav2.generator import (
19     ExLlamaV2StreamingGenerator,
20     ExLlamaV2Sampler
21 )
22
23 # Define argument parser
24 parser = argparse.ArgumentParser(description='RAG system using ExLlamaV2 with PDF support')
25 parser.add_argument('--model_path', type=str, required=True, help='Path to ExLlamaV2 model')
26 parser.add_argument('--pdf_path', type=str, required=True, help='Path to PDF document')
27 parser.add_argument('--question', type=str, required=True, help='Question to ask about the PDF')
28 parser.add_argument('--chunk_size', type=int, default=500, help='Text chunk size for splitting')
29 parser.add_argument('--chunk_overlap', type=int, default=50, help='Overlap between text chunks')
30 parser.add_argument('--top_k', type=int, default=5, help='Number of most relevant chunks to re
31 args = parser.parse_args()
32
```

```
33 # Function to extract text from PDF
34 def extract_text_from_pdf(pdf_path):
35     print(f"Extracting text from PDF: {pdf_path}")
36     reader = PdfReader(pdf_path)
37     text = ""
38     for page in reader.pages:
39         text += page.extract_text() + "\n"
40     return text
41
42 # Function to split text into chunks
43 def split_text_into_chunks(text, chunk_size=500, chunk_overlap=50):
44     words = text.split()
45     chunks = []
46     ...
47     i = 0
48     while i < len(words):
49         chunk = ''.join(words[i:i + chunk_size])
50         chunks.append(chunk)
51         i += chunk_size - chunk_overlap
52     ...
53     print(f"Split text into {len(chunks)} chunks")
54     return chunks
55
56 # Function to create embeddings for chunks
57 def create_embeddings(chunks, embedding_model):
58     print("Creating embeddings for chunks...")
59     embeddings = embedding_model.encode(chunks)
60     return embeddings
61
62 # Function to build a FAISS index
63 def build_faiss_index(embeddings):
64     print("Building FAISS index...")
65     dimension = embeddings.shape[1]
66     index = faiss.IndexFlatL2(dimension)
67     faiss.normalize_L2(embeddings)
68     index.add(embeddings)
69     return index
70
71 # Function to retrieve relevant chunks
72 def retrieve_relevant_chunks(question, chunks, embedding_model, index, k=5):
73     question_embedding = embedding_model.encode([question])
74     faiss.normalize_L2(question_embedding)
75     ...
76     distances, indices = index.search(question_embedding, k)
77     ...
78     relevant_chunks = [chunks[idx] for idx in indices[0]]
79     return relevant_chunks
80
81 # Function to format prompt with context
82 def format_prompt_with_context(question, contexts):
83     prompt = f"""I need you to answer a question based on the following context:
84
85 CONTEXT:
86 {" ".join(contexts)}
87
88 QUESTION:
89 {question}
90
91 Please provide a comprehensive answer based solely on the information provided in the context.
```

```
92 ANSWER:  
93 """  
94     return prompt  
95  
96 # Initialize ExLlamaV2 model  
97 def initialize_model(model_path):  
98     print(f"Initializing ExLlamaV2 model from: {model_path}")  
99  
100    config = ExLlamaV2Config()  
101    config.model_dir = model_path  
102    config.prepare()  
103  
104    model = ExLlamaV2(config)  
105    model.load()  
106  
107    tokenizer = ExLlamaV2Tokenizer(config)  
108    cache = ExLlamaV2Cache(model)  
109  
110    return model, cache, tokenizer  
111  
112 # Generate response using ExLlamaV2  
113 def generate_response(model, cache, tokenizer, prompt, max_new_tokens=512):  
114     generator = ExLlamaV2StreamingGenerator(model, cache, tokenizer)  
115  
116     # Settings  
117     settings = ExLlamaV2Sampler.Settings()  
118     settings.temperature = 0.7  
119     settings.top_k = 50  
120     settings.top_p = 0.9  
121     settings.top_a = 0.0  
122     settings.token_repetition_penalty = 1.05  
123     settings.disallow_tokens(tokenizer, [tokenizer.eos_token_id])  
124  
125     input_ids = tokenizer.encode(prompt)  
126     prompt_tokens = input_ids.shape[-1]  
127  
128     # Warmup  
129     generator.warmup()  
130  
131     # Begin stream  
132     time_begin_prompt = time.time()  
133  
134     print("\n" + prompt, end="")  
135     sys.stdout.flush()  
136  
137     generator.set_stop_conditions([])  
138     generator.begin_stream(input_ids, settings)  
139  
140     # Streaming loop  
141     time_begin_stream = time.time()  
142     generated_tokens = 0  
143  
144     response_text = ""  
145  
146     while True:  
147         chunk, eos, _ = generator.stream()  
148         generated_tokens += 1  
149         print(chunk, end="")  
150         sys.stdout.flush()
```

```
151 ..... response_text += chunk
152 ..... if eos or generated_tokens == max_new_tokens:
153 .....     break
154 .....
155 ..... time_end = time.time()
156 .....
157 ..... time_prompt = time_begin_stream - time_begin_prompt
158 ..... time_tokens = time_end - time_begin_stream
159 .....
160 ..... print()
161 ..... print()
162 ..... print(f"Prompt processed in {time_prompt:.2f} seconds, {prompt_tokens} tokens, {prompt_tok}
163 ..... print(f"Response generated in {time_tokens:.2f} seconds, {generated_tokens} tokens, {gener
164 .....
165 ..... return response_text
166
167 def main():
168 ..... # Initialize the embedding model (using a free model from sentence-transformers)
169 ..... print("Loading embedding model...")
170 ..... embedding_model = SentenceTransformer('paraphrase-MiniLM-L6-v2')
171 .....
172 ..... # Extract text from PDF
173 ..... pdf_text = extract_text_from_pdf(args.pdf_path)
174 .....
175 ..... # Split text into chunks
176 ..... chunks = split_text_into_chunks(pdf_text, args.chunk_size, args.chunk_overlap)
177 .....
178 ..... # Create embeddings
179 ..... embeddings = create_embeddings(chunks, embedding_model)
180 .....
181 ..... # Build FAISS index
182 ..... index = build_faiss_index(embeddings)
183 .....
184 ..... # Retrieve relevant chunks
185 ..... relevant_chunks = retrieve_relevant_chunks(args.question, chunks, embedding_model, index,
186 .....
187 ..... # Initialize ExLlamaV2 model
188 ..... model, cache, tokenizer = initialize_model(args.model_path)
189 .....
190 ..... # Format prompt with context
191 ..... prompt = format_prompt_with_context(args.question, relevant_chunks)
192 .....
193 ..... # Generate response
194 ..... generate_response(model, cache, tokenizer, prompt)
195
196 if __name__ == "__main__":
197 ..... 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](#)

```

1 !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

1 !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

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

1 !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) (4.56)
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_transformer)
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->sente)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sente)
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence_tr)
Requirement already satisfied: nvidia-cUBLAS-cu12==12.4.5.8 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence_tr)
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence_tr)
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-cusparsse-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_transfc)
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->sente

```

```
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from transformers<5.0.0,>=4.41.0->sentence_transformers)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.11/dist-packages (from transformers<5.0.0,>=4.41.0->sentence_transformers)
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.11/dist-packages (from transformers<5.0.0,>=4.41.0->sentence_transformers)
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.11/dist-packages (from transformers<5.0.0,>=4.41.0->sentence_transformers)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn->sentence_transformers) (1.4.1)
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_transformers)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.20.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.20.0->sentence_transformers)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.20.0->sentence_transformers)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.20.0->sentence_transformers)
Downloaded 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

```
1 %cd /content/exllamav2
2 import os
3 import sys
4 import time
5 import argparse
6 import torch
7 import faiss
8 import numpy as np
9 from PyPDF2 import PdfReader
10 from sentence_transformers import SentenceTransformer
11
12 from exllamav2 import (
13     ExLlamaV2,
14     ExLlamaV2Config,
15     ExLlamaV2Cache,
16     ExLlamaV2Tokenizer,
17 )
18
19 from exllamav2.generator import (
20     ExLlamaV2StreamingGenerator,
21     ExLlamaV2Sampler
22 )
23
24 # Define argument parser
25 parser = argparse.ArgumentParser(description='RAG system using ExLlamaV2 with PDF support')
26 parser.add_argument('--model_path', type=str, required=True, help='Path to ExLlamaV2 model')
27 parser.add_argument('--pdf_path', type=str, required=True, help='Path to PDF document')
28 parser.add_argument('--question', type=str, required=True, help='Question to ask about the PDF')
29 parser.add_argument('--chunk_size', type=int, default=500, help='Text chunk size for splitting')
30 parser.add_argument('--chunk_overlap', type=int, default=50, help='Overlap between text chunks')
31 parser.add_argument('--top_k', type=int, default=5, help='Number of most relevant chunks to return')
32 args = parser.parse_args()
33
34 # Function to extract text from PDF
35 def extract_text_from_pdf(pdf_path):
36     print(f"Extracting text from PDF: {pdf_path}")
37     reader = PdfReader(pdf_path)
38     text = ""
39     for page in reader.pages:
40         text += page.extract_text() + "\n"
41     return text
42
43 # Function to split text into chunks
44 def split_text_into_chunks(text, chunk_size=500, chunk_overlap=50):
45     words = text.split()
46     chunks = []
47
```

```
+,
48     i = 0
49     while i < len(words):
50         chunk = ' '.join(words[i:i + chunk_size])
51         chunks.append(chunk)
52         i += chunk_size - chunk_overlap
53
54     print(f"Split text into {len(chunks)} chunks")
55     return chunks
56
57 # Function to create embeddings for chunks
58 def create_embeddings(chunks, embedding_model):
59     print("Creating embeddings for chunks...")
60     embeddings = embedding_model.encode(chunks)
61     return embeddings
62
63 # Function to build a FAISS index
64 def build_faiss_index(embeddings):
65     print("Building FAISS index...")
66     dimension = embeddings.shape[1]
67     index = faiss.IndexFlatL2(dimension)
68     faiss.normalize_L2(embeddings)
69     index.add(embeddings)
70     return index
71
72 # Function to retrieve relevant chunks
73 def retrieve_relevant_chunks(question, chunks, embedding_model, index, k=5):
74     question_embedding = embedding_model.encode([question])
75     faiss.normalize_L2(question_embedding)
76
77     distances, indices = index.search(question_embedding, k)
78
79     relevant_chunks = [chunks[idx] for idx in indices[0]]
80     return relevant_chunks
81
82 # Function to format prompt with context
83 def format_prompt_with_context(question, contexts):
84     prompt = f"""I need you to answer a question based on the following context:
85
86 CONTEXT:
87 {" ".join(contexts)}
88
89 QUESTION:
90 {question}
91
92 Please provide a comprehensive answer based solely on the information provided in the context.
93 ANSWER:
94 """
95     return prompt
96
97 # Initialize ExLlamaV2 model
98 def initialize_model(model_path):
99     print(f"Initializing ExLlamaV2 model from: {model_path}")
100
101    config = ExLlamaV2Config()
102    config.model_dir = model_path
103    config.prepare()
104
105    model = ExLlamaV2(config)
106    ...
```

```
106     model.load()
107
108     tokenizer = ExLlamaV2Tokenizer(config)
109     cache = ExLlamaV2Cache(model)
110
111     return model, cache, tokenizer
112
113 # Generate response using ExLlamaV2
114 def generate_response(model, cache, tokenizer, prompt, max_new_tokens=512):
115     generator = ExLlamaV2StreamingGenerator(model, cache, tokenizer)
116
117     # Settings
118     settings = ExLlamaV2Sampler.Settings()
119     settings.temperature = 0.7
120     settings.top_k = 50
121     settings.top_p = 0.9
122     settings.top_a = 0.0
123     settings.token_repetition_penalty = 1.05
124     settings.disallow_tokens(tokenizer, [tokenizer.eos_token_id])
125
126     input_ids = tokenizer.encode(prompt)
127     prompt_tokens = input_ids.shape[-1]
128
129     # Warmup
130     generator.warmup()
131
132     # Begin stream
133     time_begin_prompt = time.time()
134
135     print("\n" + prompt, end="")
136     sys.stdout.flush()
137
138     generator.set_stop_conditions([])
139     generator.begin_stream(input_ids, settings)
140
141     # Streaming loop
142     time_begin_stream = time.time()
143     generated_tokens = 0
144
145     response_text = ""
146
147     while True:
148         chunk, eos, _ = generator.stream()
149         generated_tokens += 1
150         print(chunk, end="")
151         sys.stdout.flush()
152         response_text += chunk
153         if eos or generated_tokens == max_new_tokens:
154             break
155
156     time_end = time.time()
157
158     time_prompt = time_begin_stream - time_begin_prompt
159     time_tokens = time_end - time_begin_stream
160
161     print()
162     print()
163     print(f"Prompt processed in {time_prompt:.2f} seconds, {prompt_tokens} tokens, {prompt_tok}
164     print(f"Response generated in {time_tokens:.2f} seconds, {generated_tokens} tokens, {gener
```

```

165
166     return response_text
167
168 def main():
169     # Initialize the embedding model (using a free model from sentence-transformers)
170     print("Loading embedding model...")
171     embedding_model = SentenceTransformer('paraphrase-MiniLM-L6-v2')
172
173     # Extract text from PDF
174     pdf_text = extract_text_from_pdf(args.pdf_path)
175
176     # Split text into chunks
177     chunks = split_text_into_chunks(pdf_text, args.chunk_size, args.chunk_overlap)
178
179     # Create embeddings
180     embeddings = create_embeddings(chunks, embedding_model)
181
182     # Build FAISS index
183     index = build_faiss_index(embeddings)
184
185     # Retrieve relevant chunks
186     relevant_chunks = retrieve_relevant_chunks(args.question, chunks, embedding_model, index,
187
188     # Initialize ExLlamaV2 model
189     model, cache, tokenizer = initialize_model(args.model_path)
190
191     # Format prompt with context
192     prompt = format_prompt_with_context(args.question, relevant_chunks)
193
194     # Generate response
195     generate_response(model, cache, tokenizer, prompt)
196
197 if __name__ == "__main__":
198     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](#)

1 %cd /content/exllamav2

2 !python a.py --model_path /content/exllamav2/Mistral-7B-instruct-ex12 --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 cuDNN
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-ex12

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

```
1 a.py
2
3 import os
4 import sys
5 import time
6 import argparse
7 import torch
8 import faiss
9 import numpy as np
10 from PyPDF2 import PdfReader
11 from sentence_transformers import SentenceTransformer
12
13 from exllamav2 import (
14     ExLlamaV2,
15     ExLlamaV2Config,
16     ExLlamaV2Cache,
17     ExLlamaV2Tokenizer,
18 )
19
20 from exllamav2.generator import (
21     ExLlamaV2StreamingGenerator,
22     ExLlamaV2Sampler
23 )
24
25 # Define argument parser
26 parser = argparse.ArgumentParser(description='RAG system using ExLlamaV2 with PDF support')
27 parser.add_argument('--model_path', type=str, required=True, help='Path to ExLlamaV2 model')
28 parser.add_argument('--pdf_path', type=str, required=True, help='Path to PDF document')
29 parser.add_argument('--question', type=str, required=True, help='Question to ask about the P
30 parser.add_argument('--chunk_size', type=int, default=500, help='Text chunk size for splitti
31 parser.add_argument('--chunk_overlap', type=int, default=50, help='Overlap between text chun
32 parser.add_argument('--top_k', type=int, default=5, help='Number of most relevant chunks to
33 args = parser.parse_args()
34
35 # Function to extract text from PDF
36 def extract_text_from_pdf(pdf_path):
37     print(f"Extracting text from PDF: {pdf_path}")
38     reader = PdfReader(pdf_path)
39     text = ""
40     for page in reader.pages:
41         text += page.extract_text() + "\n"
```

```
42     return text
43
44 # Function to split text into chunks
45 def split_text_into_chunks(text, chunk_size=500, chunk_overlap=50):
46     words = text.split()
47     chunks = []
48
49     i = 0
50     while i < len(words):
51         chunk = ' '.join(words[i:i + chunk_size])
52         chunks.append(chunk)
53         i += chunk_size - chunk_overlap
54
55     print(f"Split text into {len(chunks)} chunks")
56     return chunks
57
58 # Function to create embeddings for chunks
59 def create_embeddings(chunks, embedding_model):
60     print("Creating embeddings for chunks...")
61     embeddings = embedding_model.encode(chunks)
62     return embeddings
63
64 # Function to build a FAISS index
65 def build_faiss_index(embeddings):
66     print("Building FAISS index...")
67     dimension = embeddings.shape[1]
68     index = faiss.IndexFlatL2(dimension)
69     faiss.normalize_L2(embeddings)
70     index.add(embeddings)
71     return index
72
73 # Function to retrieve relevant chunks
74 def retrieve_relevant_chunks(question, chunks, embedding_model, index, k=5):
75     question_embedding = embedding_model.encode([question])
76     faiss.normalize_L2(question_embedding)
77
78     distances, indices = index.search(question_embedding, k)
79
80     relevant_chunks = [chunks[idx] for idx in indices[0]]
81     return relevant_chunks
82
83 # Function to format prompt with context
84 def format_prompt_with_context(question, contexts):
85     prompt = f"""I need you to answer a question based on the following context:
86
87 CONTEXT:
88 {" ".join(contexts)}
89
90 QUESTION:
91 {question}
92
93 Please provide a comprehensive answer based solely on the information provided in the context
94 ANSWER:
95 """
96     return prompt
97
98 # Initialize ExLlamaV2 model
99 def initialize_model(model_path):
```

```
100     print(f"Initializing ExLlamaV2 model from: {model_path}")
101
102     config = ExLlamaV2Config()
103     config.model_dir = model_path
104     config.prepare()
105
106     model = ExLlamaV2(config)
107     model.load()
108
109     tokenizer = ExLlamaV2Tokenizer(config)
110     cache = ExLlamaV2Cache(model)
111
112     return model, cache, tokenizer
113
114 # Generate response using ExLlamaV2
115 def generate_response(model, cache, tokenizer, prompt, max_new_tokens=512):
116     generator = ExLlamaV2StreamingGenerator(model, cache, tokenizer)
117
118     # Settings
119     settings = ExLlamaV2Sampler.Settings()
120     settings.temperature = 0.7
121     settings.top_k = 50
122     settings.top_p = 0.9
123     settings.top_a = 0.0
124     settings.token_repetition_penalty = 1.05
125     settings.disallow_tokens(tokenizer, [tokenizer.eos_token_id])
126
127     input_ids = tokenizer.encode(prompt)
128     prompt_tokens = input_ids.shape[-1]
129
130     # Warmup
131     generator.warmup()
132
133     # Begin stream
134     time_begin_prompt = time.time()
135
136     print("\n" + prompt, end="")
137     sys.stdout.flush()
138
139     generator.set_stop_conditions([])
140     generator.begin_stream(input_ids, settings)
141
142     # Streaming loop
143     time_begin_stream = time.time()
144     generated_tokens = 0
145
146     response_text = ""
147
148     while True:
149         chunk, eos, _ = generator.stream()
150         generated_tokens += 1
151         print(chunk, end="")
152         sys.stdout.flush()
153         response_text += chunk
154         if eos or generated_tokens == max_new_tokens:
155             break
156
157     time_end = time.time()
```

```
158
159     time_prompt = time_begin_stream - time_begin_prompt
160     time_tokens = time_end - time_begin_stream
161
162     print()
163     print()
164     print(f"Prompt processed in {time_prompt:.2f} seconds, {prompt_tokens} tokens, {prompt_t}")
165     print(f"Response generated in {time_tokens:.2f} seconds, {generated_tokens} tokens, {gen")
166
167     return response_text
168
169 def main():
170     # Initialize the embedding model (using a free model from sentence-transformers)
171     print("Loading embedding model...")
172     embedding_model = SentenceTransformer('paraphrase-MiniLM-L6-v2')
173
174     # Extract text from PDF
175     pdf_text = extract_text_from_pdf(args.pdf_path)
176
177     # Split text into chunks
178     chunks = split_text_into_chunks(pdf_text, args.chunk_size, args.chunk_overlap)
179
180     # Create embeddings
181     embeddings = create_embeddings(chunks, embedding_model)
182
183     # Build FAISS index
184     index = build_faiss_index(embeddings)
185
186     # Retrieve relevant chunks
187     relevant_chunks = retrieve_relevant_chunks(args.question, chunks, embedding_model, index)
188
189     # Initialize ExLlamaV2 model
190     model, cache, tokenizer = initialize_model(args.model_path)
191
192     # Format prompt with context
193     prompt = format_prompt_with_context(args.question, relevant_chunks)
194
195     # Generate response
196     generate_response(model, cache, tokenizer, prompt)
197
```

1 /content/exllamav2/Mistral-7B-instruct-ex12