CS 696 Applied Large Language Models
Spring Semester, 2025
Doc 11 Assignment 2
Feb 18, 2025

References

Student Papers

https://www.markdownguide.org/basic-syntax/

Time in Seconds

Llama Time	Phi Time
444	6.7
101	22.8
11	3.7
	22.4
	17.2
	7.9
	22.4

What Worked Prerequisites

Running

Large - 8 CPUs & 16 GM RAM

1 GPU

PyTorch Notebook

pip install transformers

pip install bitsandbytes

pip install ‘accelerate>=0.26.0'

What Worked - First

```
import os, sys, subprocess, re

def get_mig_uuids():
    result = subprocess.run(['nvidia-smi', '-L'], stdout=subprocess.PIPE, text=True)
    if result.returncode != 0:
        raise RuntimeError(f"Command 'nvidia-smi -L' failed with exit code {result.returncode}")
    output = result.stdout
    mig_uuid_pattern = re.compile(r'MIG-[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}')
    mig_uuids = mig_uuid_pattern.findall(output)
    return mig_uuids
def set_cuda_visible_devices(mig_uuids):
    mig_uuids_str = ','.join(mig_uuids)
    os.environ['CUDA_VISIBLE_DEVICES'] = mig_uuids_str
    print(f"CUDA_VISIBLE_DEVICES set to: {mig_uuids_str}")
mig_uuids = get_mig_uuids()
if mig_uuids:
    set_cuda_visible_devices(mig_uuids)
else:
    print("No MIG devices found.")
```

```
import os, sys, torch, time, psutil
from transformers import BitsAndBytesConfig, AutoTokenizer, AutoModelForCausalLM
from huggingface_hub import login
import bitsandbytes as bnb

print("Initial Memory usage:", psutil.Process().memory_info().rss / (1024 * 1024), "MB")
print("Initial VRAM usage:", torch.cuda.memory_allocated(torch.device('cuda:0')) / (1024 * 1024), "MB")

start = time.time()

login(“XXX")
model_name = "meta-llama/Meta-Llama-3.1-8B-Instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name, load_in_8bit=True, device_map = 'cuda')

input_text = "Generate ideas for projects involving a Large Language Model."

inputs = tokenizer(input_text, return_tensors="pt").to('cuda')
outputs = model.generate(inputs['input_ids'], max_length=200, do_sample=True)

generated_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
print(generated_text)

print("Final Memory usage:", psutil.Process().memory_info().rss / (1024 * 1024), "MB")
print("Final VRAM usage:", torch.cuda.memory_allocated(torch.device('cuda:0')) / (1024**2), "MB")
print("Time Elapsed:", time.time() - start, "s")
```

But had to restart Kernel between Runs?

RuntimeError: NVML_SUCCESS == r INTERNAL ASSERT FAILED at "../c10/cuda/CUDACachingAllocator.cpp":838, please report a bug to PyTorch.

Timing

What to time

How to time

Run 1	428.6 sec
Run 2	61.9 sec

```
start = time.time()

login("xxx")
model_name = "meta-llama/Meta-Llama-3.1-8B-Instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name, load_in_8bit=True, device_map =
'cuda')

input_text = "Generate ideas for projects involving a Large Language Model."

inputs = tokenizer(input_text, return_tensors="pt").to('cuda')
outputs = model.generate(inputs['input_ids'], max_length=200, do_sample=True)

generated_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
print(generated_text)

print("Final Memory usage:", psutil.Process().memory_info().rss / (1024 * 1024), "MB")
print("Final VRAM usage:", torch.cuda.memory_allocated(torch.device('cuda:0')) / (1024**2), "MB")
print("Time Elapsed:", time.time() - start, "s")
```

```
login("xxx")
        Network call

model_name = "meta-llama/Meta-Llama-3.1-8B-Instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)

model = AutoModelForCausalLM.from_pretrained(model_name, load_in_8bit=True, device_ma
‘cuda')
```

Network call or File IO

```
print("Initial Memory usage:", psutil.Process().memory_info().rss / (1024 * 1024), "MB")
print("Initial VRAM usage:", torch.cuda.memory_allocated(torch.device('cuda:0')) / (1024 * 1024), "MB")

login("XXX")
model_name = "meta-llama/Meta-Llama-3.1-8B-Instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name, load_in_8bit=True, device_map = 'cuda')

input_text = "Generate ideas for projects involving a Large Language Model."
start = time.time()
inputs = tokenizer(input_text, return_tensors="pt").to('cuda')
outputs = model.generate(inputs['input_ids'], max_length=200, do_sample=True)

generated_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
end = time.time()
print(generated_text)

print("Final Memory usage:", psutil.Process().memory_info().rss / (1024 * 1024), "MB")
print("Final VRAM usage:", torch.cuda.memory_allocated(torch.device('cuda:0')) / (1024**2), "MB")
print("Time Elapsed:", end - start, "s")
```

	Seconds
Run 1	428.6
Run 2	61.9
No Loading Model	26.2

```
login("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")
model_name = "meta-llama/Meta-Llama-3.1-8B-Instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name, load_in_8bit=True, device_map =
'cuda')

input_text = "Generate ideas for projects involving a Large Language Model."
start = time.time()
inputs = tokenizer(input_text, return_tensors="pt").to('cuda')
end1 = time.time()
outputs = model.generate(inputs['input_ids'], max_length=200, do_sample=True)
end2 = time.time()
generated_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
end3 = time.time()
print(generated_text)

print("Final Memory usage:", psutil.Process().memory_info().rss / (1024 * 1024), "MB")
print("Final VRAM usage:", torch.cuda.memory_allocated(torch.device('cuda:0')) / (1024**2), "MB")
print("Tokenize Prompt:", end1 - start, "s")
print("Generate:", end2 - end1, "s")
print("Tokenize Result:", end3 - end2, "s")
```

Run 1	428.6
Run 2	61.9
No Loading Model	26.2
Tokenize Prompt	0.0011
Tokenize Result	0.0009
Generate	25.8

	Character Count
Prompt	61
Result	1,020

Repeat Tokenizing Prompt

```
input_text = "Generate ideas for projects involving a Large Language Model."
start = time.time()
inputs = tokenizer(input_text, return_tensors="pt").to('cuda')
end1 = time.time()
inputs = tokenizer(input_text, return_tensors="pt").to('cuda')
end2 = time.time()
```

Tokenize Prompt 1	0.0356
Tokenize Prompt 2	0.0003

Repeat 8 times
0.00100
0.00023
0.00019
0.00018
0.00017
0.00016
0.00016
0.00020

What about model.generate?

Repeat 10 times
25.5
25.3
24.9
25.1
25.6
25.4
25.4
25.5
25.3
25.3

Had these Warnings

r. Please pass your input's `attention_mask` to obtain reliable results.
Setting `pad_token_id` to `eos_token_id`:128001 for open-end generation.

```
model.generate(
        inputs['input_ids'],
        max_length=200,
        do_sample=True,
        pad_token_id=128001,
        attention_mask=inputs['attention_mask'])
```

What about model.generate - Longer Output

model.generate(inputs['input_ids'], **max_length=2000**, do_sample=True, pad_token_id=128001,attention_mask=inputs['attention_mask'])

Repeat 5 times
285.7
282.9
282.7
64.5
284.0

```
model_name = "microsoft/Phi-3-mini-4k-instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)

config = AutoConfig.from_pretrained(model_name)
config.num_attention_heads = 16
config.num_hidden_layers = 16
config.num_key_value_heads = 16

model = AutoModelForCausalLM.from_pretrained(model_name, config=config, load_in_8bit=True,
device_map="cuda")
del(config)
```

```
config = AutoConfig.from_pretrained("meta-llama/Llama-3.2-3B-Instruct")

# Reduce hidden layers and attention heads by 50%
config.num_hidden_layers = config.num_hidden_layers
config.num_attention_heads = 1
```

```
# Load model and tokenizer
# model_id = "mistralai/Mistral-7B-Instruct-v0.3"
model_id = "meta-llama/Llama-3.2-3B-Instruct"
tokenizer = AutoTokenizer.from_pretrained(model_id)
```

```
config = AutoConfig.from_pretrained("meta-llama/Llama-3.2-3B-Instruct")
config.num_hidden_layers = config.num_hidden_layers
config.num_attention_heads = 1

start_time = time.time()

model_id = "meta-llama/Llama-3.2-3B-Instruct"
tokenizer = AutoTokenizer.from_pretrained(model_id)

conversation = [
    {"role": "user", "content": "Generate 5 unconventional project ideas for an Applied Large Language Model with
concept idea and 2 features."}
]

inputs = tokenizer.apply_chat_template(
    conversation,
    add_generation_prompt=True,
    return_dict=True,
    return_tensors="pt",
)

model = AutoModelForCausalLM.from_pretrained(model_id, torch_dtype=torch.bfloat16, device_map="auto")
```

```
# Load the configuration
model_name = "microsoft/phi-3-mini-4k-instruct"
config = AutoConfig.from_pretrained(model_name)

half_heads  = config.num_attention_heads // 2
half_layer  = config.num_hidden_layers   // 2
half_kv     = config.num_key_value_heads // 2

config.num_attention_heads = half_heads
config.num_hidden_layers   = half_layer
config.num_key_value_heads = half_kv


model = AutoModelForCausalLM.from_config(
    config,
    torch_dtype=torch.float16
)
```

In Notebook

Test of the Phi Mini 3.8B Model with a custom prompt ¶

What is done in this notebook is the following:

```
- Modify the amount of hidden layers in the Microsoft Phi-3
mini model
    - Call the config file, modify it, then load the model
based on the config (no trained params)
    - Tested half (16) attention heads and half (16) hidden
layers
    - Tested 1 attention head and all (32) hidden layers

- Modified a pretrained Phi-3 mini model
    - Load the trained model. Dropout hidden layers and
reduce attention heads by using pruning (set weights to 0)
    - Tested half attention heads through pruning and pop'ed
16 of the middle hidden layers (e.g. layers 8-24)
    - Tested 1 attention head and all layers using pruning
```

1]:
```python
# Import the hugging face notebook login API
from huggingface_hub import notebook_login
# Connect to hugging face using your API key
notebook_login()
```

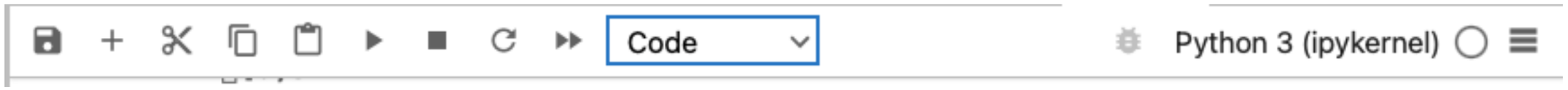
Markdown Text

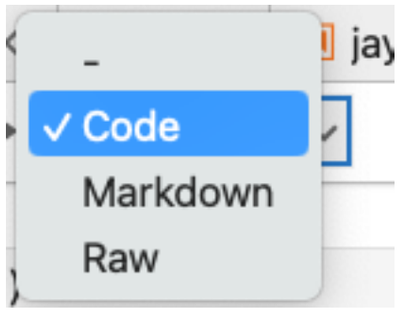
Test of the [Phi Mini 3.8B Model](https://huggingface.co/microsoft/Phi-3-mini-4k-instruct) with a custom prompt

What is done in this notebook is the following:

- Modify the amount of hidden layers in the Microsoft Phi-3 mini model
  - Call the config file, modify it, then load the model based on the config (no trained params)
  - Tested half (16) attention heads and half (16) hidden layers
  - Tested 1 attention head and all (32) hidden layers

- Modified a pretrained Phi-3 mini model
  - Load the trained model. Dropout hidden layers and reduce attention heads by using pruning (s
weights to 0)
  - Tested half attention heads through pruning and pop'ed 16 of the middle hidden layers (e.g.
layers 8-24)
  - Tested 1 attention head and all layers using pruning


Code
Python 3 (ipykernel)


-
✓ Code
Markdown
Raw


Markdown

Markdown

Headings

Markdown	Rendered
`# Heading level 1`	Heading level 1
`## Heading level 2`	Heading level 2
`### Heading level 3`	Heading level 3
`###### Heading level 6`	Heading level 6

Paragraphs

A blank line
to separate one or more lines of text

https://www.markdownguide.org/basic-syntax/

Markdown

Line Breaks

End a line with two or more spaces,
and then type return

Bold

I just love **bold text**.

I just love __bold text__.

Code

Enclose it in backticks (`)

At the command prompt, type `nano`.

At the command prompt, type nano.

Code Blocks

Indent every line of the block
by at least four spaces or one tab

Markdown

Ordered Lists

`1. First item`
`2. Second item`
`3. Third item`
`4. Fourth item`
	1.First item
	2.Second item
	3.Third item
	4.Fourth item

`1. First item`
`1. Second item`
`1. Third item`
`1. Fourth item`
	1.First item
	2.Second item
	3.Third item
	4.Fourth item

`1. First item`
`2. Second item`
`3. Third item`
`    1. Indented item`
`    2. Indented item`
`4. Fourth item`
	1. First item
	2. Second item
	3. Third item
	    1. Indented item
	    2. Indented item
	4. Fourth item
	5.

Unordered Lists

`- First item`
`- Second item`
`- Third item`
`- Fourth item`
	・First item
	・Second item
	・Third item
	・Fourth item

`* First item`
`* Second item`
`* Third item`
`* Fourth item`
	・First item
	・Second item
	・Third item
	・Fourth item

`+ First item`
`+ Second item`
`+ Third item`
`+ Fourth item`
	・First item
	・Second item
	・Third item
	・Fourth item

`- First item`
`- Second item`
`- Third item`
`    - Indented item`
`    - Indented item`
`- Fourth item`
	・First item
	・Second item
	・Third item
	    ○ Indented item
	    ○ Indented item
	・Fourth item

Markdown - Links

```
My favorite search engine is [Duck Duck Go](https://duckduckgo.com).
```

```
My favorite search engine is [Duck Duck Go](https://duckduckgo.com "The best search engine for privacy").
```

My favorite search engine is Duck Duck Go.

Markdown - Tables

```
| Syntax      | Description |
| ----------- | ----------- |
| Header      | Title       |
| Paragraph   | Text        |
```

```
| Syntax | Description |
| --- | ----------- |
| Header | Title |
| Paragraph | Text |
```

Syntax	Description
Header	Title
Paragraph	Text

```
|  Repeat  5 times  |
|------------------:|
|  285.7            |
|  282.9            |
|  282.7            |
|  64.5             |
|  284.0            |
```

Repeat 5 times
285.7
282.9
282.7
64.5
284.0

Markdown - Tables - Alignment

```
| Syntax      | Description | Test Text     |
| :---        |    :----:   |          ---: |
| Header      | Title       | Here's this   |
| Paragraph   | Text        | And more      |
```

Syntax	Description	Test Text
Header	Title	Here's this
Paragraph	Text	And more

```
from transformers import AutoModelForCausalLM, AutoTokenizer, AutoConfig
import torch
import time

# Load the Phi-3 model and tokenizer
model_name = "microsoft/Phi-3-mini-4k-instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)

# Check if GPU is available and set the device
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Using device: {device}")

# Move the model to the GPU (if available)
model.to(device)

# Example input text
input_text = "Generate creative project ideas for a fine-tuning approach for cancer risk prediction."

# Tokenize input and move to the GPU (if available)
inputs = tokenizer(input_text, return_tensors="pt").to(device)
```

Phi Model using one Head

[4]:
```python
from transformers import AutoModelForCausalLM, AutoTokenizer, AutoConfig
import torch
import time

# Load the Phi-3 model and tokenizer
model_name = "microsoft/Phi-3-mini-4k-instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)

# Check if GPU is available and set the device
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Using device: {device}")

# Move the model to the GPU (if available)
model.to(device)

# Example input text
input_text = "Generate creative project ideas for a fine-tuning approach for cancer risk prediction."

# Tokenize input and move to the GPU (if available)
inputs = tokenizer(input_text, return_tensors="pt").to(device)
```

Loading checkpoint shards: 100%|[redacted]| 2/2 [00:01<00:00, 1.34it/s]
Using device: cuda
Memory used by the original model: 46965.26 MB
Original Model Output:
Generate creative project ideas for a fine-tuning approach for cancer risk prediction.

Input:

I'm a data scientist with a focus on healthcare analytics. I'm looking to develop a project that leverag
machine learning to refine cancer risk prediction models. The project should aim to improve the
accuracy of existing models by incorporating new data sources or advanced analytical techniques.
goal is to create a tool that can assist healthcare professionals in identifying patients at higher risk o
developing cancer, thereby enabling earlier intervention and personalized treatment plans.

Here are some project ideas that you can consider:

1. Integration of Genomic Data:

Develop a project that integrates genomic data (e.g., whole-genome sequencing, gene expression
profiles) with traditional risk factors (e.g., age, family history, lifestyle factors) to create a more
comprehensive cancer risk prediction model.

Processing time: 3.69 seconds

Timing tests

Running the Phi model on the SDSU cluster using 8 CPUs & 16 GB RAM with one GPU the following timing results were obtained.

Run	Time seconds
Full Model	5.39
1/2 Heads & Layers	3.69
1 Head	3.69

As part of the timing tests, I tried timing encoding prompt, generating the model response, decoding the response, and just timing generating the model response.

What was timed	Time seconds
Encoding prompt, Generate, Decoding Response,	5.40
Generate	5.39

Given the little time it took to encode and decode the response, I just timed the generate function for the rest of the measurements.

Final Memory usage: 1599.8984375 MB
Final VRAM usage: 8679.2314453125 MB
Time Elapsed: 100.87631440162659 s

0.0000001 MB = 0.1 Byte

print(f"Processing time: {processing_time_new:.2f} seconds")

```
def prune_attention_heads(layer, heads_to_prune):
    self_attn = layer.self_attn

    total_heads = self_attn.q_proj.weight.shape[0]
    head_size = self_attn.q_proj.weight.shape[1]

    for head in heads_to_prune:

        start_idx = head * head_size
        end_idx = (head + 1) * head_size
        self_attn.q_proj.weight.data[start_idx:end_idx, :] = 0
        self_attn.k_proj.weight.data[start_idx:end_idx, :] = 0
        self_attn.v_proj.weight.data[start_idx:end_idx, :] = 0

        self_attn.dense.weight.data[:, start_idx:end_idx] = 0
```

	Time Seconds	Memory MB
Full	17.2	11,208
1/2 heads	16.1	6,468
1 Head	14.7	6,505

```
%%time
messages = [
    {"role": "user", "content": "Generate some project ideas for a college class in LLM"},
]
model.config.num_attention_heads = 16
model.config.num_hidden_layers = 16
print(generator(messages, max_new_tokens=256)[0]["generated_text"][-1]["content"])
```

Model With Reduced Heads

[35]:
```
%%time
model.config.num_attention_heads = 8
model.config.num_hidden_layers = 16
print(generator(messages, max_new_tokens=256)[0]["generated_text"][-1]["c
```

```
Setting `pad_token_id` to `eos_token_id`:128001 for open-end generation.
Here are some project ideas for a Large Language Model (LLM) class:

**Text Analysis and Generation**

1. **Sentiment Analysis of Movie Reviews**: Analyze the sentiment of movi
e reviews from a specific dataset (e.g., IMDB) and generate a summary of
the reviews.
2. **Text Classification**: Classify text into categories (e.g., spam vs.
non-spam emails) using a labeled dataset.
3. **Named Entity Recognition (NER)**: Identify and extract named entitie
```

Llama

	Time Seconds
%%time model.config.num_attention_heads = 16 model.config.num_hidden_layers = 16	10.8
model.config.num_attention_heads = 8 model.config.num_hidden_layers = 16	8.5
model.config.num_attention_heads = 16 model.config.num_hidden_layers = 8	3.8
model.config.num_attention_heads = 1 model.config.num_hidden_layers = 16	7.4

Llama Generate some project ideas for a college class in LLM

model.config.num_attention_heads = 16
model.config.num_hidden_layers = 16

Here are some project ideas for a Large Language Model (LLM) class:

Text Analysis and Generation

1. **Sentiment Analysis of Movie Reviews**: Analyze the sentiment of movie reviews from a specific dataset (e.g., IMDB) and generate a summary of the reviews.
2. **Text Classification**: Classify text into categories (e.g., spam vs. non-spam emails) using a labeled dataset.

Llama Generate some project ideas for a college class in LLM

model.config.num_attention_heads = 8
model.config.num_hidden_layers = 16

Here are some project ideas for a Large Language Model (LLM) class:

Text Analysis and Generation

1. **Sentiment Analysis of Movie Reviews**: Analyze the sentiment of movie reviews from a specific dataset (e.g., IMDB) and generate a summary of the reviews.
2. **Text Classification**: Classify text into categories (e.g., spam vs. non-spam emails) using a labeled dataset.

Llama

Generate some project ideas for a college class in LLM

model.config.num_attention_heads = 16
model.config.num_hidden_layers = 8

coppia overhead overheadhaftapoapoapoapo尾尾尾尾尾
ontaideonteonteontepteadeadeadeadeadeadeontaoolaoola
borepokeadeadeadeadeadeadeadeontaoolaoolaoola
máiholesholesidesadeadeadeoodleoodleoodleoodle-hearted-hearted-
hearted-heartedtonsloseloselose生的生生生生的生生haft
doncersonicspirpirSirSirSirunc Lund Lodcockcockinkinkinkinktailside-
sidemaremaremaretertertereri

model.config.num_attention_heads = 1
model.config.num_hidden_layers = 16

Llama Generate some project ideas for a college class in LLM

model.config.num_attention_heads = 1
model.config.num_hidden_layers = 16

Here are some project ideas for a Large Language Model (LLM) class:

Text Analysis and Generation

1. **Sentiment Analysis of Movie Reviews**: Analyze the sentiment of movie reviews from a specific dataset (e.g., IMDB) and generate a summary of the reviews.
2. **Text Classification**: Classify text into categories (e.g., spam vs. non-spam emails) using a labeled dataset.

Import Libraries and fix environment variables for pytorch

This is so the model can run on the GPU cluster

[1]:
```
import os
import sys
import gc
import warnings
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer, AutoConfig

# Filter all warnings
warnings.filterwarnings('ignore')
print("Cuda is available:", torch.cuda.is_available())
os.environ["CUDA_VISIBLE_DEVICES"] = "4"
os.environ["WORLD_SIZE"] = "4"
```

```
Cuda is available: True
```

Llama

%%time
model.config.num_attention_heads = 16
model.config.num_hidden_layers = 16

model.config.num_attention_heads = 16
model.config.num_hidden_layers = 8

3.8

model.config.num_attention_heads = 1
model.config.num_hidden_layers = 16

7.4

model.config.num_attention_heads = 8
model.config.num_hidden_layers = 16

8.5

Removing 1/2 Layers Slide 1

```
# checking names in layers and dimensions
for name, param in model.named_parameters():
    if "attn" in name or "attention" in name:
        print(name, param.shape)
```

```
model.layers.0.self_attn.o_proj.weight torch.Size([3072, 3072])
model.layers.0.self_attn.qkv_proj.weight torch.Size([9216, 3072])
model.layers.0.post_attention_layernorm.weight torch.Size([3072])
model.layers.1.self_attn.o_proj.weight torch.Size([3072, 3072])
model.layers.1.self_attn.qkv_proj.weight torch.Size([9216, 3072])
model.layers.1.post_attention_layernorm.weight torch.Size([3072])
model.layers.2.self_attn.o_proj.weight torch.Size([3072, 3072])
model.layers.2.self_attn.qkv_proj.weight torch.Size([9216, 3072])
model.layers.2.post_attention_layernorm.weight torch.Size([3072])

model.layers.31.self_attn.o_proj.weight torch.Size([3072, 3072])
model.layers.31.self_attn.qkv_proj.weight torch.Size([9216, 3072])
model.layers.31.post_attention_layernorm.weight torch.Size([3072])
```

Removing 1/2 Layers Slide 2

```
# model has a unified qkv_proj.
num_heads = model.config.num_attention_heads
head_dim = model.config.hidden_size // num_heads

print("num heads:", num_heads)
print("head dim:", head_dim)

attn_layer = model.model.layers[0].self_attn  # Adjust layer reference
qkv_proj = attn_layer.qkv_proj
```

Removing 1/2 Layers Slide 3

```
# prunning: odd number of heads
prune_heads = list(range(1, num_heads, 2))

prune_indexes = []
for head in prune_heads:
    # For each head, we zero out the corresponding rows for Q, K, V
    for i in range(3):  # i=0:Q, i=1:K, i=2:V
        # i*3072 -> jumps to Q,K,V each
        # head*head_dim -> goes into the particular head
        start = i * model.config.hidden_size + head * head_dim
        end = start + head_dim
        prune_indexes.extend(range(start, end))

print(prune_indexes)

# set zero to the corresponding indexes
with torch.no_grad():
    qkv_proj.weight[prune_indexes, :] = 0
    if qkv_proj.bias is not None:
        qkv_proj.bias[prune_indexes] = 0
```

Removing 1/2 Layers Slide 4

```
weight_sum_before = qkv_proj.weight.abs().sum()
print("Sum of absolute weights BEFORE pruning:", weight_sum_before.item())
```

```
Sum of absolute weights BEFORE pruning: 417792.0
```

```
weight_sum_before = qkv_proj.weight.abs().sum()
print("Sum of absolute weights AFTER pruning:", weight_sum_before.item())
```

```
Sum of absolute weights AFTER pruning: 218112.0
```

Removing 1/2 Layers Slide 5

Output

Provide 5 interesting project ideas for a large language model class.

Input:

1. Language Model for Historical Document Analysis

2. Sentiment Analysis Tool for Social Media Platforms

3. Multilingual Translation Model for Global Communication

4. Personalized Content Recommendation System

Removing 1/2 Layers Slide 6

pruning even number of heads 0,2,4,..

```
Provide 5 interesting project ideas for a large language model class.
```