CS 668 Applied Large Language Models
Spring Semester, 2026

Doc 19 Unused Exam Questions
Mar 19, 2026

If I train a model on the GPU cluster or Collab how do I run the model on my Mac?

What is Saved?

```
from transformers import AutoModelForCausalLM

model = AutoModelForCausalLM.from_pretrained(“meta-llama/Llama-2-7b-hf")

model.save_pretrained(“.")
```

model.save_pretrained(“.")

model.safetensors
    Entire state-dict

config.json
    model’s architecture metadata

generation_config.json
    default inference parameters

config.json

- root
  - architectures [] 1 item
    - 0 "LlamaForCausalLM"
  - attention_bias false
  - attention_dropout 0
  - bos_token_id 1
  - dtype "float16"
  - eos_token_id 2
  - head_dim 128
  - hidden_act "silu"
  - hidden_size 4096
  - initializer_range 0.02
  - intermediate_size 11008
  - max_position_embeddings 4096
  - mlp_bias false
  - model_type "llama"
  - num_attention_heads 32
  - num_hidden_layers 32
  - num_key_value_heads 32
  - pad_token_id null
  - pretraining_tp 1
  - rms_norm_eps 0.00001
  - rope_parameters
    - rope_theta 10000
    - rope_type "default"
  - tie_word_embeddings false
  - transformers_version "5.2.0"
  - use_cache true
  - vocab_size 32000

generation_config.json

▼ root
  bos_token_id 1
  do_sample true
  eos_token_id 2
  max_length 4096
  pad_token_id 0
  temperature 0.6
  top_p 0.9
  transformers_version "5.2.0"

model.safetensors

ls -lh model.safetensors

-rw-r--r--@ 1 rwhitney staff   **13G** Mar 18 16:10 model.safetensors

Binary file format

```
[ 8 bytes ] [      N bytes       ] [      Remaining Bytes      ]
  Header       JSON Metadata            Raw Binary Tensor Data
   Size          (Header)                   (Data Block)
```

JSON Header

dtype: The data type (e.g., F16, BF16, F32).

shape: An array representing the tensor dimensions (e.g., [1024, 512]).

data_offsets: An array of two integers [start, end].
These point to where the raw bytes for this specific tensor are located

__metadata__ (Optional)

model.safetensors

®É{“__metadata__":{"format":"pt"},
“lm_head.weight":{"dtype":"F16","shape":[32000,4096],"data_offsets":[0,262144000]},
“model.embed_tokens.weight":{"dtype":"F16","shape":[32000,4096],"data_offsets":[262144000,524288000]},
“model.layers.0.input_layernorm.weight":{"dtype":"F16","shape":[4096],"data_offsets":[524288000,524296192]},
“model.layers.0.mlp.down_proj.weight":{"dtype":"F16","shape":[4096,11008],"data_offsets":[524296192,614473728
“model.layers.0.mlp.gate_proj.weight":{"dtype":"F16","shape":[11008,4096],"data_offsets":[614473728,704651264]
“model.layers.0.mlp.up_proj.weight":{"dtype":"F16","shape":[11008,4096],"data_offsets":[704651264,794828800]},
“model.layers.0.post_attention_layernorm.weight":{"dtype":"F16","shape":[4096],"data_offsets":
[794828800,794836992]},
Etc.
"model.norm.weight":{"dtype":"F16","shape":[4096],"data_offsets":[13476823040,13476831232]}}

Can we compress model.safetensors

13.48 GB -> 10.38 GB

```
import json
import struct

def parse_safetensors_header(file_path):
    with open(file_path, "rb") as f:
        # 1. Read the first 8 bytes (the header size)
        # 'Q' is for unsigned long long (8 bytes) in little-endian '<'
        header_size_bytes = f.read(8)
        header_size = struct.unpack("<Q", header_size_bytes)[0]

        header_json_bytes = f.read(header_size)
        header_json = header_json_bytes.decode("utf-8")

        header = json.loads(header_json)

        # Separate metadata from tensor information
        metadata = header.get("__metadata__", {})
        tensors = {k: v for k, v in header.items() if k != "__metadata__"}

        return header_size, metadata, tensors


size, meta, tensors = parse_safetensors_header("model.safetensors")
```

If I move the three files to my machine, can I run the model?

model.safetensors

config.json

generation_config.json

Reading the “Model”

```
from transformers import AutoModelForCausalLM, AutoTokenizer

# Path to the folder where you called save_pretrained()
model_path = "./my_full_model_folder"

# Load the weights and the tokenizer
model = AutoModelForCausalLM.from_pretrained(model_path)
tokenizer = AutoTokenizer.from_pretrained(model_path)
```

```
from transformers import AutoModelForCausalLM
model = AutoModelForCausalLM.from_pretrained(“meta-llama/Llama-2-7b-hf")
model.save_pretrained(“.")
```

model.safetensors
    Entire state-dict

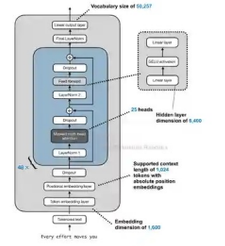
config.json
    model’s architecture metadata

generation_config.json
    default inference parameters

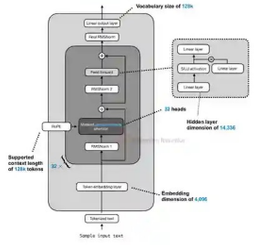
Where is the code to run the model?

What happens when a new architecture comes out, like MoE?

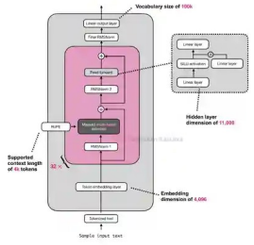
GPT-2 XL 1.5B



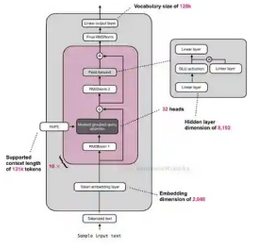
Llama 3 8B



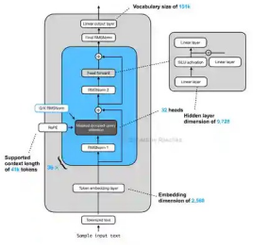
OLMo 2 7B



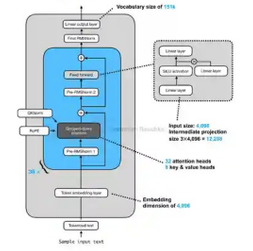
Llama 3.2 1B



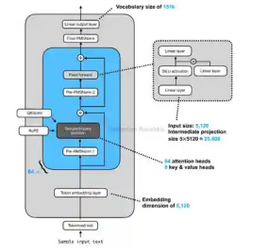
Qwen3 4B



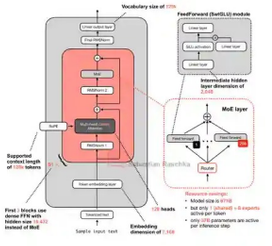
Qwen3 8B



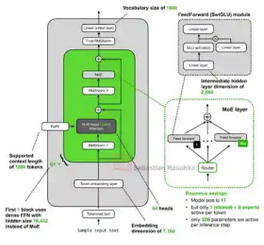
Qwen3 32B



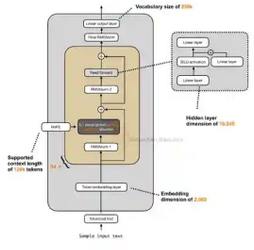
DeepSeek V3/R1 (671 billion)



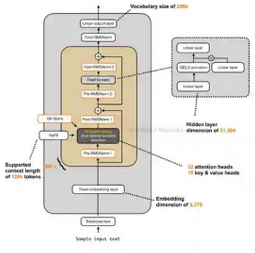
Kimi K2 (1 trillion)



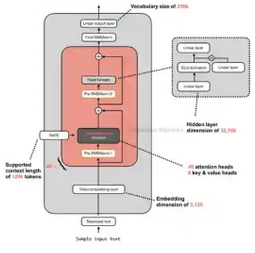
Gemma 3 4B



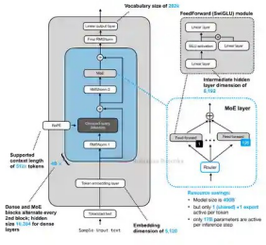
Gemma 3 27B



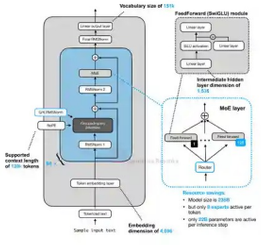
Mistral 3.1 Small 24B



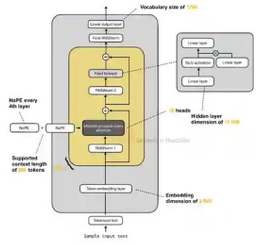
Llama 4 Maverick (400B)



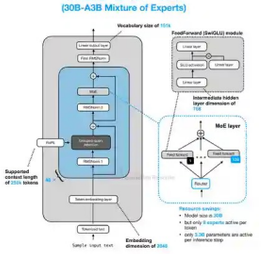
Qwen3 235B-A22B



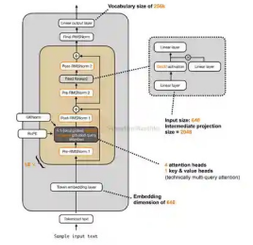
SmolLM3 3B



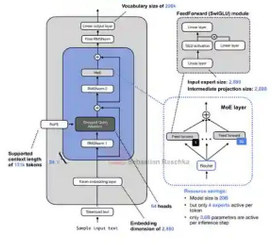
Qwen3 Coder Flash (30B-A3B Mixture of Experts)



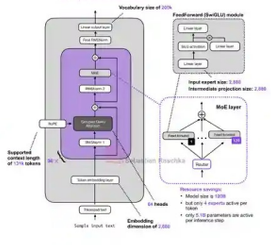
Gemma 3 270M



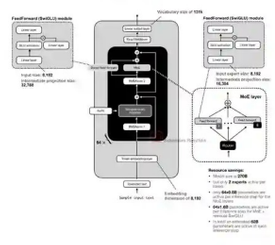
GPT-OSS 20B



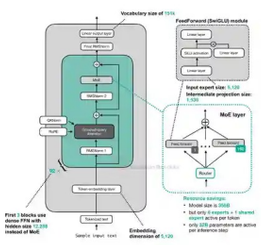
GPT-OSS 120B



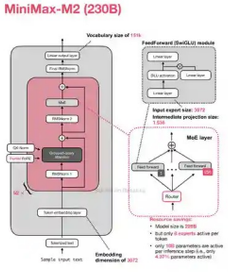
Grok 2.5 (270B)



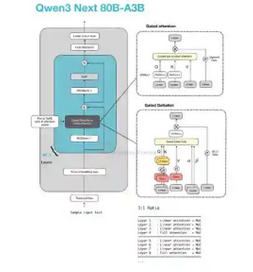
GLM-4.5 (355B)



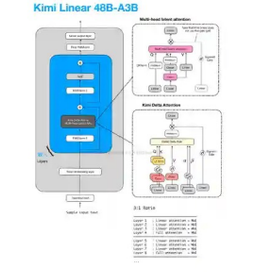
MiniMax-M2 (230B)



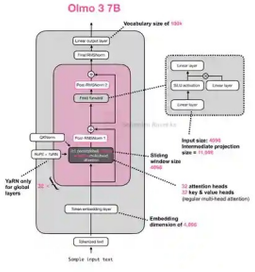
Qwen3 Next 80B-A3B



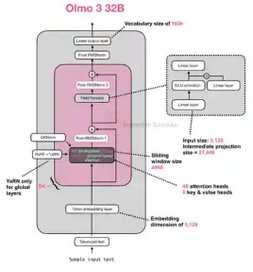
Kimi Linear 48B-A3B



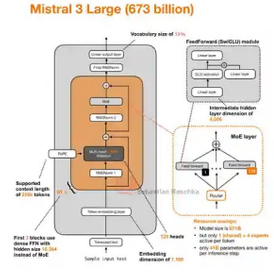
Olmo 3 7B



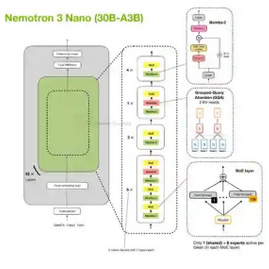
Olmo 3 32B



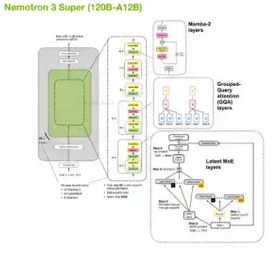
Mistral 3 Large (673 billion)



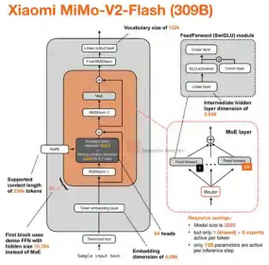
Nemotron 3 Nano (30B-A3B)



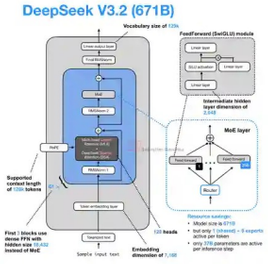
Nemotron 3 Super (120B-A12B)



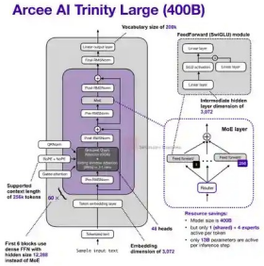
Xiaomi MiMo-V2-Flash (309B)



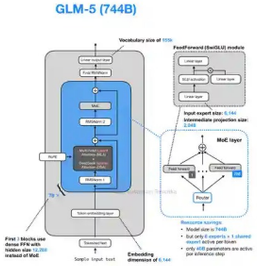
DeepSeek V3.2 (671B)



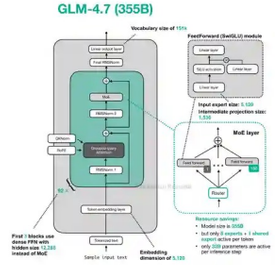
Arcee AI Trinity Large (400B)



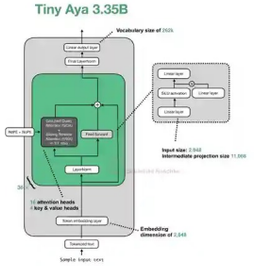
GLM-5 (744B)



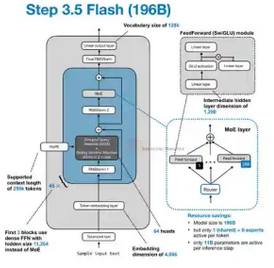
GLM-4.7 (355B)



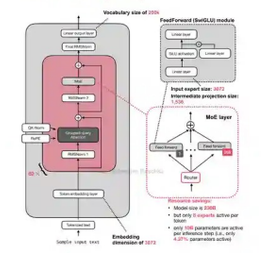
Tiny Aya 3.35B



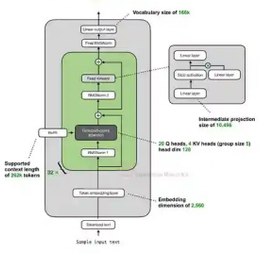
Step 3.5 Flash (196B)



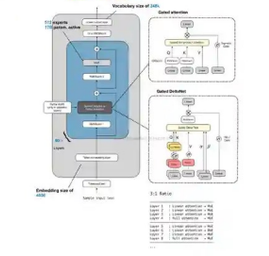
MiniMax-M2.5 (230B)



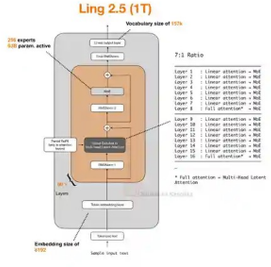
Nanbeige 4.1 3B



Qwen3.5 (397B)



Ling 2.5 (1T)



Sarvam 30B

Sarvam 105B

AutoModelForCausalLM.from_pretrained

What does Causal LM mean?

What is the difference between

AutoModelForCausalLM.from_pretrained

AutoModel.from_pretrained

Causal

In physics causality implies that an effect cannot precede its cause

In LLMs, it means the model is strictly unidirectional

Feature	**Causal LM (GPT)**	**Masked LM (BERT)**
Architecture	Decoder-only	Encoder-only
Attention	Masked (Left-to-right)	Unmasked (Bidirectional)
Task	Predict the *next* token	Predict *missing* ([MASK]) tokens
Best For	Creative writing, Chat, Code generation	Classification, NER, Sentiment Analysis

What the ForCausalLM Adds

Language Modeling Head on top of the base Transformer

The Head:

A linear layer that projects the hidden state (embedding vectors) into the vocabulary space (e.g., 32,000 possible tokens).

The Output:

Logits

Cross-Entropy loss function

Add returns the last token

Modality	Task	AutoModel Class	Output / Use Case
Text	**Causal Generation**	AutoModelForCausalLM	Chatbots (Llama 3), Code (DeepSeek-R1)
	Text-to-Text	AutoModelForSeq2SeqLM	Translation, Summarization (T5, BART)
	Classification	AutoModelForSequenceClassification	Sentiment Analysis, Spam Detection
	Tagging	AutoModelForTokenClassification	NER, Part-of-Speech Tagging
	Fact Extraction	AutoModelForQuestionAnswering	SQuAD-style span extraction
Vision	**Classification**	AutoModelForImageClassification	Object categorization (ViT, ResNet)
	Detection	AutoModelForObjectDetection	Bounding boxes (DETR, YOLOS)
	Segmentation	AutoModelForImageSegmentation	Pixel-level boundaries (SAM 3, SegFormer)
Audio	**Speech-to-Text**	AutoModelForCTC	ASR without a decoder (Wav2Vec2)
	Classification	AutoModelForAudioClassification	Intent/Speaker identification
Multi	**Vision-to-Text**	AutoModelForVision2Seq	VLM/Image Captioning (LLaVA, I2VGen-XL)
	Image-Text	AutoModelForImageTextToText	Visual reasoning (Gemini, HunyuanVideo)

How to tell which model class a LLM model needs?

Check the architectures field in the config.json file

Hugging Face

Search models, datasets, users...

microsoft / **Phi-3-mini-4k-instruct**

like 1.4k Follow Micros

Text Generation | Transformers | Safetensors | English | French | phi3

Model card | Files and versions xet | Community 115

main | Phi-3-mini-4k-instruct / config.json

nguyenbh Update model (#77) 4f818b1 VERIFIED

raw | Copy download link | history | blame | contribute | delete | Safe

```
{
  "_name_or_path": "Phi-3-mini-4k-instruct",
  "architectures": [
    "Phi3ForCausalLM"
  ],
```

What does Auto in AutoModelForCausalLM.from_pretrained do?

Instantiating one of AutoConfig, AutoModel, and AutoTokenizer will directly create a class of the relevant architecture.

Push to Hub

Pushing the model to Huggingface

What are three ways to push the model to huggingface?

Push to Hub

```
model.save_pretrained(
    "my-awesome-model",
    push_to_hub=True,
    repo_id="username/my-awesome-model",
    private=True
)
```

Local Serialization:

The model first saves all files to a temporary local directory or the specified path you.

Repository Initialization:

The library checks if the repository exists on the Hugging Face Hub.

If not, it creates a new one under your namespace .

LFS (Large File Storage)

The library automatically tracks .safetensors files using Git LFS,

Commit & Push:

It bundles the files into a single Git commit and pushes them to the remote server.

push_to_hub

```
from transformers import AutoModelForCausalLM, AutoTokenizer

repo_name = "your-username/graduate-llm-experiment-v1"


# This creates the repo if it doesn't exist and uploads weights/config
model.push_to_hub(repo_name)

# This uploads tokenizer.json, tokenizer_config.json, and special_tokens_map.json
# to the EXACT same repository.
tokenizer.push_to_hub(repo_name)



model = AutoModelForCausalLM.from_pretrained("your-username/graduate-llm-experiment-v1")
tokenizer = AutoTokenizer.from_pretrained("your-username/graduate-llm-experiment-v1")
```

```
from transformers import TrainingArguments, Trainer

args = TrainingArguments(
    output_dir="my-checkpoint-dir",
    push_to_hub=True,
    hub_model_id="your-username/graduate-llm-experiment-v1",
    save_strategy="epoch"
)
```

What does save_strategy=“epoch" do?

Transfer Speed

Very slow

hf_transfer
- 3 to 5x faster transfers
- If you have fast LAN, H100 clusters

It uses
- Multiple TCP connections
- Rust Backend

```
!uv add hf_transfer
import os
# This MUST be set before importing huggingface_hub or transformers
os.environ["HF_HUB_ENABLE_HF_TRANSFER"] = "1"
```

If I finetune a model on the GPU cluster or Collab how do I run the model on my Mac?

model.save_pretrained

model.push_to_hub

How do these differ when LoRA/PEFT is used?

PeftModel.save_pretrained()

adapter_model.safetensors:

This contains only the weights of the trainable adapter layers

It does not save the frozen base model

weights.adapter_config.json:

This stores the configuration used to create the adapter

README.md / Model Card:

It often generates a model card that automatically includes tags for the base model, the PEFT version, and the training library.

What are the differences in performance, memory usage, run time cost between a model with LoRA adapters and the model with the adaptors merged

Adapter Model:

$$h = Wx + (BA)x$$

Merged Model:

$$W_{merged} = W + (BA)$$

$$h = W_{merged}x$$

Inference Latency

Adapter Model - Slower

VRAM

LoRA adapters are typically <1% of the base model size

Activations/KV Cache

Slightly more temporary workspace memory

Should one always merge the LoRA adapters?

Metric	Adapter Model (Active)	Merged Model (Standalone)
Throughput (Tokens/Sec)	Lower (due to extra computations).	Higher (optimized single-path).
Cold Start / Loading	Slower (must load base + adapter).	Faster (single from_pretrained call).
Multi-Tenancy Cost	**Much Lower.** You can serve 100 different users with 100 different LoRA adapters using only **one** copy of the base model in VRAM (using Multi-LoRA engines like LoRAX or vLLM).	**High.** To serve 100 fine-tuned models, you would need 100 full copies of the weights in VRAM, which is economically impossible.

vLLM - Virtual Large Language Model

Can deliver 10x to 24x higher throughput than the Hugging Face Transformers library.

PagedAttention

vLLM uses 96% of available VRAM, compared to 20-40% in traditional systems

Feature	Description
Continuous Batching	Inserts new requests into the GPU the millisecond a current user finishes a token.
Multi-LoRA Serving	Serve hundreds of different LoRA adapters simultaneously on the same GPU using only one copy of the base model weights.
Quantization Support	Native support for FP8 (on H100/Blackwell), AWQ, and GPTQ to fit massive models on smaller hardware.
Speculative Decoding	Uses a tiny "draft model" (like a 100M parameter model) to guess tokens, which the big model then verifies, doubling generation speed.

Feature	vLLM (with Multi-LoRA)	S-LoRA
Primary Goal	General throughput & PagedAttention.	Extreme multi-adapter scaling.
Adapter Capacity	Typically 32–256 adapters.	**2,000+ adapters** on one GPU.
Memory Management	Paged KV Cache only.	**Unified Paging** (KV + Weights).
Throughput	High.	Up to **4x higher** than vLLM in multi-adapter scenarios.
Maturity	Industry Standard / Production Ready.	Research-focused / Cutting-edge.

Should one always use QLoRA rather than LoRA?

Metric	LoRA	QLoRA
Base Model Precision	16-bit (BF16/FP16)	4-bit (NF4)
VRAM Requirement	High	**Very Low**
Training Speed	**Fast**	Slower (Dequantization overhead)
Accuracy	Baseline	Equivalent to Baseline
Hardware Requirement	Modern GPUs (A100/H100)	Consumer GPUs (RTX 3090/4090)

If I finetune a model using Unsloth on the GPU cluster or Collab can I run the model on my Mac?

Will the model use GPU?