Page 1:

1. Understanding Model Parameters:

+ Amodel's parameters are the weights (e.g., matrices, vectors) that the model learns during training,

+ For a model with n parameters, the memory required depends on the precision used to store these parameters.

2. Precision Formats:

+ FP32 (Single Precision): Each parameter uses 4 bytes.

+ FP16 (Half Precision): Each parameter uses 2 bytes,

+ FP16 is commonly used in deep learning inference to reduce memory usage and computational requirements without significant loss in accuracy.

3. Memory Requirement Formula:

?+ Memory required for model parameters = Number of parameters x Bytes per parameter.

4. Application to LLaVA-Video-72B-Qwen2:

?+ This model has 72 billion parameters:

n=72x 10°

?+ In FP16 format, each parameter requires 2 bytes.

+ Total memory required for parameters in FP16:

Memory (FP16) =n x 2 = 72 x 10° x 2bytes

5. Convert to Gigabytes (GB):

+ 1GB = 1024° bytes = 1,073, 741, 824 bytes.

* Converting memory from bytes to GB:

Memory (bytes)

Memory (GB) = "73-741, and

* Substituting the values:

72 x 10° x 2

Memory (GB) = ~~" ~*~ 44GB.

1, 073, 741, 824

6. Addi

nal Memory for Activations:

+ The activations (intermediate computations like attention maps) during inference also require memory.

+ These activations typically add 20-30% of the model size:

¥

?Total VRAM (FP16)

144 GB + (30% x 144 GB) ~ 180 GB.

Page 2:

Final Breakdown:

+ Model Parameters (FP16): 72B x 2 bytes ~ 144 GB.

+ Additional Activations: ~30% of model size > ~180 GB total memory required for inference.

This calculation helps determine the number of GPUs needed, as you must split this memory requirement across GPUs with limited VRAM.

AGP ov

can we reduce it to further down to FP16?

If you're already using FP 16 (half-precision) for inference with LLaVA-Video-72B-Qwen2, the model parameters are already stored in the most commonly used efficient precision format for deep learning. However, there are additional methods to further reduce memory requirements without significantly impacting performance.

Page 3:

1. Use Quantization (Lower than FP16)

Quantization is a technique to reduce the precision of model parameters further while maintaining

accuracy.

a. Int Quantization

Converts parameters from FP16 (16 bits) to Int8 (8 bits), halving memory usage

Int8 parameters require only 1 byte per parameter.

Memory Savings:

72x 10° x lbyte=72GB (parameters only).

Tools like Hugging Face's bitsandbytes or DeepSpeed can handle Int8 quantization during inference.

Example with Hugging Face:

python

```
from transformers import AutoModelForCausalLM

model = AutoModelForCausallM. fron_pretrained(
?Unms~Lab/LLaVA-Video-728-Qwen2",
Voad_in_sbit=True,
device map="auto?
```

b, 4-Bit Quantization (FP4 or Int4)

Reduces precision even further to 4 bits (half of Int8).

Parameters require only 0.5 bytes per parameter.

Memory Savings:

72 x 10° x 0.5 byte = 36GB.

Note: Int4 quantization may impact model accuracy, but it's suitable for certain use cases.

Page 4:

Example:

+ Using bitsandbytes for 4-bit quantization:

python copy code

from transformers import AutoModelForCausalLM

model = AutoModelForCausalM. from_pretrained(

?Unms~Lab/LLaVA-Video-728-Qwen2",

2. Layer Offloading

* Offload less active model layers to CPU or disk while keeping critical layers on the GPU.

+ Tools like DeepSpeed or Hugging Face's device map can offload parts of the model to reduce GPU memory usage.

+ Example:

python copy code

from transformers import AutoModelForCausalLM

model = AutoModelForCausallM. from_pretrained(

?Unms~Lab/LLaVA-Video-728-Qwen2" ,

device_map={""transformer. layers.0-23": "cpu", ?transformer. layers.24~47"

torch_dtype=torch. float16

Page 5:

3. Use Activation Checkpointing

+ Saves memory by recomputing intermediate activations during backpropagation, rather than

storing them.

+ While this increases compute time, it drastically reduces GPU memory usage.

Example with Hugging Fac

python copy code

fron transformers import AutoModeLForCausalLm

model = AutoModelForCausalLM. from_pretrained(

?Unms-Lab/LLaVA-Video-728-Qwen2",

device_map="auto",

torch_dtype=torch. floati6

)eeval()

model. gradient_checkpointing_enable()

4. Distributed Model Parallelism

+ Split the model parameters across multiple GPUs (e.g., using DeepSpeed, Tensor

Parallelism, or Accelerate).

+ This reduces the memory load on a single GPU.

5. Sparse Attention

* Use sparse attention mechanisms to reduce the memory footprint of attention layers.

?+ Many transformer-based models allow replacing dense attention with sparse approximations,

reducing activation memory.

Page 6:

?Summary of Techniques:

Method

FPI6

Inte

Quantization

abit

Quantization

Layer Offloading

?Activation

Checkpointing

Conclusion

16 bits

abits

4 bits

Mixed

16 bits

Memory per Parameter

Parameter

2 bytes

byte

oS byte

Dependent

NYA

Total

Memory Pros

144 6B High accuracy

7268 Halves memory

usage

3668 Drastically

reduces memory

usage

Varies Reduces GPU

memory

requirements

Varies Significant

memory savings

+ For maximum memory reduction, use Int8 or 4-bit quantization.

* Combine with techniques like offloac

+ If extreme memory savings are needed, prioritize 4-bit quantization and distributed inference

setups.

Cons

Already

optimized.

Slight

accuracy

drop.

Potential

accuracy

loss,

Slower

inference.

Increases

compute

time,

1g and model parallelism to handle large models like

LLaVA-Video-72B-Qwen2 on limited hardware.