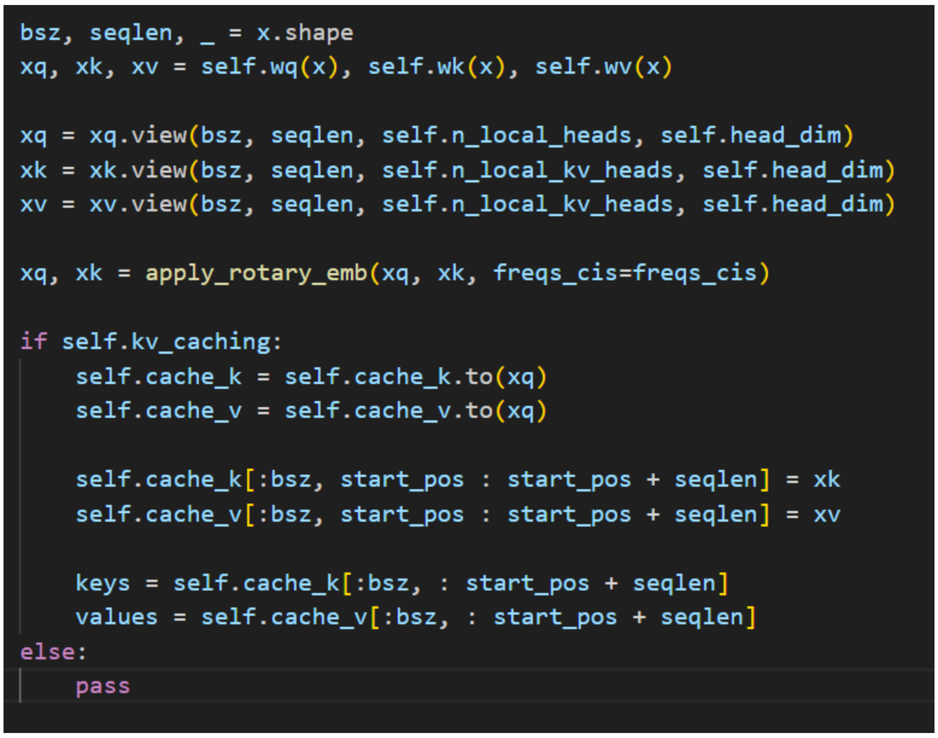
Summary of changes:

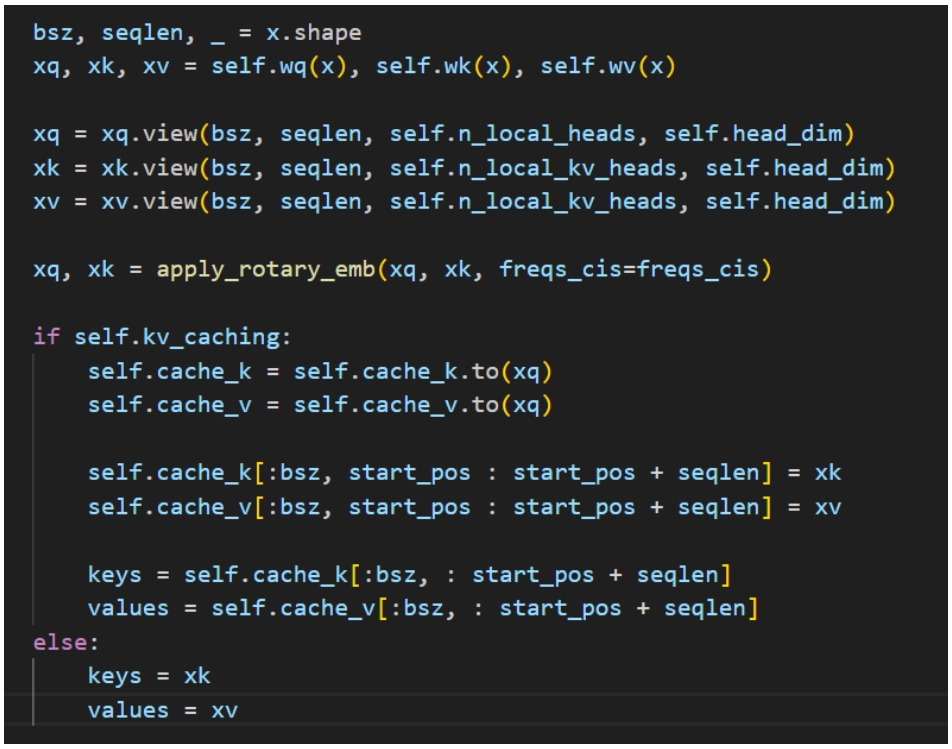
Steps taken to disable KV caching:

1. In model.py modify Attention.forward():
   Before:



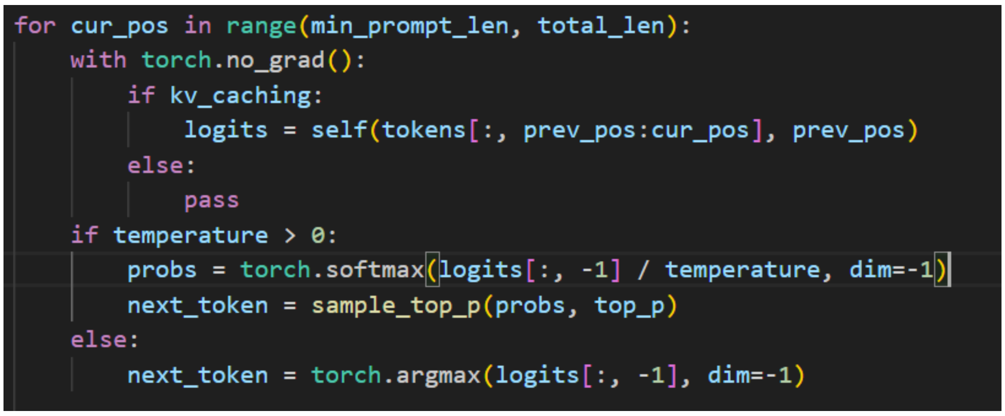
```
bsz, seqlen, _ = x.shape
xq, xk, xv = self.wq(x), self.wk(x), self.wv(x)

xq = xq.view(bsz, seqlen, self.n_local_heads, self.head_dim)
xk = xk.view(bsz, seqlen, self.n_local_kv_heads, self.head_dim)
xv = xv.view(bsz, seqlen, self.n_local_kv_heads, self.head_dim)

xq, xk = apply_rotary_emb(xq, xk, freqs_cis=freqs_cis)

if self.kv_caching:
    self.cache_k = self.cache_k.to(xq)
    self.cache_v = self.cache_v.to(xq)

    self.cache_k[:bsz, start_pos : start_pos + seqlen] = xk
    self.cache_v[:bsz, start_pos : start_pos + seqlen] = xv

    keys = self.cache_k[:bsz, : start_pos + seqlen]
    values = self.cache_v[:bsz, : start_pos + seqlen]
else:
    pass
```

   After:



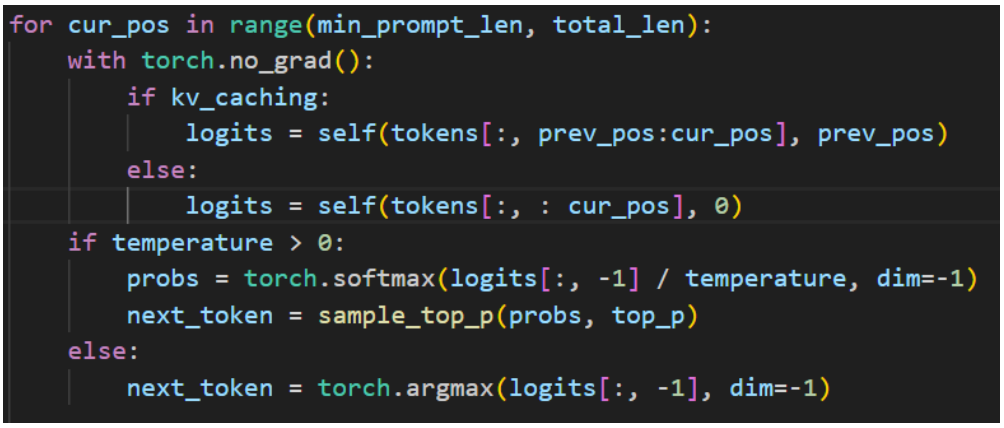
```
bsz, seqlen, _ = x.shape
xq, xk, xv = self.wq(x), self.wk(x), self.wv(x)

xq = xq.view(bsz, seqlen, self.n_local_heads, self.head_dim)
xk = xk.view(bsz, seqlen, self.n_local_kv_heads, self.head_dim)
xv = xv.view(bsz, seqlen, self.n_local_kv_heads, self.head_dim)

xq, xk = apply_rotary_emb(xq, xk, freqs_cis=freqs_cis)

if self.kv_caching:
    self.cache_k = self.cache_k.to(xq)
    self.cache_v = self.cache_v.to(xq)

    self.cache_k[:bsz, start_pos : start_pos + seqlen] = xk
    self.cache_v[:bsz, start_pos : start_pos + seqlen] = xv

    keys = self.cache_k[:bsz, : start_pos + seqlen]
    values = self.cache_v[:bsz, : start_pos + seqlen]
else:
    keys = xk
    values = xv
```

2. In generation.py modify Generation.generate():
   Before:



```python
for cur_pos in range(min_prompt_len, total_len):
    with torch.no_grad():
        if kv_caching:
            logits = self(tokens[:, prev_pos:cur_pos], prev_pos)
        else:
            pass
    if temperature > 0:
        probs = torch.softmax(logits[:, -1] / temperature, dim=-1)
        next_token = sample_top_p(probs, top_p)
    else:
        next_token = torch.argmax(logits[:, -1], dim=-1)
```

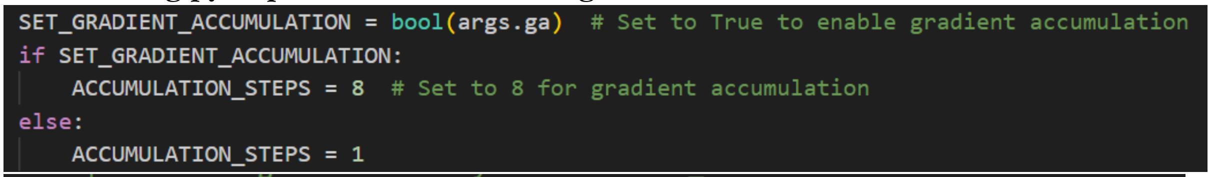
   After:



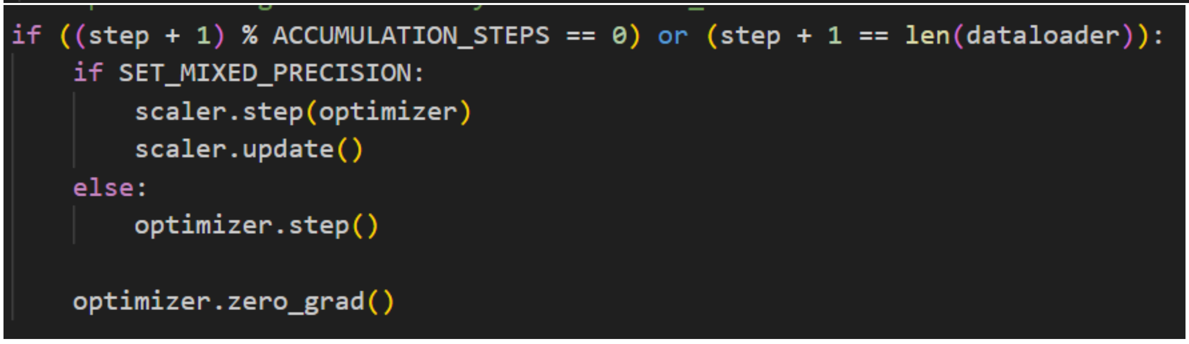
```python
for cur_pos in range(min_prompt_len, total_len):
    with torch.no_grad():
        if kv_caching:
            logits = self(tokens[:, prev_pos:cur_pos], prev_pos)
        else:
            logits = self(tokens[:, : cur_pos], 0)
    if temperature > 0:
        probs = torch.softmax(logits[:, -1] / temperature, dim=-1)
        next_token = sample_top_p(probs, top_p)
    else:
        next_token = torch.argmax(logits[:, -1], dim=-1)
```

Steps taken to implement Gradient Accumulation:

1. In finetuning.py implemented the following code:



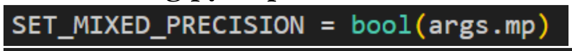
```python
SET_GRADIENT_ACCUMULATION = bool(args.ga)  # Set to True to enable gradient accumulation
if SET_GRADIENT_ACCUMULATION:
    ACCUMULATION_STEPS = 8  # Set to 8 for gradient accumulation
else:
    ACCUMULATION_STEPS = 1
```



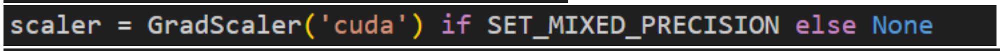
```python
if ((step + 1) % ACCUMULATION_STEPS == 0) or (step + 1 == len(dataloader)):
    if SET_MIXED_PRECISION:
        scaler.step(optimizer)
        scaler.update()
    else:
        optimizer.step()

    optimizer.zero_grad()
```

Steps taken to implement Mixed Precision:

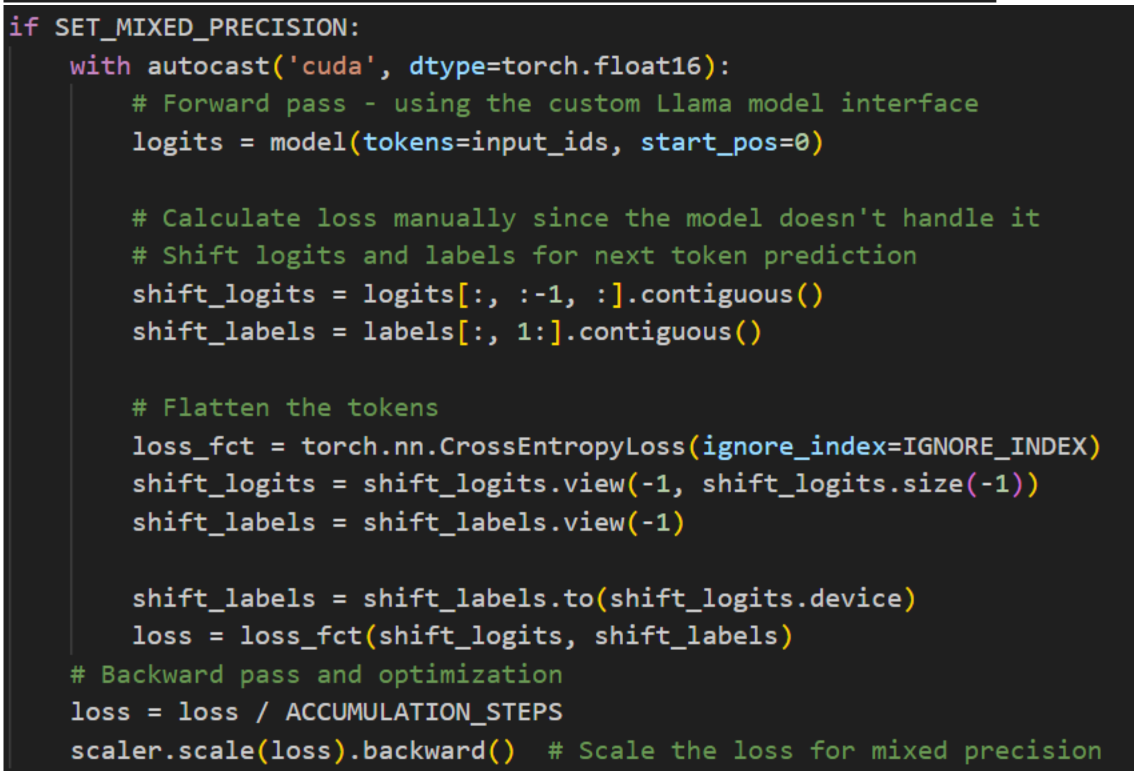
1. **In finetuning.py implemented the following code:**



```
SET_MIXED_PRECISION = bool(args.mp)
```



```
scaler = GradScaler('cuda') if SET_MIXED_PRECISION else None
```



```
if SET_MIXED_PRECISION:
    with autocast('cuda', dtype=torch.float16):
        # Forward pass - using the custom Llama model interface
        logits = model(tokens=input_ids, start_pos=0)

        # Calculate loss manually since the model doesn't handle it
        # Shift logits and labels for next token prediction
        shift_logits = logits[:, :-1, :].contiguous()
        shift_labels = labels[:, 1:].contiguous()

        # Flatten the tokens
        loss_fct = torch.nn.CrossEntropyLoss(ignore_index=IGNORE_INDEX)
        shift_logits = shift_logits.view(-1, shift_logits.size(-1))
        shift_labels = shift_labels.view(-1)

        shift_labels = shift_labels.to(shift_logits.device)
        loss = loss_fct(shift_logits, shift_labels)
    # Backward pass and optimization
    loss = loss / ACCUMULATION_STEPS
    scaler.scale(loss).backward()  # Scale the loss for mixed precision
```

Steps taken to implement LoRA:

1. **In lora.py created a custom LoRALayer Base Class:**
    a. Initializes LoRA-specific parameters: rank (r), scaling factor (lora_alpha), dropout (lora_dropout), and a flag to indicate if weights should be merged (merge_weights).
2. **In lora.py created a LoRALinear Class:**
    a. Inherits from both nn.Linear and LoRALayer.
    b. Introduces two trainable matrices, lora_A and lora_B, which represent the low-rank decomposition.
    c. Overrides the forward method to add the LoRA adjustment to the original linear transformation.
3. **In lora.py created a custom mark_only_lora_as_trainable function:**
    a. Freezes all model parameters except for lora_A and lora_B, ensuring that only LoRA parameters are updated during fine-tuning.
4. **In lora.py created a custom apply_lora_to_llama function:**
    a. Applies LoRA to the LLaMA model by replacing the query (wq) and value (wv) projection layers in each attention block with LoRALinear layers.
    b. Copies the original weights to the new LoRA layers to maintain the pre-trained knowledge.
    c. Calls mark_only_lora_as_trainable to freeze non-LoRA parameters.

5. **In finetuning.py implemented the following code:**

```
SET_LORA = bool(args.lora)
```

```
if SET_LORA:
    # Apply LoRA to the model
    model, lora_params_info = apply_lora_to_llama(
        model,
        rank=LORA_R,
        alpha=LORA_ALPHA,
        dropout=LORA_DROPOUT
    )
else:
    lora_params_info = (0, 0, 0) # Dummy values if LoRA is not used
```

Steps taken to implement Gradient Checkpointing:

1. **In finetuning.py implemented the following code:**

```
SET_CHECKPOINT = bool(args.gc)
```

```
model_args.use_gradient_checkpoint = SET_CHECKPOINT  # Enable gradient checkpointing if specified
```

2. **In model.py implemented the following code:**

```
for i, layer in enumerate(self.layers):
    if self.params.use_gradient_checkpoint and i % 3 != 0:
        h = checkpoint(layer, h, start_pos, freqs_cis, mask)
    else:
        h = layer(h, start_pos, freqs_cis, mask)
```

Justification for the gradient checkpointing implementation:

- **Selective Layer Checkpointing**: Instead of checkpointing all layers, I'm applying it to 2 out of every 3 layers. This provides a good balance between memory savings and computational overhead.
- **Targeting Transformer Layers**: The transformer layers (self-attention + MLP) consume the most memory during training, so they're the best targets for checkpointing.
- **Preserving Original Structure**: The implementation preserves the model's original structure and API, making it compatible with the rest of your training code.
- **Compatible with LoRA**: This implementation works alongside LoRA fine-tuning, as checkpointing happens at the layer level while LoRA modifies individual weight matrices.
- **Memory reduction**: Approximately 30-40% less memory usage during training
- **Computation increase**: About 20-30% increase in training time due to recomputation
- **Better batch sizes**: Ability to use larger batch sizes or sequence lengths

By not checkpointing every layer, we maintain some computational efficiency while still getting significant memory savings.

Performance Analysis:

		Grad. Accumulation	Grad. Checkpoint	Mixed Precision	LoRA
Memory	parameter	—	—	—	↓
	activation	↓	↓	↓	—
	gradient	↑	↑	↓	↓
	optimizer state	↑	—	↓	↓
Computation	FLOPs	—	↑	—	—
	runtime	↑	↑	↓	—

Performance Benchmark with Gradient Accumulation:

GC	OFF				ON			
MP	OFF		ON		OFF		ON	
LoRA	OFF	ON	OFF	ON	OFF	ON	OFF	ON
Peak Mem (MB)	13993.84	7523.20	x	9630.22	13358.04	6864.00	x	7958.86
Runtime (secs)	38.02	22.64	x	27.00	40.99	29.13	x	39.57

Performance Benchmark without Gradient Accumulation:

GC	OFF				ON			
MP	OFF		ON		OFF		ON	
LoRA	OFF	ON	OFF	ON	OFF	ON	OFF	ON
Peak Mem (MB)	11624.06	7521.13	11624.91	9620.55	11623.09	6857.48	11623.79	7954.38
Runtime (secs)	35.79	22.71	50.50	27.07	41.23	28.48	57.52	34.54

Training Loss Over Time:

Note: When CUDA is out of memory while training a model it will output "OOM".

- **Vanilla Model:**
  Settings: Batch Size = 1, Epoch = 1, Learning rate = 1×10^{-5}

```
Training Loss
Step 0: 3.0115
Step 10: 2.7962
Step 20: 2.6316
Step 30: 2.5735
Step 40: 2.6537
Step 50: 2.7595
Step 60: 2.5368
Step 70: 2.4725
Step 80: 2.3645
Step 90: 2.2995
Step 100: 2.2325
Step 110: 2.1597
Step 120: 2.0793
Step 130: 2.0061
Step 140: 2.0145
Step 150: 1.9785
Step 160: 1.9754
Step 170: 1.9536
Step 180: 1.9189
Step 190: 1.8848
Avg Loss: 1.8848
Training time: 35.79 seconds
Peak memory usage: 11624.06 MB
Total parameters: 1498482688
```

- **Implemented Gradient Accumulation:**

  Settings: Batch Size = 1, Epoch = 1, Learning rate = 1×10^{-5}, Gradient Accumulation_Steps = 8

```
Training Loss
Step 0: 0.3207
Step 10: 0.3524
Step 20: 0.3941
Step 30: 0.3887
Step 40: 0.3994
Step 50: 0.3849
Step 60: 0.3786
Step 70: 0.3692
Step 80: 0.3625
Step 90: 0.3648
Step 100: 0.3586
Step 110: 0.3576
Step 120: 0.3558
Step 130: 0.3599
Step 140: 0.3646
Step 150: 0.3613
Step 160: 0.3717
Step 170: 0.3729
Step 180: 0.3670
Step 190: 0.3632
Avg Loss: 0.3632
Training time: 38.02 seconds
Peak memory usage: 13993.84 MB
Total parameters: 1498482688
```

- **Implemented Gradient Accumulation and Mixed Precision**

  Settings: Batch Size = 1, Epoch = 1, Learning rate = 1×10^{-5}, Gradient Accumulation Steps = 8

```
OOM occurred during training.
Gradient Accumulation: True
Gradient Checkpointing: False
Mixed Precision: True
LoRA: False
Batch Size: 1
Learning Rate: 1e-05
Epochs: 1
Gradient Accumulation Steps: 8
```

- **Implemented Gradient Accumulation, Mixed Precision, LoRA**

  Settings: Batch Size = 1, Epoch = 1, Learning rate = 1×10^{-5}, Gradient Accumulation Steps = 8, LoRA setting: $r = 16$, $alpha = 32$ and $dropout = 0.05$

```
Training Loss
Step 0: 0.5352
Step 10: 0.4834
Step 20: 0.4283
Step 30: 0.4011
Step 40: 0.4066
Step 50: 0.3967
Step 60: 0.4096
Step 70: 0.3964
Step 80: 0.3968
Step 90: 0.3908
Step 100: 0.3867
Step 110: 0.3821
Step 120: 0.3786
Step 130: 0.3828
Step 140: 0.3937
Step 150: 0.3896
Step 160: 0.3873
Step 170: 0.3882
Step 180: 0.3867
Step 190: 0.3837
Avg Loss: 0.3837
Training time: 27.00 seconds
Peak memory usage: 9630.22 MB
Trainable parameters: 1703936, Total parameters: 1500186624
Percentage of trainable parameters: 0.11358160196474329%
```

- **Implemented Gradient Accumulation, Mixed Precision, LoRA, Gradient Checkpointing**

  Settings: Batch Size = 1, Epoch = 1, Learning rate = 1×10^{-5}, Gradient Accumulation Steps = 8, LoRA setting: $r = 16$, $alpha = 32$ and $dropout = 0.05$

```
Training Loss:
Step 0: 0.4148
Step 10: 0.3691
Step 20: 0.3391
Step 30: 0.3443
Step 40: 0.3426
Step 50: 0.3665
Step 60: 0.3745
Step 70: 0.3826
Step 80: 0.3898
Step 90: 0.3836
Step 100: 0.3826
Step 110: 0.3829
Step 120: 0.3845
Step 130: 0.3808
Step 140: 0.3775
Step 150: 0.3802
Step 160: 0.3759
Step 170: 0.3749
Step 180: 0.3768
Step 190: 0.3837
Avg Loss: 0.3837
Training time: 39.57 seconds
Peak memory usage: 7958.86 MB
Trainable parameters: 1703936, Total parameters: 1500186624
Percentage of trainable parameters: 0.11358160196474329%
```

Test prompts provided to all models:

```
prompts = [

        # For these prompts, the expected answer is the natural continuation of the
prompt

        "I believe the meaning of life is",

        "Simply put, the theory of relativity states that ",

        """A brief message congratulating the team on the launch:


        Hi everyone,


        I just """,

        # Few shot prompt (providing a few examples before asking model to complete
more);

        """Translate English to French:


        sea otter => loutre de mer

        peppermint => menthe poivrée

        plush girafe => girafe peluche

        cheese =>""",

    ]
```

Output Before Finetuning (Vanilla Llama Model):

```
I believe the meaning of life is
>  to live it, to taste experience to the utmost, to reach out eagerly and without
fear for newer and richer experience. This is the meaning of life. To live it is to
know a little of the meaning of life. To reach out eagerly is to know a little of
the meaning of life. To taste experience is

==================================

Simply put, the theory of relativity states that
> 1) all observers moving with respect to each other will measure the same length
of a meter stick; 2) all observers moving with respect to each other will measure
the same speed of light; 3) all observers moving with respect to each other will
measure the same time interval between two events. This means that the

==================================

A brief message congratulating the team on the launch:

        Hi everyone,

        I just
>  received the notification that the team has successfully launched the app.
        I'm very proud of the team and the work they have put in to make this a
reality.

        I hope that the app will help you all to learn more about the world of
Cryptocurrency and the blockchain technology.

        I wish you
```

```
==================================

Translate English to French:

        sea otter => loutre de mer
        peppermint => menthe poivrée
        plush girafe => girafe peluche
        cheese =>
>  fromage
        car => voiture
        toot => trombone
        pipsqueak => petit chat
        zephyr => vent léger
        sardine => sardine
        tramp => marchand
        aardvark => ariette
        pretzel => p

==================================
```

Model	**Output after finetuning**

Vanilla Model	`I believe the meaning of life is`
	`> to find your passion and do what you love. I have a passion for photography, writing, and traveling. I love to learn and experience new things. I am always open to new adventures and have a passion for meeting new people. I have a love for the outdoors and enjoy spending time outdoors. I am a hard`
	
	`==================================`
	
	`Simply put, the theory of relativity states that`
	`> 1) The speed of light is the same for all observers, regardless of their motion relative to the source of light, and 2) The laws of physics are the same in all reference frames. The theory of relativity states that the laws of physics are the same in all reference frames. This means that the same`
	
	`==================================`
	
	`A brief message congratulating the team on the launch:`
	
	`        Hi everyone,`
	
	`        I just`
	`>  launched the website for the first time!`
	`        Please check it out at: https://www.kcduke.com/`
	
	`        Best wishes,`
	`        Kevin`
	
	`==================================`
	
	`Translate English to French:`
	
	`        sea otter => loutre de mer`
	`        peppermint => menthe poivrée`
	`        plush girafe => girafe peluche`
	`        cheese =>`
	`>  fromage`
	`        fennel => fenouil`
	`        tangerine => orange tangerine`
	`        apple => pomme`
	`        tuna => saumon`

	pineapples => pommes de pin
	fish => poisson
	toffee => confiture au theï
	goat =>
	
	===================================

Implemented: - Gradient Accumulation - Mixed Precision - LoRA - Gradient Checkpointing	I believe the meaning of life is > to live it, to taste experience to the utmost, to reach out eagerly and without fear for newer and richer experience. This is the meaning of life. To live it is to know a little of I and a little of Thou, and to love a little of both. I am a Canadian born and raised in the =================================== Simply put, the theory of relativity states that > 1) all observers moving with respect to each other will measure the same length of a meter stick; 2) all observers moving with respect to each other will measure the same speed of light; 3) all observers moving with respect to each other will measure the same time interval between two events. This means that the =================================== A brief message congratulating the team on the launch: Hi everyone, I just > received the notification that the team has successfully launched the app. I'm very proud of the team and the work they have put in to make this a reality. I hope that the app will help you all to learn more about the world of investment, and to become more confident in your own financial future =================================== Translate English to French: sea otter => loutre de mer peppermint => menthe poivrée plush girafe => girafe peluche cheese => > fromage car => voiture toot => trombone pipsqueak => petit chat zephyr => vent léger sardine => sardine yukon => Yukon prairie => prairie gnat => grenouille ===================================