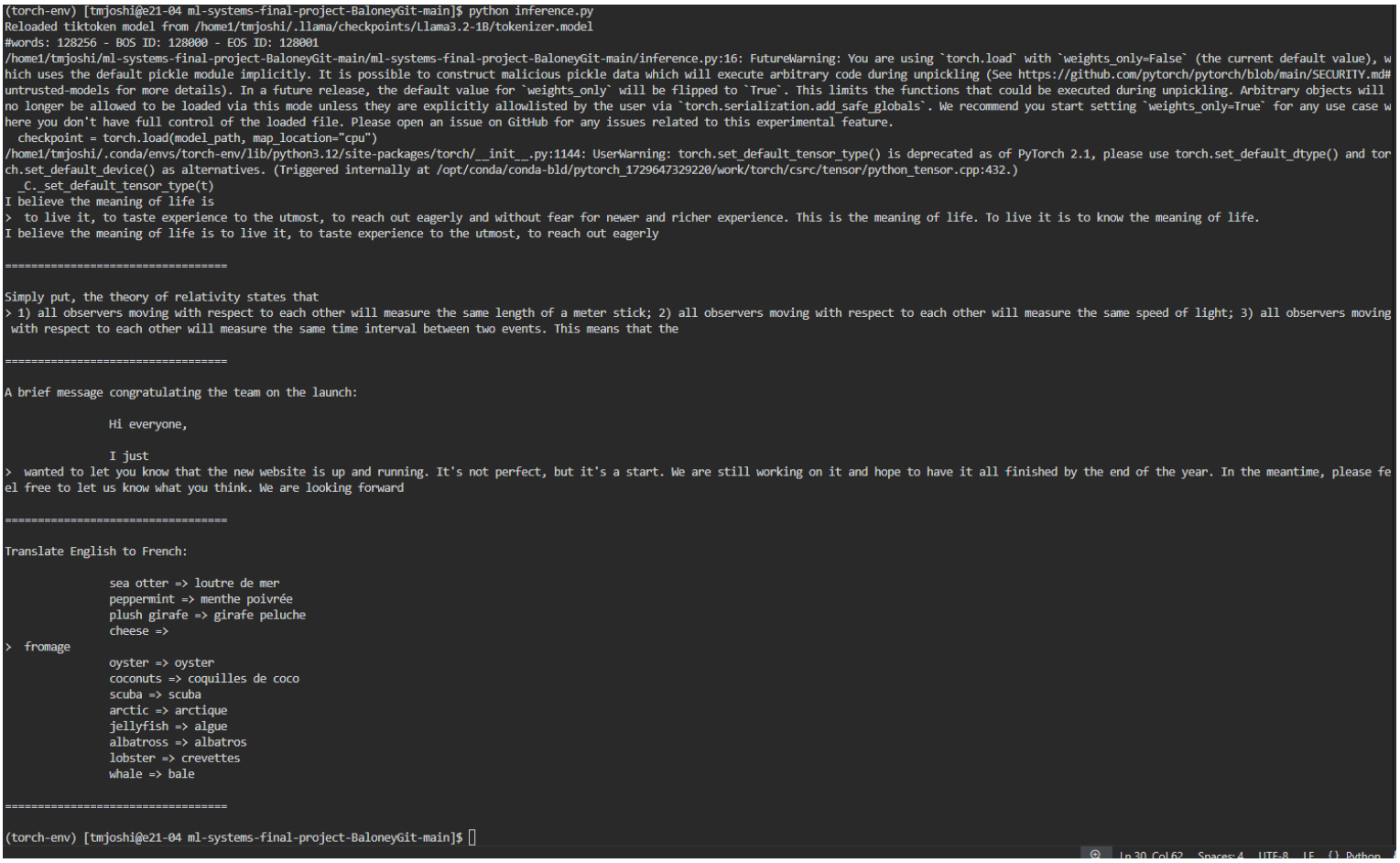
Phase 2

CARC P100 KV Cache enabled inference.py output:

```
(torch-env) [tmjoshi@e21-04 ml-systems-final-project-BaloneyGit-main]$ python inference.py
Reloaded tiktoken model from /home1/tmjoshi/.llama/checkpoints/Llama3.2-1B/tokenizer.model
#words: 128256 - BOS ID: 128000 - EOS ID: 128001
/home1/tmjoshi/ml-systems-final-project-BaloneyGit-main/ml-systems-final-project-BaloneyGit-main/inference.py:16: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), w
hich uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#
untrusted-models for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will
no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True` for any use case w
here you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feature.
  checkpoint = torch.load(model_path, map_location="cpu")
/home1/tmjoshi/.conda/envs/torch-env/lib/python3.12/site-packages/torch/__init__.py:1144: UserWarning: torch.set_default_tensor_type() is deprecated as of PyTorch 2.1, please use torch.set_default_dtype() and tor
ch.set_default_device() as alternatives. (Triggered internally at /opt/conda/conda-bld/pytorch_1729647329220/work/torch/csrc/tensor/python_tensor.cpp:432.)
  _C._set_default_tensor_type(t)
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
>  wanted to let you know that the new website is up and running. It's not perfect, but it's a start. We are still working on it and hope to have it all finished by the end of the year. In the meantime, please fe
el free to let us know what you think. We are looking forward

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

Translate English to French:

        sea otter => loutre de mer
        peppermint => menthe poivrée
        plush girafe => girafe peluche
        cheese =>
>  fromage
        oyster => oyster
        tortoise => tortue
        pineapple => ananas
        loofah => luffe
        rubber duck => canard en caoutchouc
        canary => canard
        eel => loutre
        coconut => co

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

(torch-env) [tmjoshi@e21-04 ml-systems-final-project-BaloneyGit-main]$
```

CARC P100 KV Cache disabled inference.py output:



```
(torch-env) [tmjoshi@e21-04 ml-systems-final-project-BaloneyGit-main]$ python inference.py
Reloaded tiktoken model from /home1/tmjoshi/.llama/checkpoints/Llama3.2-1B/tokenizer.model
#words: 128256 - BOS ID: 128000 - EOS ID: 128001
/home1/tmjoshi/ml-systems-final-project-BaloneyGit-main/ml-systems-final-project-BaloneyGit-main/inference.py:16: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), w
hich uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#
untrusted-models for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will
no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True` for any use case w
here you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feature.
  checkpoint = torch.load(model_path, map_location="cpu")
/home1/tmjoshi/.conda/envs/torch-env/lib/python3.12/site-packages/torch/__init__.py:1144: UserWarning: torch.set_default_tensor_type() is deprecated as of PyTorch 2.1, please use torch.set_default_dtype() and tor
ch.set_default_device() as alternatives. (Triggered internally at /opt/conda/conda-bld/pytorch_1729647329220/work/torch/csrc/tensor/python_tensor.cpp:432.)
  _C._set_default_tensor_type(t)
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 the meaning of life.
I believe the meaning of life is to live it, to taste experience to the utmost, to reach out eagerly

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

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
>  wanted to let you know that the new website is up and running. It's not perfect, but it's a start. We are still working on it and hope to have it all finished by the end of the year. In the meantime, please fe
el free to let us know what you think. We are looking forward

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

Translate English to French:

                sea otter => loutre de mer
                peppermint => menthe poivrée
                plush girafe => girafe peluche
                cheese =>
>  fromage
                oyster => oyster
                coconuts => coquilles de coco
                scuba => scuba
                arctic => arctique
                jellyfish => algue
                albatross => albatros
                lobster => crevettes
                whale => bale

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

(torch-env) [tmjoshi@e21-04 ml-systems-final-project-BaloneyGit-main]$
```

Differences in KV Cache enabled vs disabled prompt outputs:

1. Prompt 1: No change
2. Prompt 2: No change
3. Prompt 3: the first generation and corresponding translation is the same, the rest of the generations and corresponding translations are different

Changes made to codebase for disabling KV Cache:

1. inferency.py:

- In model.generate() function call, set kv_caching = False:

```python
def inference():
    tokenizer = Tokenizer(tokenizer_path)

    checkpoint = torch.load(model_path, map_location="cpu")
    model_args = ModelArgs()
    torch.set_default_tensor_type(torch.cuda.HalfTensor) # load model in fp16
    model = Llama(model_args)
    model.load_state_dict(checkpoint, strict=True)
    device = "cuda"
    model.to(device)

    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 =>""",
    ]

    model.eval()
    results = model.generate(tokenizer, prompts, max_gen_len=64, temperature=0.6, top_p=0.9, kv_caching=False, device=device)

    for prompt, result in zip(prompts, results):
        print(prompt)
        print(f"> {result['generation']}")
        print("\n==================================\n")
```

2. model.py:

- In the Attention class (class Attention(nn.Module)), in the forward pass definition (def forward()), change the else condition for the 'if self.kv_caching' to:

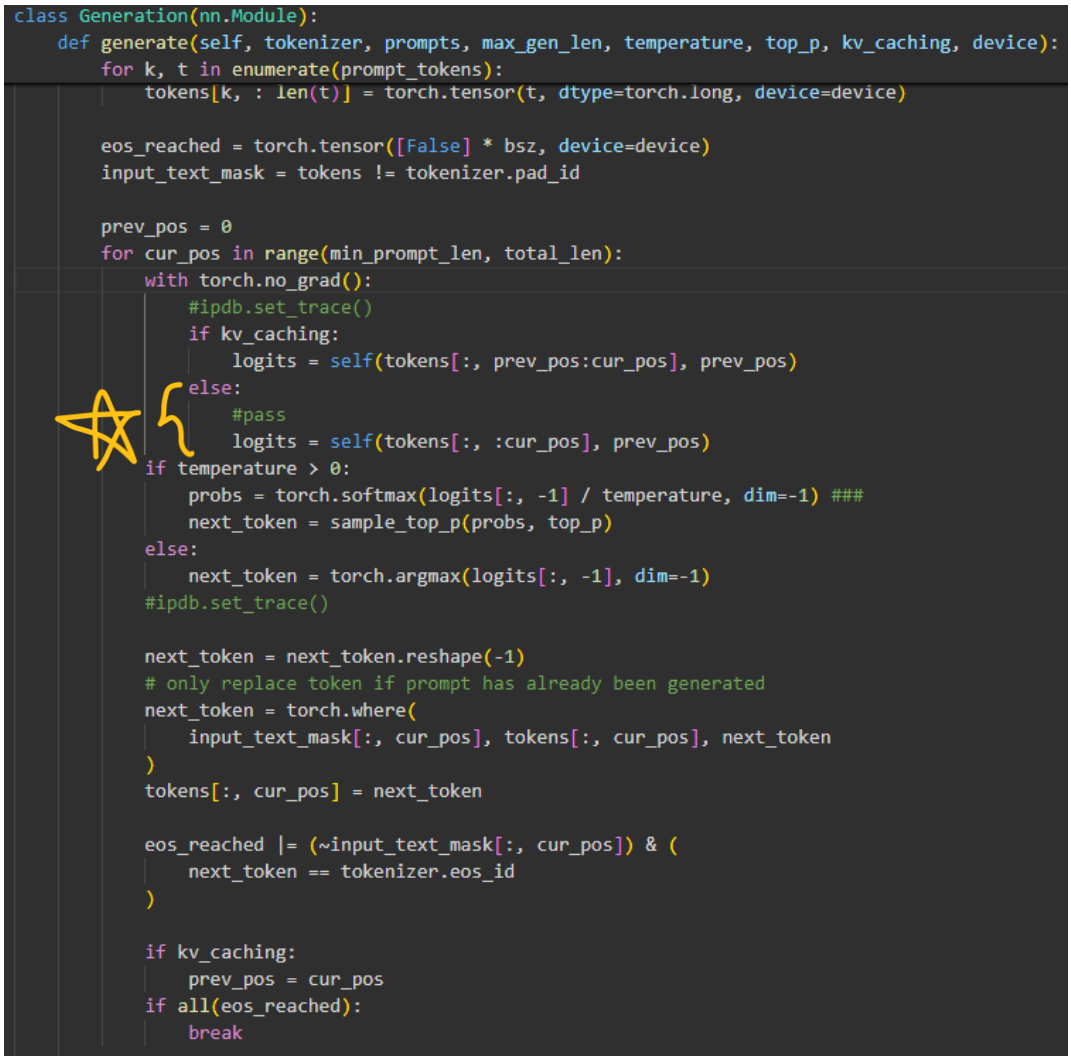
```python
class Attention(nn.Module):
    ):
        """
        Forward pass of the attention module.

        Args:
            x (torch.Tensor): Input tensor.
            start_pos (int): Starting position for caching.
            freqs_cis (torch.Tensor): Precomputed frequency tensor.
            mask (torch.Tensor, optional): Attention mask tensor.

        Returns:
            torch.Tensor: Output tensor after attention.

        """
        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
            keys = xk
            values = xv
```

- In ModelArgs class (class ModelArgs), set kv_caching: bool = False:

```python
# Copyright (c) Meta Platforms, Inc. and affiliates.
# This software may be used and distributed in accordance with the terms of the Llama 3 Community License Agreement.

import math
from dataclasses import dataclass
from typing import Optional, Tuple

import torch
import torch.nn.functional as F
from torch import nn

from llama.generation import Generation


@dataclass
class ModelArgs: # fixed model configurations for Llama3.2-1B
    dim: int = 2048
    n_layers: int = 16
    n_heads: int = 32
    n_kv_heads: int = 8
    vocab_size: int = 128256
    multiple_of: int = 256  # make SwiGLU hidden layer size multiple of large power of 2
    ffn_dim_multiplier: float = 1.5
    norm_eps: float = 1e-5
    rope_theta: float = 500000

    max_batch_size: int = 4  # for kv caching pre-allocation
    max_seq_len: int = 256   # for kv caching pre-allocation

    kv_caching: bool = False # enable/disable KV Cache
```

3. generation.py:

- in the Generation class (class Generation(nn.Module)), in the generate definition (def generate()), change the else condition for the ‘if kv_caching’ condition in the ‘for cur_pos in range(min_prompt_len, total_len)’ loop like this:



```
class Generation(nn.Module):
    def generate(self, tokenizer, prompts, max_gen_len, temperature, top_p, kv_caching, device):
        for k, t in enumerate(prompt_tokens):
            tokens[k, : len(t)] = torch.tensor(t, dtype=torch.long, device=device)

        eos_reached = torch.tensor([False] * bsz, device=device)
        input_text_mask = tokens != tokenizer.pad_id

        prev_pos = 0
        for cur_pos in range(min_prompt_len, total_len):
            with torch.no_grad():
                #ipdb.set_trace()
                if kv_caching:
                    logits = self(tokens[:, prev_pos:cur_pos], prev_pos)
                else:
                    #pass
                    logits = self(tokens[:, :cur_pos], prev_pos)
            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)
            #ipdb.set_trace()

            next_token = next_token.reshape(-1)
            # only replace token if prompt has already been generated
            next_token = torch.where(
                input_text_mask[:, cur_pos], tokens[:, cur_pos], next_token
            )
            tokens[:, cur_pos] = next_token

            eos_reached |= (~input_text_mask[:, cur_pos]) & (
                next_token == tokenizer.eos_id
            )

            if kv_caching:
                prev_pos = cur_pos
            if all(eos_reached):
                break
```