LLMs Holiday Assignment

Student: Fausto Bravo Cuvi

Brief explanation of what was done:

This assignment consisted in getting a hands on introduction into LLMS by watching a YouTube tutorial "Let's build GPT: from scratch, in code, spelled out" from Andrej Karpathy's (a renowned AI profesional). And after watching his video I decide to look into his nanoGPT repository, which ended up being been really enlightening, and actually is the repository I used to set up everything to test my own model because I had some problems with the google colab enviroment.

The tutorial, with its hands-on approach, demystified the building blocks of GPT models, while the nanoGPT repository provided a practical platform for experimentation. This exploration was further enriched by applying these learnings to a real-world dataset, in my case I used a text file on Chilean Navy history, to test the model's text generation capabilities after being trained.

Challenges like setting up the environment and replacing unavailable libraries, such as 'tiktoken', with suitable alternatives like Hugging Face's transformers, were part of this learning curve. Future directions include learning how to enhance model performance through fine-tuning, integrating diverse functionalities, and scaling up the model for complex tasks.

This journey into AI and natural language processing, particularly through hands-on learning with GPT models, has been a practical eye opener. It's more than just understanding theories; it's about applying these concepts to real-world scenarios, like the small text I used to train the model on some Chilean Navy's history. This process not only sharpens technical skills but also nurtures a problem-solving mindset,

This experience has shown to me the vast potential in AI and natural language processing, moving from conceptual understanding to practical application and

optimization of LLMs.

Detailed walkthrough of what was done for this assignment:

Created an Anaconda env and then started it:

```
conda activate nanoGPT
```

Cloned the GitHub repository

```
git clone https://github.com/karpathy/nanoGPT.git
```

Installed needed dependencies

```
pip install torch numpy transformers datasets tiktoken wandb t
```

Difficulty: I was not able to install pytorch with pip install. So I looked at the pytorch documentation and discovered I had to use conda:

```
conda install pytorch::pytorch torchvision torchaudio -c pytor
```

Then to check if I had pytorch correctly installed I run the following commands in the terminal:

```
python -c "import torch; print(torch.__version__)"

python -c "import torch; x = torch.rand(5, 3); print(x)"
```

```
#results:
tensor([[0.6958, 0.1172, 0.6493],
        [0.8367, 0.7028, 0.9675],
        [0.3068, 0.4344, 0.1318],
        [0.7367, 0.8451, 0.9993],
        [0.4607, 0.0719, 0.7405]])
```

```
pytho
n -c "import torch; print(torch.backends.mps.is_available())"

# results
True
```

```
python -c "import torch; print(torch.backends.mps.is_built())"

# results
True
```

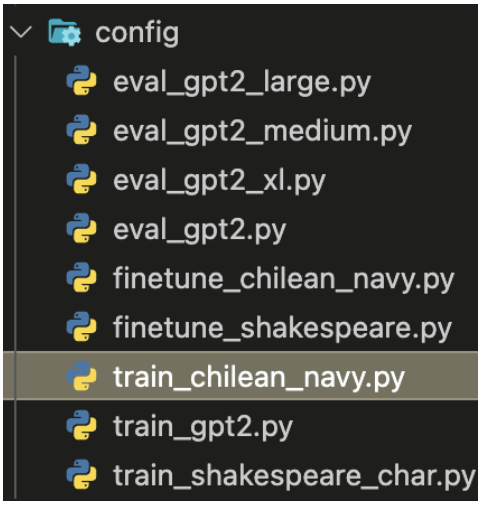
Then, I runned the **prepare.py** script with a dataset that I stored at my github account: “https://raw.githubusercontent.com/gitbravehub/nanoGPT_navy/main/chilean_navy.txt” It was a sample created specific for this exercise, about some history of the Chilean Navy.

Here are the results I got:

```
(nanoGen) faustobravocuvi@MacBook-Pro-de-Fausto nanoGPT % pyth

length of dataset in characters: 3,038
all the unique characters:
 '(),-.012345789ABCDEFHILMNOPSTUWabcdefghiklmnopqrstuvwxyz
vocab size: 59
train has 2,734 tokens
val has 304 tokens
```

Then run the **train.py** script to train the model on our **chilean_navy.txt (**https://raw.githubusercontent.com/gitbravehub/nanoGPT_navy/main/chilean_navy.txt)

But first we have to create inside the "**config**" folder a "training scrypt". I used as an example the "**train_shakespeare_char.py"** and create a new **"train_chilean_navy.py".** After this was done I runned in the terminal the following code: (`--device=mps` (short for "Metal Performance Shaders") because we are using a macOS with a M1 chip, as suggested in the nanoGPT Documentation)




```
python train.py config/train_chilean_navy.py --device=mps
```

```
Overriding config with config/train_chilean_navy.py:
# train a miniature character-level shakespeare model
# good for debugging and playing on macbooks and such

out_dir = 'out-chilean-navy-'
eval_interval = 250 # keep frequent because we'll overfit
eval_iters = 200
log_interval = 10 # don't print too too often

# we expect to overfit on this small dataset, so only save whe
always_save_checkpoint = False
```

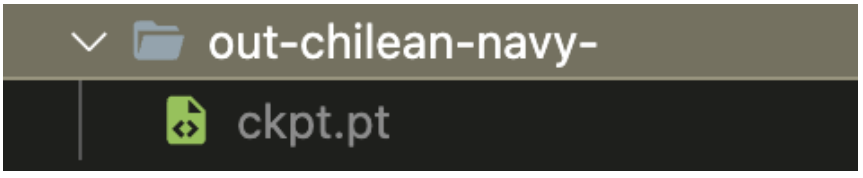
```
wandb_log = False # override via command line if you like
wandb_project = 'chilean-navy'
wandb_run_name = 'mini-gpt'

dataset = 'chilean_navy'
gradient_accumulation_steps = 1
batch_size = 64
block_size = 256 # context of up to 256 previous characters

# baby GPT model :)
n_layer = 6
n_head = 6
n_embd = 384
dropout = 0.2

learning_rate = 1e-3 # with baby networks can afford to go a b
max_iters = 5000
lr_decay_iters = 5000 # make equal to max_iters usually
min_lr = 1e-4 # learning_rate / 10 usually
beta2 = 0.99 # make a bit bigger because number of tokens per

warmup_iters = 100 # not super necessary potentially

# on macbook also add
device = 'cpu'  # run on cpu only
compile = False # do not torch compile the model

Overriding: device = mps
tokens per iteration will be: 16,384
found vocab_size = 59 (inside data/chilean_navy/meta.pkl)
Initializing a new model from scratch
number of parameters: 10.64M
/Users/faustobravocuvi/opt/anaconda3/envs/nanoGen/lib/python3.
  warnings.warn(
num decayed parameter tensors: 26, with 10,737,792 parameters
num non-decayed parameter tensors: 13, with 4,992 parameters
using fused AdamW: False
```

Screenshot of the Training Process:

```
num decayed parameter tensors: 26, with 10,737,792 parameters
num non-decayed parameter tensors: 13, with 4,992 parameters
using fused AdamW: False
step 0: train loss 4.1804, val loss 4.2183
iter 0: loss 4.1670, time 204981.82ms, mfu -100.00%
iter 10: loss 3.0657, time 1425.02ms, mfu 0.26%
iter 20: loss 2.5838, time 1416.84ms, mfu 0.26%
iter 30: loss 2.3646, time 1403.27ms, mfu 0.26%
iter 40: loss 2.2782, time 1408.57ms, mfu 0.26%
iter 50: loss 2.1921, time 1401.05ms, mfu 0.26%
iter 60: loss 2.1454, time 1456.50ms, mfu 0.26%
iter 70: loss 2.0453, time 1429.31ms, mfu 0.26%
iter 80: loss 1.9800, time 1395.86ms, mfu 0.26%
iter 90: loss 1.8270, time 1455.41ms, mfu 0.26%
iter 100: loss 1.6557, time 1416.69ms, mfu 0.26%
iter 110: loss 1.4934, time 1414.07ms, mfu 0.26%
iter 120: loss 1.2358, time 1440.50ms, mfu 0.26%
iter 130: loss 1.0535, time 1464.49ms, mfu 0.26%
iter 140: loss 0.8677, time 1457.59ms, mfu 0.26%
```

It was interesting to see that during the training process the progress was saved in a checkpoint, just like it was explained during the Youtube video:

```
iter 80: loss 1.9800, time 1395.86ms, mfu 0.26%
iter 90: loss 1.8270, time 1455.41ms, mfu 0.26%
iter 100: loss 1.6557, time 1416.69ms, mfu 0.26%
iter 110: loss 1.4934, time 1414.07ms, mfu 0.26%
iter 120: loss 1.2358, time 1440.50ms, mfu 0.26%
iter 130: loss 1.0535, time 1464.49ms, mfu 0.26%
iter 140: loss 0.8677, time 1457.59ms, mfu 0.26%
iter 150: loss 0.7154, time 1426.51ms, mfu 0.26%
iter 160: loss 0.5725, time 1417.64ms, mfu 0.26%
iter 170: loss 0.4628, time 1396.45ms, mfu 0.26%
iter 180: loss 0.3784, time 1407.64ms, mfu 0.26%
iter 190: loss 0.2961, time 1442.65ms, mfu 0.26%
iter 200: loss 0.2513, time 1440.84ms, mfu 0.26%
iter 210: loss 0.2152, time 1428.31ms, mfu 0.26%
iter 220: loss 0.1906, time 1416.89ms, mfu 0.26%
iter 230: loss 0.1628, time 1464.32ms, mfu 0.26%
iter 240: loss 0.1370, time 1447.16ms, mfu 0.26%
step 250: train loss 0.0569, val loss 3.6022
saving checkpoint to out-chilean-navy-
iter 250: loss 0.1295, time 196736.26ms, mfu 0.23%
iter 260: loss 0.1168, time 1469.05ms, mfu 0.24%
```

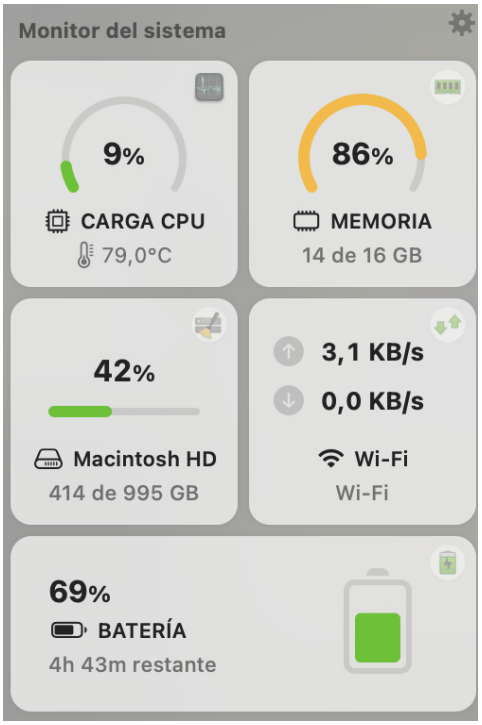


Curiosity made me search what these lines meant and found the following insights:

1. **Iteration ("iter") Numbers**: Each of these lines starts with "iter" and a number, right? These numbers (like 250, 260, 270) are basically telling us which round of training the model is on. Think of it like doing reps at the gym. Each iteration is one rep, where the model tries to lift the data, sees how well it does, and then tries to do it a bit better next time.
2. **Loss**: The "loss" numbers you see next to each iteration (like 0.1295, 0.1168) are super important. They're like the model's scorecard. It shows how much the model's guesses are off from the actual answers. The lower this number gets over time, the better the model is getting at making predictions. It's like the model's getting smarter with each rep.
3. **Time**: This is just how long each iteration takes, measured in milliseconds (ms). So, when you see "196736.26ms", it means that particular iteration took 196,736.26 milliseconds to complete. We care about this because we don’t want our training to be super slow, especially if we have a ton of data.
4. **MFU (Most Frequent Updates)**: The "mfu" percentage (like 0.23%, 0.24%) is probably telling us about specific parts of the model that are getting updated a lot. Imagine you're playing a video game and you keep hitting the same button because it's working for you. In the model, if a part is getting updated a lot (a high mfu), it's like it’s hitting that button a bunch. But we have to be careful that the model isn’t just learning one trick, but is actually getting better overall.

So, in short, these lines are like a quick update on how our model's training session is going. We're looking for the loss to go down, which means our model is learning, and keeping an eye on the time and MFU to make sure everything's running smoothly and efficiently.

Also something interesting to notice while training the model, was to monitor how much memory it used while doing so (86%).



After arriving to another checkpoint during the training step I decide to terminate it, because of a possible overfitting, given by the value: **“val loss 3.7870”. In total I was able to run 760 iterations.**

```
iter 720: loss 0.0397, time 1407.59ms, mfu 0.26%
iter 730: loss 0.0412, time 1390.35ms, mfu 0.26%
iter 740: loss 0.0407, time 1406.80ms, mfu 0.26%
step 750: train loss 0.0266, val loss 3.7870
iter 750: loss 0.0401, time 196856.33ms, mfu 0.24%
iter 760: loss 0.0431, time 1399.94ms, mfu 0.24%
^CTraceback (most recent call last):
```

Then I proceeded with the next step that consisted in generating a sample from the trained model, by using the **“sample.py”** script. Unluckily I was unable to install the `tiktoken` **Library** used for encoding and decoding text for a GPT-2 model.

```
#import tiktoken
from transformers import GPT2Tokenizer
from model import GPTConfig, GPT
```

Because I wanted to continue learning how to train a model over a specific dataset I changed the `tiktoken` **Library** by the `transformers` **Library by Hugging Face.**

Here you can see the modified code in the **“sample.py”** script:

- Original part of the code:

```
THE CODE WAS COMMENTED, NOT ERASED

# look for the meta pickle in case it is available in the d
# load_meta = False
# if init_from == 'resume' and 'config' in checkpoint and '
#     meta_path = os.path.join('data', checkpoint['config']
#     load_meta = os.path.exists(meta_path)
# if load_meta:
#     print(f"Loading meta from {meta_path}...")
#     with open(meta_path, 'rb') as f:
#         meta = pickle.load(f)
#     # TODO want to make this more general to arbitrary en
#     stoi, itos = meta['stoi'], meta['itos']
#     encode = lambda s: [stoi[c] for c in s]
#     decode = lambda l: ''.join([itos[i] for i in l])
# else:
#     # ok let's assume gpt-2 encodings by default
#     print("No meta.pkl found, assuming GPT-2 encodings...
#     enc = tiktoken.get_encoding("gpt2")
#     encode = lambda s: enc.encode(s, allowed_special={"<|
#     decode = lambda l: enc.decode(l)
```

- Modified code:

```
from transformers import GPT2Tokenizer

model.eval()
model.to(device)
if compile:
    model = torch.compile(model) # requires PyTorch 2.0 (op
```

```
# look for the meta pickle in case it is available in the d
load_meta = False
if init_from == 'resume' and 'config' in checkpoint and 'da
    meta_path = os.path.join('data', checkpoint['config']['
    load_meta = os.path.exists(meta_path)
if load_meta:
    print(f"Loading meta from {meta_path}...")
    with open(meta_path, 'rb') as f:
        meta = pickle.load(f)
    stoi, itos = meta['stoi'], meta['itos']
    encode = lambda s: [stoi[c] for c in s]
    decode = lambda l: ''.join([itos[i] for i in l])
else:
    # Using GPT-2 tokenizer
    print("No meta.pkl found, using GPT-2 tokenizer...")
    tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
    encode = lambda s: tokenizer.encode(s, add_special_toke
    decode = lambda l: tokenizer.decode(l)
```

Results:

After running **sample.py,** I had to wait for a couple of minutes and received the following text as an output, showing that the model dod not get perfectly trained but it got a lot from the training 🚀:

```
(nanoGen) faustobravocuvi@MacBook-Pro-de-Fausto nanoGPT % pyth
```

Results after training and running the **sample.py** script:

```
Loading meta from data/chilean_navy/meta.pkl...

Wr I and World War II, Chile maintained a position of neutrali
Cold War Era (1947-1991)
Throughout the Cold War, the Chilean Navy aligned with Western
---------------
```

```
The War of the Pacific (1879-1884)
One of the most significant periods in Chilean naval history i


Modernization Efforts (Late 19th Century - Early 20th Century)
The late 19th and early 20th cent
---------------

Curnt World War I and World War II, Chile maintained a positio

Cold War Era (1947-1991)
Throughout the Cold War, the Chilean Navy aligned with Western
---------------

The Chilean Navy stands as modern and capable maritime force,

Conclusion
The history of the Chilean Navy igneed was ors, a ruating cont
---------------

The Navy stands as modern and capable maritime force, playing

Conclus
The history of the Chilean Navy ign tobal te atien at me are i

Conclusion
The history of the Chilean Navy i in tes a te ment resentsand
---------------

Ostober 4, 1818, following the independence of Chile. Lord Coc

The War of the Pacific (1879-1884)
One of the most significant periods in Chilean naval history i
---------------

I (191879-1884)
One the most significant periods in Chilean naval history is t
```

```
Modernization Efforts (Late 19th Century - Early 20th Century)
The late 19th and early 20th centuries saw significant m
---------------

ornization, its fleet and reginal maritime history, has a rich

Early (1817-1830)
The Chilean Navy was officiallly established on October 4, 181
---------------

21st Century, the Chilean Navy continues to play a vital role

Current Status
TToday, the Chilean Navy stands as a modern and capable mariti
---------------

The War of the Pacific (1879-184)
One of the most significant periods in Chilean naval history i


Modernization Efforts (Late 19th Century - Early 20th Century)
The late 19th and early 20th centuries sa
```