



Department of Electrical and Electronics Engineering

EEE 443: Neural Networks

Class Project VIII Report

Name and Surname: Ali Aral Takak

Student ID: 22001758

Department: EEE

Instructor: Erdem Koyuncu

Introduction

The eighth project of the course focuses on experimenting with different aspects of large language models (LLMs), using nanoGPT; a minimal implementation of GPT-2 large language model. In this project, we will explore the architecture of the GPT-2, and by tuning its parameters, observe the effects of parameters adjustment and fine-tuning.

Assignment Questions

Q2) Run the code given in the assignment file.

The figure provided below displays the output of the sample code given in the instruction file:

```
one ring to rule them all, hence the strong bonds of peace, peace and tranquillity that have been affixed to the community, and to the great spiritual union between those communiti
es."
Prayers for peace, peace and tranquillity are not only a common truth in this age, but also a founding principle of the First Amendment.
Not only does the First Amendment protect individual rights, but it also protects rights or freedoms from the tyranny of big business. As Pope Benedict XVI explained in 2009, "The
one ring to rule them all, because they always have. That is what happens in life, but this week's answer was the best.
And that's why I believe in this man.
I am the best player ever, the best player ever, the best player ever. I am the greatest player ever. I'm the greatest player ever. And I am the greatest player ever. I'm the great
est player ever.
I know the only way I could possibly handle it is to play with me.
=====
one ring to rule them all, but an even more effective one, that of a group of six or seven, to fight for their own ends.
In the past, fighting had been a job of choice for the Alliance. The Iron Horde and Dark Portal, and the Horde itself, had been responsible for most of the conflict. But since the
beginning of the war, the Horde had become utterly consumed by fighting, and were willing to pay for it with their own blood, and as such was not willing to go along with
o (base) aral@Alis-MacBook-Pro nanoGPT-master %
```

Figure 1: Output of the code given in the instructions.

Hence, we infer that the GPT setup is handled properly and works as intended.

Q3) Briefly explain how the model generates new words in terms of model architecture, number of layers, activation functions etc. You may assume the reader knows what a multihead attention block is, so you do not have to explain the very basics.

GPT2 generates text in an autoregressive fashion by repeatedly feeding its own outputs back into the model. At each step, the sequence of tokens generated up to that point is first converted into continuous representation via learned token and positional embeddings. These embedding then pass through a stack of **12 identical Transformer blocks, each comprising a causal multi-head self-attention layer with 12 heads and residual connections, followed by a position-wise feedforward network of inner dimension of 3072 layers that uses Gaussian Error Linear Unit (GELU) activation** [1]. The final hidden-state corresponding to the most recent token is linearly projected to a 50 257-dimensional vector of logits, which are normalized via softmax to yield a probability distribution over the next token in the vocabulary. Hence, a token is sampled from this distribution, appended to the sequence, and the process repeats until generation completes.

Q4) Briefly explain the purpose of “start”, “num samples” and “max_new_tokens” parameters.

The three parameters given in the question control how generation begins, how many outputs are produced, and how long each continuation can grow. The “start” argument defines the initial text prompt that directs the model the context of generation. The “num_samples” parameter determines how many independent continuations from the script will be produced, allowing comparison of multiple outputs from the same prompt. The “max_new_tokens” parameter sets a cap on the number of tokens added beyond the prompt in order to ensure that generation stops once this limit is reached.

Q5) Choose a prompt of your choice, change the temperature to 0.1, and include the outputs of your results in your report. What is the purpose of the temperature? Why do the outputs behave the way they do?

For this question, we chose the prompt “A long time ago in a galaxy far, far away...”. The results of the prompt can be observed in the figure given below, with temperature parameter set to 0.1:

[illegible]

Figure 2: Text generation results with temperature = 0.1.

The temperature parameter regulates the model’s next token distribution by dividing the raw logits before the softmax application. If z_i are the unnormalized log-probabilities for each token i , then temperature T rescales them as:

$$p_i = \frac{\exp (z_i / T)}{\sum_j \exp (z_j / T)}$$

When $T < 1$, the distribution becomes more amplified, amplifying the highest-scoring log-probabilities and suppressing lower-scoring ones; on the other hand, $T > 1$ flattens the distribution, making lower probability tokens relatively more likely.

Q6) Choose a prompt of your choice, change the temperature to 10, and include the outputs of your results in your report. Why do the outputs behave the way they do?

The figure given below displays the results when the temperature is set to 10, with the same prompt in the question 5:

```

(base) aral@Alis-MacBook-Pro nanoGPT-master % python sample.py \
--init_from=gpt2 \
--start="A long time ago in a galaxy far, far away..." \
--num_samples=3 --max_new_tokens=100
Overriding: init_from = gpt2
Overriding: start = A long time ago in a galaxy far, far away...
Overriding: num_samples = 3
Overriding: max_new_tokens = 100
loading weights from pretrained gpt: gpt2
forcing vocab_size=50257, block_size=1024, bias=True
overriding dropout rate to 0.0
number of parameters: 123.65M
/opt/anaconda3/lib/python3.12/site-packages/huggingface_hub/file_download.py:896: FutureWarning: 'resume_download' is deprecated and will be removed in version 1.0.0. Downloads alw
ays resume when possible. If you want to force a new download, use 'force_download=True'.
warnings.warn(
No meta.pkl found, assuming GPT-2 encodings...
A long time ago in a galaxy far, far away...only light doesnning such far objects couldn ereveyly exist out." ) PENTSULE TO RELIGENCIES!

PARTB ANGORE-LASHIVIC COMIL OF IT ANAYRATHISM / BY RTA Klaptronei = YOM! PERSCUYST P: "To everyone involved!! Nobody save lives especially child...Kuhoooo-oh !!o!" ITGY DID BURRAD
E Greetings Tl you

A long time ago in a galaxy far, far away...The Elder Egon may grow the fartelkii is small this week not like anyone on Glimtv have this beast look super early the eocidal casserid
o had spout our last ever episode 4 not known . :; the two very colorful fishes The eagle are floating! ...and so round :^ gags going our over its tew is for TATLAP your wut these
just to whech around yair efter just pherochtu tatl... how

A long time ago in a galaxy far, far away... - ancient artifact can manifest like that.. Let IT hold new messages without putting to fight me Or when God revealed which thoughts Yo
u threw past my nightmares!" Ori spoke and passed alone as Time grew greater

by to Darkest day Balfazza gazed away glottering expression While Fauna-tat explained which to form That Time reached across entire dour nation/eg A-Time The past always arrived
At his mercy. Of great number how Svega dagon Was sure S

(base) aral@Alis-MacBook-Pro nanoGPT-master %

```

Figure 3: Text generation results with temperature = 10.

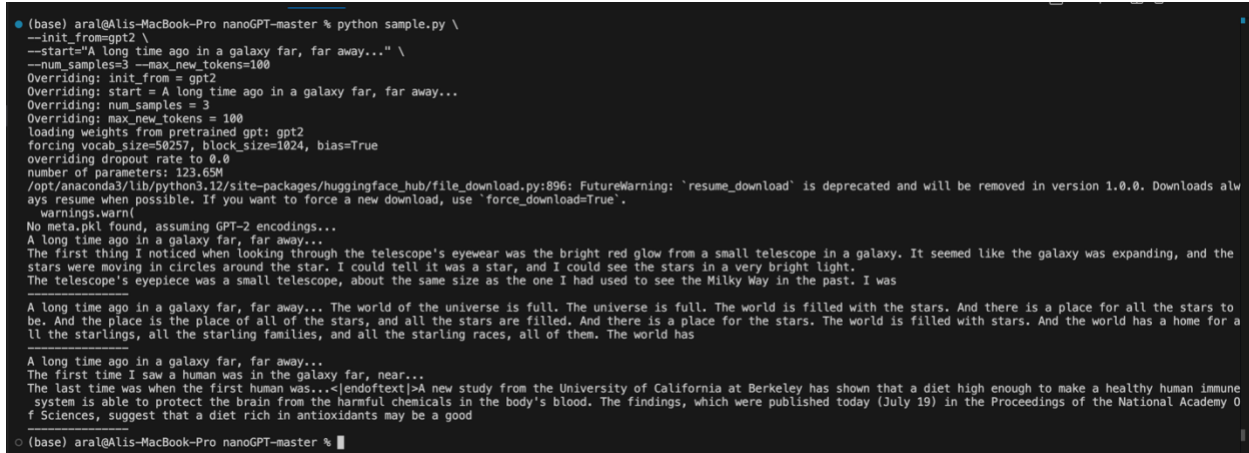
The temperature parameter regulates the model's next token distribution by dividing the raw logits before the softmax application. If z_i are the unnormalized log-probabilities for each token i , then temperature T rescales them as:

$$p_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

When $T < 1$, the distribution becomes more amplified, amplifying the highest-scoring log-probabilities and suppressing lower-scoring ones; on the other hand, $T > 1$ flattens the distribution, making lower probability tokens relatively more likely.

Q7) Keep the temperature a 10 but change top_k to 2. Describe the reasons for the change in outputs as compared with the previous subquestion.

The figure given below displays the results of text generation when the temperature is set to 10 and top_k parameters is reduced to 2 from 200:



```
(base) aral@Alis-MacBook-Pro nanoGPT-master % python sample.py \
--init_from=gpt2 \
--start="A long time ago in a galaxy far, far away..." \
--num_samples=3 --max_new_tokens=100
Overriding: init_from = gpt2
Overriding: start = A long time ago in a galaxy far, far away...
Overriding: num_samples = 3
Overriding: max_new_tokens = 100
loading weights from pretrained gpt: gpt2
forcing vocab_size=50257, block_size=1024, bias=True
overriding dropout rate to 0.0
number of parameters: 123.65M
/opt/anaconda3/lib/python3.12/site-packages/huggingface_hub/file_download.py:896: FutureWarning: 'resume_download' is deprecated and will be removed in version 1.0.0. Downloads alw
ays resume when possible. If you want to force a new download, use 'force_download=True'.
warnings.warn(
No meta.pkl found, assuming GPT-2 encodings...
A long time ago in a galaxy far, far away...
The first thing I noticed when looking through the telescope's eyewear was the bright red glow from a small telescope in a galaxy. It seemed like the galaxy was expanding, and the
stars were moving in circles around the star. I could tell it was a star, and I could see the stars in a very bright light.
The telescope's eyepiece was a small telescope, about the same size as the one I had used to see the Milky Way in the past. I was
A long time ago in a galaxy far, far away... The world of the universe is full. The universe is full. The world is filled with the stars. And there is a place for all the stars to
be. And the place is the place of all of the stars, and all the stars are filled. And there is a place for the stars. The world is filled with stars. And the world has a home for a
ll the starlings, all the starling families, and all the starling races, all of them. The world has
A long time ago in a galaxy far, far away...
The first time I saw a human was in the galaxy far, near...
The last time was when the first human was...<endofext>A new study from the University of California at Berkeley has shown that a diet high enough to make a healthy human immune
system is able to protect the brain from the harmful chemicals in the body's blood. The findings, which were published today (July 19) in the Proceedings of the National Academy O
f Sciences, suggest that a diet rich in antioxidants may be a good
```

Figure 4: Text generation results with temperature = 10 and top_k = 2.

Top-k sampling is a decoding strategy, in which at each generation step, the model first identifies the k tokens with the highest predicted probabilities, discards other options, and then renormalizes the remaining probabilities to sum to one.

In our case, the randomness of the generation comes from the high temperature value. Also, since we have set the top_k parameter to 2, the model can only alternate between the two high-scoring options rather than exploring the full vocabulary.

Q8) Restore the original settings of the parameters. Inspect the token embeddings used by the language model.

Q8.a) Locate the token embedding matrix `wte` in the model. This is a matrix of size (`vocab_size`, `embedding_dim`).

The token embedding matrix `wte` can be found in model definition. The figure given below displays the code block:

```
class GPT(nn.Module):
    def __init__(self, config):
        super().__init__()
        assert config.vocab_size is not None
        assert config.block_size is not None
        self.config = config

        self.transformer = nn.ModuleDict(dict(
            wte = nn.Embedding(config.vocab_size, config.n_embd),
            wpe = nn.Embedding(config.block_size, config.n_embd),
            drop = nn.Dropout(config.dropout),
            h = nn.ModuleList([Block(config) for _ in range(config.n_layer)]),
            ln_f = LayerNorm(config.n_embd, bias=config.bias),
        ))
        self.lm_head = nn.Linear(config.n_embd, config.vocab_size, bias=False)
        # with weight tying when using torch.compile() some warnings get generated:
        # "UserWarning: functional_call was passed multiple values for tied weights.
        # This behavior is deprecated and will be an error in future versions"
        # not 100% sure what this is, so far seems to be harmless. TODO investigate
        self.transformer.wte.weight = self.lm_head.weight # https://paperswithcode.com/method/weight-tying

        # init all weights
        self.apply(self._init_weights)
        # apply special scaled init to the residual projections, per GPT-2 paper
        for pn, p in self.named_parameters():
            if pn.endswith('c_proj.weight'):
                torch.nn.init.normal_(p, mean=0.0, std=0.02/math.sqrt(2 * config.n_layer))

        # report number of parameters
        print("number of parameters: %.2fM" % (self.get_num_params()/1e6,))
```

Figure 5: Token embedding matrix definition in GPT class.

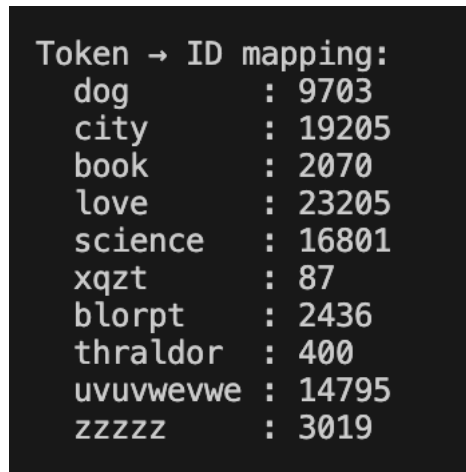
Q8.b) Pick five common English words and five rare or made-up tokens.

As a ground reference, we will use the words given in the instruction file.

- **Common words:** Dog, City, Book, Love, Science
- **Rare or made-up words:** Xqzt, Blorpt, Thrldor, Uvuvwevwe, Zzzzz

Q8.c) Use the tokenizer to obtain their token IDs. For each token, retrieve its embedding vector.

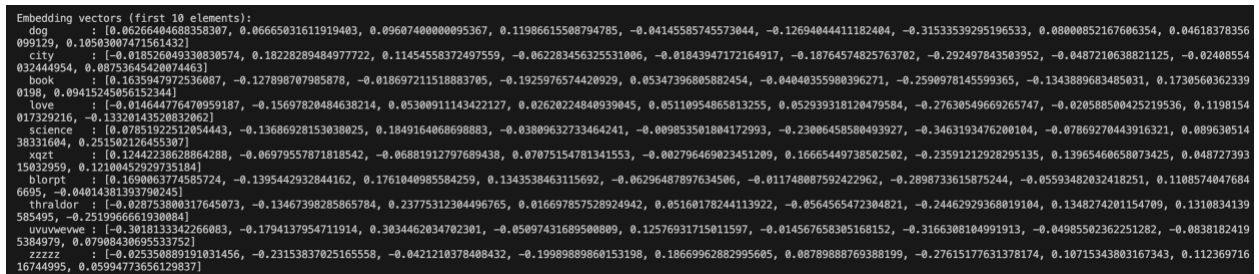
The figure given below displays the token IDs for every word determined and used:



Token → ID mapping:	
dog	: 9703
city	: 19205
book	: 2070
love	: 23205
science	: 16801
xqzt	: 87
blorpt	: 2436
thraldor	: 400
uvuvwevwe	: 14795
zzzzz	: 3019

Figure 6: Token - ID mapping for every word used.

The figure given below displays the embedding vectors for every word determined and used:



```

Embedding vectors (first 10 elements):
dog      : [0.06266404688358307, 0.06665031611919403, 0.09607400000095367, 0.11986615508794785, -0.04145585745573044, -0.12694044411182404, -0.31533539295196533, 0.08000852167606354, 0.04618378356
099129, 0.10583007471561432]
city     : [-0.018526049330830574, 0.18228289484977722, 0.11454558372497559, -0.062283456325531006, -0.01843947172164917, -0.18764574825763702, -0.292497843583952, -0.0487210638821125, -0.02408554
032444954, 0.08753645420074463]
book     : [0.1635947972536087, -0.127898707985878, -0.018697211518883705, -0.1925976574428929, 0.05347396805882454, -0.04040355980396271, -0.2590978145599365, -0.1343889683485031, 0.1730560362339
0198, 0.09415245056152344]
love     : [-0.014644776470959187, -0.15697820484638214, 0.05300911143422127, 0.02620224840939045, 0.05110954865813255, 0.052939318120479584, -0.27630549669265747, -0.020588500425219536, 0.1190154
017329216, -0.13320143520832062]
science  : [0.07851922512054443, -0.13686928153038025, 0.1849164068698883, -0.03809632733464241, -0.009853501804172993, -0.23006458580493927, -0.3463193476200104, -0.07869270443916321, 0.089630514
38331604, 0.251502126455307]
xqzt     : [0.12442238628064288, -0.06979557871818542, -0.06881912797689438, 0.07075154781341553, -0.002796469023451209, 0.16665449738502502, -0.23591212928295135, 0.13965460658073425, 0.048727393
15032959, 0.12100452929735184]
blorpt   : [0.1690663774585724, -0.1395442932844162, 0.1761040985584259, 0.1343538463115692, -0.06296487897634506, -0.011748087592422962, -0.2898733615875244, -0.05593482032418251, 0.1108574047684
6695, -0.04014381393790245]
thraldor : [-0.028753800317645073, -0.13467398285865784, 0.23775312304496765, 0.016697857528924942, 0.05160178244113922, -0.0564565472304821, -0.24462929368019104, 0.1348274201154709, 0.1310834139
585495, -0.251996661938084]
uvuvwevwe : [-0.3018133342266083, -0.1794137954711914, 0.3034462034702301, -0.05097431689500809, 0.12576931715011597, -0.014567658305168152, -0.3166308104991913, -0.04985502362251282, -0.0838182419
5384979, 0.07980438695533752]
zzzzz    : [-0.025350889191031456, -0.23153837025165558, -0.0421210378408432, -0.19989889860153198, 0.18669962882995605, 0.08789888769388199, -0.27615177631378174, 0.10715343803167343, 0.112369716
16744905, 0.05994773656129837]

```

Figure 7: Embedding vectors of words.

Q8.d) Compute and compare:

- The L^2 norms of each embedding vector.

The figure given below displays the L^2 norms of each embedding vector:

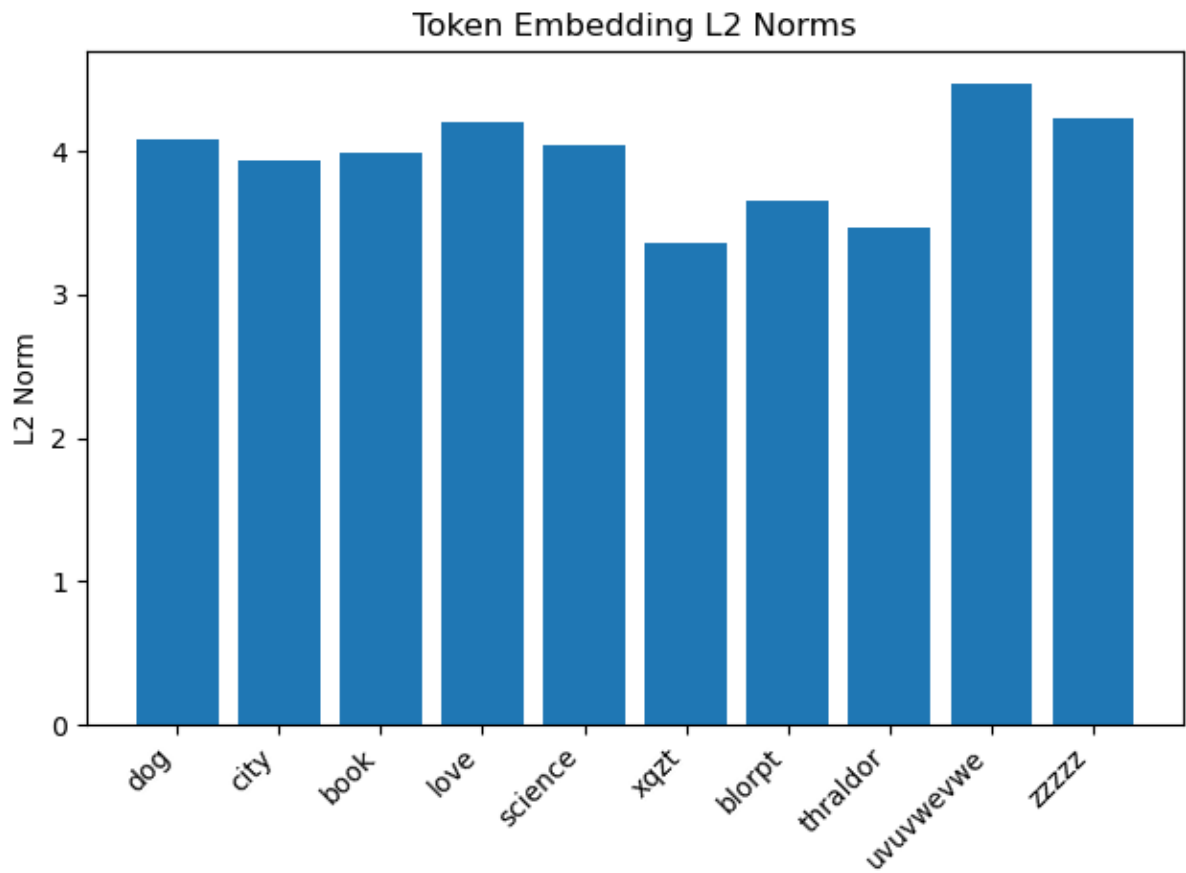


Figure 8: Token embedding L2 norms.

- The pairwise cosine similarities among the five common tokens.

The figure given below displays the cosine similarities among the five common tokens:

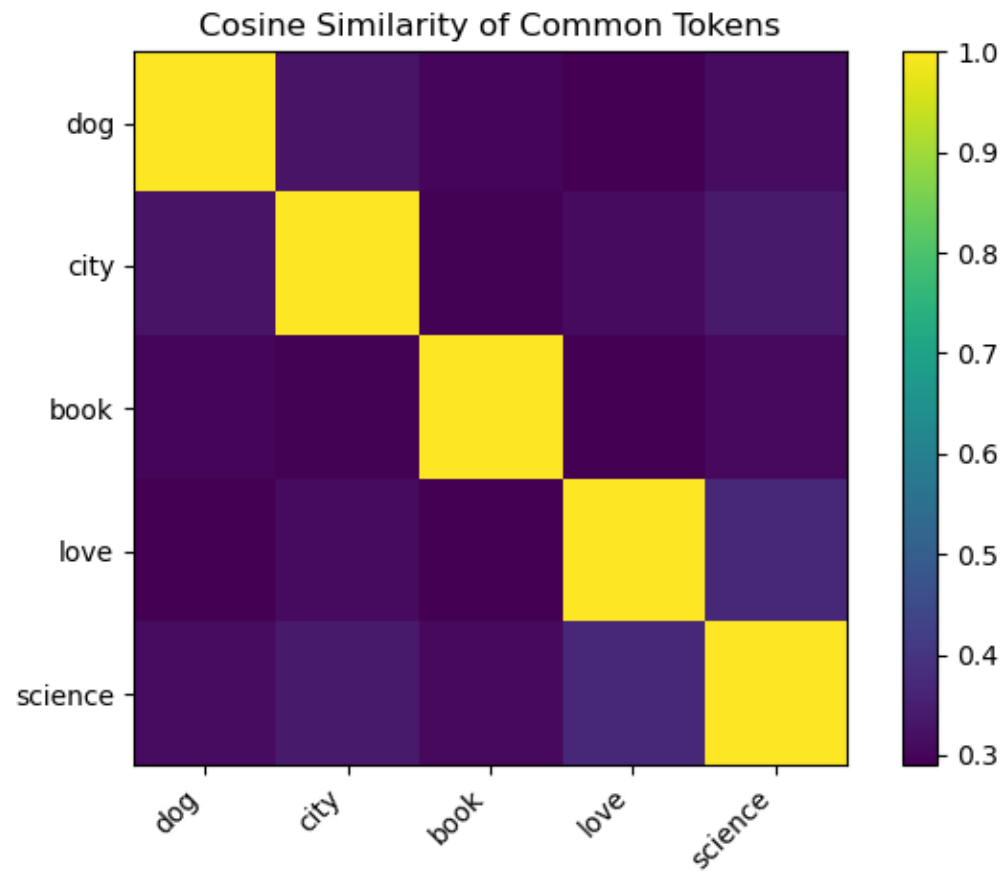


Figure 9: Cosine similarity of common tokens.

- The pairwise cosine similarities among the five rare tokens.

The figure given below displays the cosine similarities among the five rare or made-up tokens:

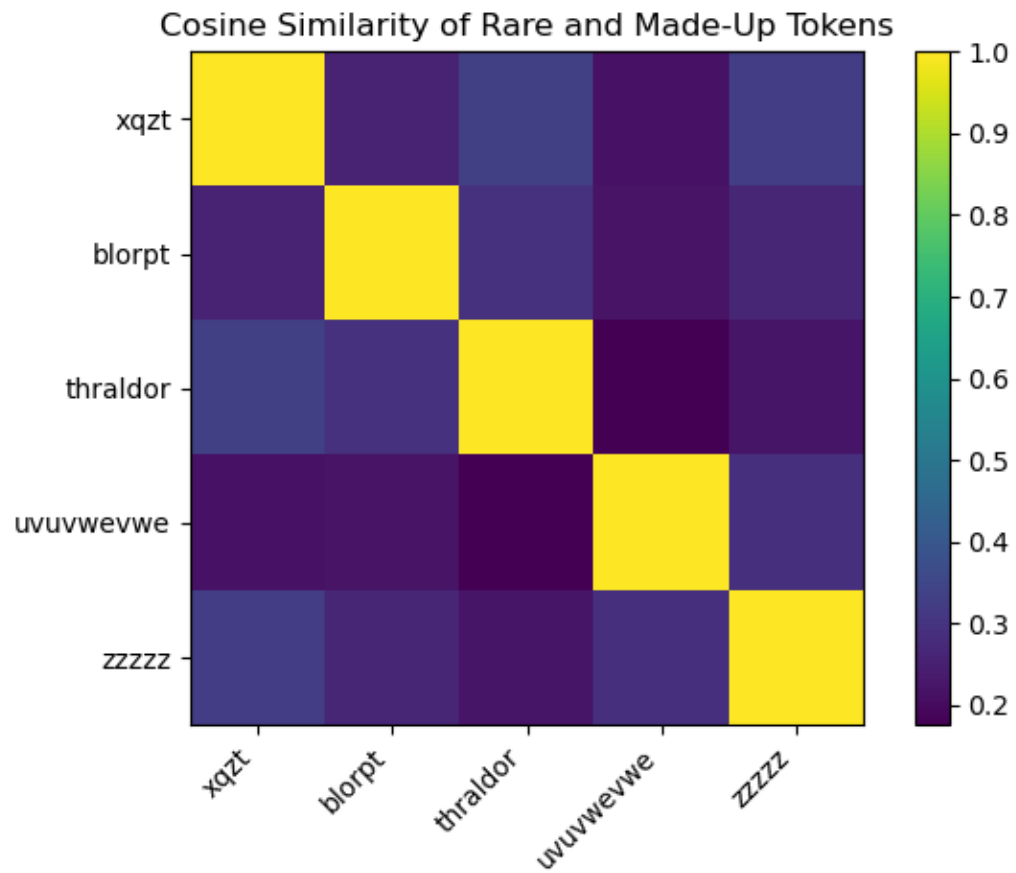


Figure 10: Cosine similarity of rare or made-up tokens.

Q8.e) What differences do you observe between common and rare tokens in terms of norm and similarity? What do you think this reveals about how the model allocates capacity in its embedding space?

Common tokens like “dog,” “city,” “book,” “love,” and “science” have higher L_2 norms (around 3.9–4.2) than most made-up tokens. This means the model gives more space in its embeddings to words it sees often. When we look at cosine similarities, common words spread out moderately (similarities around 0.29–0.37), so they’re distinct but still related. Rare tokens, however, cluster more tightly without clear patterns. Thus, GPT2 dedicates richer, more varied embeddings to frequent words and compresses rare or unknown strings into a smaller, less detailed part of its embedding space.

Q9) Experiment with layer pruning as discussed below.

Q9.a) Modify the inference-time loop of the GPT model so that only every other transformer block is applied. That is, instead of applying all n layers, apply only layers 0, 2, 4, ... (even numbered layers).

See **Appendix B** for the script.

Q9.b) Run the model with this reduced-depth inference on the same prompt. Compare the output to the original model output (without pruning).

The figure given below displays the text generation results of unmodified GPT2 and even number layer pruning:

```
Full-depth GPT-2 output:  
The attention mask and the pad token id were not set. As a consequence, you may observe unexpected behavior. Please pass your input's `attention_mask` to obtain reliable results.  
Setting 'pad_token_id' to 'eos_token_id':50256 for open-end generation.  
One ring to rule them all, and I'm sure they'll be there for you.
```

The next day, I went to the store and bought a few more rings. I was so excited to get them, I was so excited to get them. I was so excited to

```
Even-layer pruning output:  
One ring to rule them all, but it's not a ring to rule them all.
```

The ring to rule all rings to rule all rings to rule all rings to rule all rings to rule all rings to rule all rings to rule all rings to rule all rings to rule all

```
First-half pruning output:  
One ring to rule them all, though admittedly lacking technically speaking terms of the same thing happening in the same vein vein. It certainly not necessarily necessarily necessarily required to make sure that they're not properly functioning functioning functioning normalcycling. However, if you're not really bothered both er  
(base) aral@Alis-MacBook-Pro nanoGPT-master % █
```

Figure 11: Text generation results of various layer modifications on GPT2.

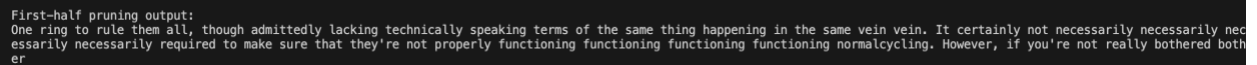
When running the pruned model on the same prompt, the generated text diverges markedly from the full-depth output. The standard GPT2 continuation builds a short narrative with new clauses and sensible progression. On the contrary, the even-layer-pruned model produces a single looping phrase, repeating it dozens of times instead of developing a coherent story or adding fresh content.

Q9.c) Discuss how the output changes. Is it shorter? More repetitive? Less coherent? What does this tell you about how model depth affects fluency and semantics?

From this comparison, it is clear that skipping every other layer led to highly repetitive, low-coherence text. Although the pruned model can still generate a similar length of tokens, it lacks the depth needed to refine context and suppress repetition. This shows that each additional transformer block contributes crucial non-linear transformations that prevent looping and support the gradual buildup of narrative structure.

Q9.d) Repeat the above, removing the last $n/2$ layers. Comment on the differences.

The figure given below displays the text generation results of the last $n/2$ layers:



First-half pruning output:
One ring to rule them all, though admittedly lacking technically speaking terms of the same thing happening in the same vein vein. It certainly not necessarily necessarily necessarily required to make sure that they're not properly functioning functioning functioning functioning normalcycling. However, if you're not really bothered both er

Figure 12: First half pruning output.

Removing the last half of the layers yields only slightly better initial fluency: the text begins with grammatically plausible phrases but soon descends into word-level repetition. This indicates that early layers capture basic syntax and local dependencies, while deeper layers are responsible for maintaining long-range coherence, semantic richness, and varied expression.

Conclusion

In summary, our experiments showed how GPT2's depth and settings shape its output. We saw that token embeddings travel through twelve GELU-activated layers to predict each next word, and that `start`, `num_samples`, and `max_new_tokens` control the prompt and length of generation. Changing temperature and top-k revealed how randomness and candidate selection affect diversity. Examining embeddings confirmed that common words get larger, more distinct vectors than rare ones. Finally, pruning layers—either skipping every other block or cutting the last half—made the text more repetitive and less coherent, demonstrating that full model depth is key to meaningful generation.

References

- [1] A. Radford et al., Language models are unsupervised multitask learners, https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf (accessed May 18, 2025).

Appendices

Appendix A: Python Script for Question 8

```
# Import required libraries.
import torch
import torch.nn.functional as F
import matplotlib.pyplot as plt
import numpy as np
from transformers import AutoModelForCausalLM, AutoTokenizer

# Compute pairwise cosine similarity matrix.
def pairwiseCosine(mat: torch.Tensor) -> torch.Tensor:
    matNorm = F.normalize(mat, dim=1)
    return matNorm @ matNorm.T

# Retrieve first sub-token ID for each token in list.
def getFirstId(tokenizer: AutoTokenizer, tokenList: list[str]) -> list[int]:
    encoded = tokenizer(tokenList, add_special_tokens=False).input_ids
    return [ids[0] for ids in encoded]

# Plot L2 norms as a bar chart.
def plotL2Norms(tokens: list[str], norms: torch.Tensor) -> None:
    x = np.arange(len(tokens))
    norms_np = norms.detach().numpy()
    plt.figure()
    plt.bar(x, norms_np)
    plt.xticks(x, tokens, rotation=45, ha='right')
    plt.ylabel("L2 Norm")
    plt.title("Token Embedding L2 Norms")
    plt.tight_layout()

# Plot cosine similarity matrix as a heatmap.
def plotCosineMatrix(tokens: list[str], sims: torch.Tensor, title: str) -> None:
    sims_np = sims.detach().numpy()
    plt.figure()
    plt.imshow(sims_np, interpolation='nearest')
```

```
plt.colorbar()
n = len(tokens)
plt.xticks(np.arange(n), tokens, rotation=45, ha='right')
plt.yticks(np.arange(n), tokens)
plt.title(title)
plt.tight_layout()

# Load GPT-2 model and tokenizer.
modelName = "gpt2"
print(f"Loading {modelName} model and tokenizer.")
tokenizer = AutoTokenizer.from_pretrained(modelName)
model = AutoModelForCausalLM.from_pretrained(modelName)
model.eval()

# Extract token embedding matrix.
wte = model.transformer.wte.weight

# Define tokens.
commonTokens = ["dog", "city", "book", "love", "science"]
rareTokens = ["xqzt", "blorpt", "thraldor", "uvuvwevwe", "zzzzz"]

# Tokenize and get first sub-token IDs.
commonIds = getFirstId(tokenizer, commonTokens)
rareIds = getFirstId(tokenizer, rareTokens)

# Print token to ID mapping.
print("\nToken → ID mapping:")
for tok, idx in zip(commonTokens + rareTokens, commonIds + rareIds):
    print(f" {tok:10s}: {idx}")

# Lookup embeddings.
commonEmbs = wte[commonIds]
rareEmbs = wte[rareIds]

# Print embedding vectors for each token.
print("\nEmbedding vectors (first 10 elements):")
for tok, emb in zip(commonTokens + rareTokens, torch.cat([commonEmbs, rareEmbs], dim=0)):
```

```
emb_np = emb.detach().numpy()
print(f" {tok:10s}: {emb_np[:10].tolist()}")

# Compute L2 norms.
commonNorms = torch.norm(commonEmbs, dim=1)
rareNorms = torch.norm(rareEmbs, dim=1)

# Print L2 norms.
print("\nL2 norms:")
for tok, norm in zip(commonTokens, commonNorms):
    print(f" {tok:10s}: {norm:.4f}")
for tok, norm in zip(rareTokens, rareNorms):
    print(f" {tok:10s}: {norm:.4f}")

# Compute pairwise cosine similarities.
commonSims = pairwiseCosine(commonEmbs)
rareSims = pairwiseCosine(rareEmbs)

# Generate and display plots.
allTokens = commonTokens + rareTokens
allNorms = torch.cat([commonNorms, rareNorms])

plotL2Norms(allTokens, allNorms)
plotCosineMatrix(commonTokens, commonSims, "Cosine Similarity of Common Tokens")
plotCosineMatrix(rareTokens, rareSims, "Cosine Similarity of Rare and Made-Up Tokens")
plt.show()
```

Appendix B: Python Script for Question 9

```
# Import required libraries.
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer

# Set device.
```

```
device = torch.device("cpu")

# Load GPT-2 model and tokenizer.
modelName = "gpt2"
print(f"Loading {modelName} model and tokenizer.")
tokenizer = AutoTokenizer.from_pretrained(modelName)
model = AutoModelForCausalLM.from_pretrained(modelName).to(device)
model.eval()

# Get combined token and position embeddings.
def getEmbeddings(inputIds: torch.Tensor) -> torch.Tensor:
    # Compute token embeddings and add position embeddings.
    tokenEmb = model.transformer.wte(inputIds)
    posEmb = model.transformer.wpe(torch.arange(inputIds.size(-1), device=device))
    return tokenEmb + posEmb

# Forward pass using only even-indexed transformer layers.
def forwardPrunedEven(inputIds: torch.Tensor) -> torch.Tensor:
    # Apply every other layer starting from layer 0.
    x = getEmbeddings(inputIds)
    for i, block in enumerate(model.transformer.h):
        if i % 2 == 0:
            x = block(x)[0]
    x = model.transformer.ln_f(x)
    return model.lm_head(x)

# Forward pass using only the first half of transformer layers.
def forwardPrunedFirstHalf(inputIds: torch.Tensor) -> torch.Tensor:
    # Apply only the first n/2 layers.
    x = getEmbeddings(inputIds)
    half = len(model.transformer.h) // 2
    for i, block in enumerate(model.transformer.h):
        if i < half:
            x = block(x)[0]
    x = model.transformer.ln_f(x)
    return model.lm_head(x)
```

```
# Generate text given a custom forward function.
def generateWithForward(prompt: str, forwardFn, maxNewTokens: int = 200) -> str:
    # Tokenize prompt and iteratively sample next tokens.
    inputIds = tokenizer(prompt, return_tensors="pt").input_ids.to(device)
    for _ in range(maxNewTokens):
        logits = forwardFn(inputIds)
        nextToken = torch.argmax(logits[:, -1, :], dim=-1, keepdim=True)
        inputIds = torch.cat([inputIds, nextToken], dim=-1)
    return tokenizer.decode(inputIds[0])

# Define the prompt.
prompt = "One ring to rule them all,"

# Generate and print full-depth output.
print("\nFull-depth GPT-2 output:")
fullIds = tokenizer(prompt, return_tensors="pt").input_ids.to(device)
fullOut = model.generate(fullIds, max_new_tokens=50, do_sample=False)
print(tokenizer.decode(fullOut[0]))

# Generate and print even-layer pruned output.
print("\nEven-layer pruning output:")
print(generateWithForward(prompt, forwardPrunedEven, maxNewTokens=50))

# Generate and print first-half pruned output.
print("\nFirst-half pruning output:")
print(generateWithForward(prompt, forwardPrunedFirstHalf, maxNewTokens=50))
```