ECE 60146 - Homework 10

Selin Kasap

1 Task 1: Tokenization

1.1 Training a BPE Tokenizer

Listing 1: Script used to train tokenizer [1].

```
import sys
sys.path.insert(0, "/tmp/bg_src/babyGPT-1.1.4")

from babyGPT import babyGPT
TrainTokenizer = babyGPT.TrainTokenizer

tokenizer_trainer = TrainTokenizer(
    corpus_directory="./saved_articles_dir_12M",
    target_vocab_size=10000
)

tokenizer_trainer.train_tokenizer()
```

The actual vocabulary size in the trained tokenizer JSON is:

10002

This is slightly above the target size of 10,000 because the tokenizer includes additional bookkeeping and special-token related entries in the final JSON representation.

To better understand the learned vocabulary, I tested a set of candidate words to see whether the tokenizer kept them as single tokens or split them into subwords. The tokenizer kept the following words whole:

- `president` → [`president`]
- `government` → [`government`]
- `player` → [`player`]

The tokenizer split the following words into subwords:

- `election` → [`electi`, `on`]
- `market` → [`mar`, `ket`]
- `economy` → [`econo`, `m`, `y`]

This behavior is consistent with how BPE works. Words that occur frequently in the training corpus are more likely to be merged into complete single tokens. In contrast, words that are less frequent, morphologically variable, or less dominant in the corpus are more likely to remain split into smaller pieces. Since the training corpus was based on a relatively small news collection, it is reasonable that common news-domain words like `president`, `government`, and `player` were learned as whole tokens, while words such as `economy` and `market` were not fully merged. The behavior also reflects the fact that BPE learns frequent character or subword patterns rather than true linguistic word boundaries.

1.2 Comparing Tokenizers Across Domains

For this part, I compared three tokenization methods across three sentence groups from different domains:

1. whitespace tokenization using `split()`
2. BERT WordPiece tokenization using `DistilBertTokenizer`
3. babyGPT BPE tokenization using the pretrained `112_babygpt_tokenizer_50002.json`

The three sentence groups were:

- sports news
- scientific text
- informal internet text

The sports-news sentences were in-domain for babyGPT's pretrained tokenizer, since that tokenizer was trained on athlete-news data. The scientific and informal internet sentences were out-of-domain to different degrees.

Tables 1, 2, and 3 show the token counts and representative token outputs for each sentence group.

Sentence [2]	Whitespace	BERT	babyGPT	BERT tokens	babyGPT tokens
The lack of quarterback depth beyond Mendoza complicates projections. Typically, the first 10 picks are clearer weeks in advance. This year, sources said there is significant uncertainty including for the Giants.	32	40	62	[the, lack, of, quarterback, depth, beyond, mendoza, com, ##pl, ##icate, ##s, projections, ., typically, ,, ...]	[The, lack, of, quarterback, depth, beyond, Men, doz, a, complicates, pro, je, c, ti, on, s, ., Ty, pic, all, y, ,, ...]
Cornerback Jermod McCoy is one of them. Four different teams told SNY they flagged his knee, which he tore in January. One team told SNY they have taken him off their board entirely. Miami offensive tackle Mauigoa might slip some, too.	41	54	69	[cornerback, je, ##rm, ##od, mccoy, is, one, of, them, ., four, different, teams, told, s, ##ny, ...]	[Co, rn, er, back, Jer, mo, d, McCo, y, is, one, of, them, ., Four, different, team, s, tol, d, ...]
Tyson, even with the injuries, is flying up draft boards. His talent, and the lack of any true receivers with his physical ability, is infatuating to teams.	27	35	44	[tyson, ,, even, with, the, injuries, ,, is, flying, up, draft, boards, ., his, talent, ,, and, the, lack, of, ...]	[Ty, son, ,, even, with, the, injuries, ,, is, flying, up, draft, board, s, ., Hi, s, talent, ,, and, ...]

Table 1: Tokenization comparison for the sports-news sentence group. babyGPT produced an average of 58.33 tokens, BERT produced 43.00, and whitespace tokenization produced 33.33.

Sentence [3]	Whitespace	BERT	babyGPT	BERT tokens	babyGPT tokens
Scientists have discovered that moringa seeds can help pull microplastics out of water, rivaling standard chemical treatments. The plant-based extract causes plastic particles to clump together, making them easier to filter away.	32	44	65	[scientists, have, discovered, that, mori, ##nga, seeds, can, help, pull, micro, ##pl, ##astic, ##s, out, of, ...]	[Sc, i, enti, st, s, have, discove, red, that, mo, rin, ga, seed, s, can, help, pul, l, micro, pla, ...]
We showed that the saline extract from the seeds performs similarly to aluminum sulfate, which is used in treatment plants to coagulate water containing microplastics. In more alkaline waters, it performed even better than the chemical product.	37	48	74	[we, showed, that, the, saline, extract, from, the, seeds, performs, similarly, to, aluminum, sulfate, ,, which, ...]	[We, sho, wed, that, the, sa, line, extract, from, the, seed, s, performs, si, mi, lar, l, y, to, al, ...]
Coagulation plays a key role because microplastics and other contaminants carry a negative electrical charge. This causes them to repel each other and prevents them from being captured easily during filtration.	31	44	63	[coa, ##gul, ##ation, plays, a, key, role, because, micro, ##pl, ##astic, ##s, and, other, con, ##tam, ...]	[Coa, gu, la, ti, on, plays, a, key, role, because, micro, pla, stic, s, and, other, contamin, ant, s, ...]

Table 2: Tokenization comparison for the scientific/medical sentence group. babyGPT produced an average of 67.33 tokens, BERT produced 45.33, and whitespace tokenization produced 33.33.

Sentence [4]	Whitespace	BERT	babyGPT	BERT tokens	babyGPT tokens
I say “bruh” and “bro”, as well as “dude”. I also say YEET but that’s a wrestling thing. Let’s see sketchy/sus, cool, far out, sweet, tea, and that’s really it for those I use unironically.	37	75	85	[i, say, ‘‘, br, ##uh, ’’, and, ‘‘, bro, ’’, ,, as, well, as, ‘‘, dude, ’’, ., ...]	[I, say, ‘‘, br, u, h, ’’, and, ‘‘, bro, ’’, ,, as, well, as, ‘‘, dude, ’’, ., I, also, say, ...]
I grew up in North Jersey with New York 20 mins in one direction and the del water gap a half hour in the other. My upbringing was unique to say the least and I’ve been saying deadass since I was 14 :’D.	43	48	53	[i, grew, up, in, north, jersey, with, new, york, 20, min, ##s, in, one, direction, and, ...]	[I, gre, w, up, in, North, Jersey, with, New, York, 20, mins, in, one, directi, on, and, the, ...]
I started saying yeet as a joke and now I use it all the time. If yeeting is wrong I don’t want to be right. It’s the kids fault. Sweet, chill, deadass, low key, extra, that’s bomb, I can’t, fam, cuz, on point, that’s boot, that’s a vibe. I’m basic bro.	52	88	96	[i, started, saying, ye, ##et, as, a, joke, and, now, i, use, it, all, the, time, ., if, ye, ##eti, ...]	[I, started, saying, ye, et, as, a, joke, and, now, I, use, it, all, the, time, ., I, f, ye, e, tin, ...]

Table 3: Tokenization comparison for the informal internet sentence group. babyGPT produced an average of 78.00 tokens, BERT produced 70.33, and whitespace tokenization produced 44.00.

Which sentence group gave babyGPT the fewest tokens? Among the three groups, babyGPT produced the fewest average tokens on the sports-news group, with an average of 58.33 tokens. This is exactly what I would expect based on the tokenizer’s training domain. The pretrained babyGPT tokenizer was trained on an athlete-news corpus, so it had more chances to learn and merge frequent subword patterns from sports reporting. In-domain text is therefore more likely to align with its learned vocabulary and produce fewer subword splits. In contrast, the scientific and informal groups contained vocabulary and styles that were less similar to the training corpus, so more splitting was necessary.

Effect of BERT lowercasing all input. BERT’s uncased tokenizer lowercases all text before tokenization. This has an important consequence: it removes distinctions carried by capitalization. For example, `US` and `us` become the same surface form after lowercasing, even though one may refer to a country and the other to a pronoun. Likewise, capitalization in named entities, abbreviations, or scientific notation may carry meaning that is lost when the text is normalized. This can make the model less sensitive to cues that would help with tasks such as named entity recognition, acronym disambiguation, or proper-noun identification.

At the same time, lowercasing reduces vocabulary sparsity. It makes the tokenizer and the model more robust when the same word appears in multiple case forms. So lowercasing is a trade-off: it simplifies the vocabulary and often helps generalization, but it also removes potentially meaningful information.

Why fixed vocabulary size matters for autoregressive language modeling. A fixed vocabulary size is necessary because the final output layer of an autoregressive language model must assign a probability to every possible next token. In practice, this means the model’s final linear

layer and softmax layer must have one output dimension for each token in the vocabulary. If the vocabulary were unbounded and kept growing with the corpus, then the size of this output layer would also keep changing, making stable training impossible.

Tokenization solves this problem by mapping open-ended text into a fixed inventory of reusable tokens. The model can then learn a probability distribution over that fixed token set. This is the connection between vocabulary size and the final softmax layer: the size of the vocabulary directly determines the dimensionality of the next-token prediction space. Without a fixed vocabulary, maximum-likelihood prediction of the next token would not be practical.

2 Task 2: Probing Pretrained BERT

2.1 Masked Language Modeling: What Does BERT Know?

For this part, I used a pretrained `bert-base-uncased` masked-language model and probed it with 10 masked sentences. The goal was to test different kinds of knowledge that BERT may have learned during pretraining, using the four categories: factual knowledge, syntactic knowledge, commonsense/associative reasoning, and adversarial or tricky examples. For each sentence, I recorded the top-5 predictions returned by the model and checked whether the intended correct answer appeared among them.

Listing 2: MLM probing script.

```python
from __future__ import annotations
from pathlib import Path
from typing import List, Dict, Any
from transformers import pipeline

OUTPUT_DIR = Path("./task2_outputs")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)
MODEL_NAME = "bert-base-uncased"


def build_masked_sentences() -> List[Dict[str, str]]:
    return [
        # factual knowledge
        {"category": "factual",
         "sentence": "The capital of France is [MASK].",
         "correct_answer": "paris",
         },
        {"category": "factual",
         "sentence": "The chemical symbol for gold is [MASK].",
         "correct_answer": "au",
         },
        {"category": "factual",
         "sentence": "The largest planet in our solar system is [MASK].",
         "correct_answer": "jupiter",
        },
        # syntactic knowledge
        {"category": "syntactic",
         "sentence": "The dogs in the park [MASK] chasing a squirrel.",
         "correct_answer": "are",
        },
        {"category": "syntactic",
```

```
        "sentence": "After Maria finished her homework, she [MASK] to bed.",
        "correct_answer": "went",
      },
      {"category": "syntactic",
       "sentence": "The book on the table belongs to John, and [MASK] cover is red.",
       "correct_answer": "its",
      },
      # commonsense
      {"category": "commonsense",
       "sentence": "She put on her coat because it was [MASK] outside.",
       "correct_answer": "cold",
      },
      {"category": "commonsense",
       "sentence": "He was tired, so he went to [MASK] early.",
       "correct_answer": "bed",
      },
      # adversarial
      {"category": "adversarial",
       "sentence": "The new meme was so [MASK] that nobody in 2017 would have understood it.",
       "correct_answer": "viral",
      },
      {"category": "adversarial",
       "sentence": "The influencer said the launch was absolutely [MASK], no cap.",
       "correct_answer": "fire",
      },
    ]


# ----------------------------
# MLM PROBING
# ----------------------------

def normalize_token(token: str) -> str:
    """
    Normalize returned token text for simple comparison.
    """
    return token.strip().lower()


def run_mlm_probe() -> List[Dict[str, Any]]:
    mlm = pipeline("fill-mask", model=MODEL_NAME)

    examples = build_masked_sentences()
    results_table: List[Dict[str, Any]] = []

    for i, ex in enumerate(examples, start=1):
        sentence = ex["sentence"]
        category = ex["category"]
        correct_answer = normalize_token(ex["correct_answer"])

```

```
        preds = mlm(sentence)
        top5_tokens = [normalize_token(p["token_str"]) for p in preds[:5]]
        top5_scores = [float(p["score"]) for p in preds[:5]]

        correct_in_top5 = correct_answer in top5_tokens

        results_table.append({
            "index": i,
            "category": category,
            "sentence": sentence,
            "top5_predictions": top5_tokens,
            "top5_scores": top5_scores,
            "correct_answer": correct_answer,
            "correct_in_top5": "yes" if correct_in_top5 else "no",
        })

        print("\n" + "=" * 80)
        print(f"Example {i}")
        print("Category:", category)
        print("Sentence:", sentence)
        print("Correct answer:", correct_answer)
        print("Top-5 predictions:")
        for tok, score in zip(top5_tokens, top5_scores):
            print(f"  {tok:15s} {score:.4f}")
        print("Correct in top-5?:", "yes" if correct_in_top5 else "no")

    return results_table


def print_summary(results: List[Dict[str, Any]]) -> None:
    total = len(results)
    correct = sum(1 for r in results if r["correct_in_top5"] == "yes")
    print("\n" + "=" * 80)
    print("OVERALL SUMMARY")
    print(f"Correct in top-5: {correct}/{total}")

    by_category: Dict[str, List[Dict[str, Any]]] = {}
    for r in results:
        by_category.setdefault(r["category"], []).append(r)

    for cat, rows in by_category.items():
        cat_correct = sum(1 for r in rows if r["correct_in_top5"] == "yes"
            )
        print(f"{cat:12s}: {cat_correct}/{len(rows)}")


results = run_mlm_probe()
print_summary(results)
```

#	Sentence	Category	Top-5 predictions	Correct	In top-5?
1	The capital of France is [MASK].	Factual	paris, lille, lyon, marseille, tours	paris	Yes
2	The chemical symbol for gold is [MASK].	Factual	gold, [UNK], k, sigma, c	au	No
3	The largest planet in our solar system is [MASK].	Factual	earth, pluto, jupiter, mars, saturn	jupiter	Yes
4	The dogs in the park [MASK] chasing a squirrel.	Syntactic	were, are, ,, was, .	are	Yes
5	After Maria finished her homework, she [MASK] to bed.	Syntactic	went, headed, returned, walked, retired	went	Yes
6	The book on the table belongs to John, and [MASK] cover is red.	Syntactic	the, its, his, that, her	its	Yes
7	She put on her coat because it was [MASK] outside.	Commonsense	cold, warm, dark, freezing, hot	cold	Yes
8	He was tired, so he went to [MASK] early.	Commonsense	bed, work, sleep, school, class	bed	Yes
9	The new meme was so [MASK] that nobody in 2017 would have understood it.	Adversarial	popular, powerful, new, controversial, offensive	viral	No
10	The influencer said the launch was absolutely [MASK], no cap.	Adversarial	safe, successful, free, normal, ready	fire	No

Table 4: Masked Language Modeling results for 10 probe sentences.

Overall, BERT placed the intended correct answer in the top-5 predictions for 7 out of 10 examples. The breakdown by category was:

- Factual: 2/3
- Syntactic: 3/3
- Commonsense: 2/2
- Adversarial: 0/2

These results suggest that BERT handled **syntactic** and **commonsense** cases best. In the syntactic examples, BERT was very strong. It correctly identified `went` as the best continuation in the sentence about Maria going to bed, and it also placed `its` in the top-5 for the sentence about the book's cover. In the commonsense examples, BERT successfully predicted `cold` for weather-related reasoning and `bed` for a tired person going somewhere early. This indicates that the model learned useful local grammar and many common semantic associations from pretraining.

The factual examples were more mixed. BERT correctly predicted `paris` for the capital of France and included `jupiter` in the top-5 for the largest planet question, although it ranked `earth` and `pluto` above it. However, it failed on the chemical-symbol example. Instead of predicting `au`, it produced `gold` and several unrelated tokens. This failure likely reflects a mismatch between the masked-token objective and the fact that `au` is a short specialized symbol that may not appear often in natural language contexts of the form "The chemical symbol for gold is [MASK]."

The weakest category was clearly the adversarial one. BERT failed both examples. For the meme sentence, the model did not return `viral`; instead it preferred more generic words such as `popular`. For the slang-heavy influencer sentence, it completely missed the intended word `fire` and predicted more formal or neutral continuations like `safe` and `successful`. This is consistent with the fact that BERT was trained on BookCorpus and Wikipedia before 2019, so it is not well adapted to recent internet slang or highly informal social-media style language. These failures are informative because they show both the strengths and the limits of unsupervised pretraining.

2.2 Next Sentence Prediction

For the Next Sentence Prediction (NSP) experiment, I used the pretrained `bert-base-uncased` NSP model. I created 8 sentence pairs in total. The 4 `isNext` examples were built from consecutive sentences taken from the same paragraph of a pasted excerpt from the Wikipedia page on *Twenty One Pilots* [5]. The 4 `notNext` examples were intentionally constructed by pairing a sentence from the *Twenty One Pilots* text as Sentence A with a sentence from a different topic, namely a COVID-19 text excerpt, as Sentence B [6]. This made the false pairs clearly unrelated while keeping the construction process controlled and transparent.

Listing 3: NSP probing script [5] [6].

```
import re
from pathlib import Path
from typing import List, Dict, Any

import torch
from transformers import BertTokenizer, BertForNextSentencePrediction

OUTPUT_DIR = Path("./task2_outputs")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

TOP_TEXT = """
Twenty One Pilots[a] are an American musical duo from Columbus, Ohio. The
    group was originally formed in 2009 by lead vocalist Tyler Joseph along
     with Nick Thomas and Chris Salih, who both left in 2011. Since their
    departure, the line-up has consisted of Joseph and drummer Josh Dun.
    The duo is best known for their singles "Stressed Out", "Ride", and "
    Heathens", which achieved commercial success between 2015 and 2016. "
    Stressed Out" earned them a Grammy Award for Best Pop Duo/Group
    Performance at the 59th Annual Grammy Awards.
"""

COVID_TEXT = """
The global COVID-19 pandemic (also known as the coronavirus pandemic),
    caused by severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2),
     began with an outbreak in Wuhan, China, in December 2019. It spread to
     other parts of Asia and then worldwide in early 2020. The World Health
     Organization (WHO) declared the outbreak a public health emergency of
    international concern (PHEIC) on 30 January 2020 and assessed it as
    having become a pandemic on 11 March. The WHO declared that the public
    health emergency caused by COVID-19 had ended in May 2023.
"""


def split_into_paragraphs(text: str) -> List[str]:
    paras = re.split(r"\n\s*\n", text.strip())
    return [p.strip() for p in paras if p.strip()]


def clean_sentence(s: str) -> str:
    s = re.sub(r"\[[^\]]+\]", "", s)
    s = re.sub(r"\s+", " ", s).strip()
    return s
```

```python


def split_paragraph_into_sentences(paragraph: str) -> List[str]:
    raw = re.split(r"(?<=[.!?])\s+", paragraph)
    sentences = []
    for s in raw:
        s = clean_sentence(s)
        if len(s) >= 20:
            sentences.append(s)
    return sentences


def collect_paragraph_sentence_lists(text: str) -> List[List[str]]:
    para_sentences = []
    for p in paragraphs:
        sents = split_paragraph_into_sentences(p)
        if len(sents) >= 2:
            para_sentences.append(sents)
    return para_sentences


def build_isnext_pairs_from_top(para_sentences: List[List[str]]) -> List[
    Dict[str, Any]]:
    true_pairs = []
    for para_idx, sents in enumerate(para_sentences):
        for i in range(len(sents) - 1):
            true_pairs.append({
                "sentence_a": sents[i],
                "sentence_b": sents[i + 1],
                "true_label": "isNext",
                "pair_source": f"TOP paragraph {para_idx}, sentences {i
                    }->{i+1}",
            })

    if len(true_pairs) < 4:
        raise ValueError("Not enough isNext pairs in Twenty One Pilots
            text.")

    return true_pairs[:4]


def build_notnext_pairs(top_para_sentences: List[List[str]],
    covid_para_sentences: List[List[str]]) -> List[Dict[str, Any]]:
    """
    Build 4 notNext pairs where:
    - sentence A comes from Twenty One Pilots
    - sentence B comes from COVID text
    """
    top_candidates = []
    covid_candidates = []

    for para_idx, sents in enumerate(top_para_sentences):
        for sent_idx, sent in enumerate(sents):
            top_candidates.append((para_idx, sent_idx, sent))
```

```
79
80      for para_idx, sents in enumerate(covid_para_sentences):
81          for sent_idx, sent in enumerate(sents):
82              covid_candidates.append((para_idx, sent_idx, sent))
83
84      if len(top_candidates) < 4 or len(covid_candidates) < 4:
85          raise ValueError("Not enough candidate sentences for notNext pairs
                .")
86
87      notnext_pairs = []
88      for i in range(4):
89          top_p, top_s, sent_a = top_candidates[i]
90          covid_p, covid_s, sent_b = covid_candidates[i]
91          notnext_pairs.append({
92              "sentence_a": sent_a,
93              "sentence_b": sent_b,
94              "true_label": "notNext",
95              "pair_source": (
96                  f"Sentence A from TOP paragraph {top_p}, sentence {top_s};
                        "
97                  f"Sentence B from COVID paragraph {covid_p}, sentence {
                        covid_s}"
98              ),
99          })
100
101     return notnext_pairs
102
103
104 def run_nsp_experiment(pairs: List[Dict[str, Any]]) -> List[Dict[str, Any
        ]]:
105     tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
106     model = BertForNextSentencePrediction.from_pretrained("bert-base-
            uncased")
107     model.eval()
108
109     results = []
110
111     with torch.no_grad():
112         for i, pair in enumerate(pairs, start=1):
113             sentence_a = pair["sentence_a"]
114             sentence_b = pair["sentence_b"]
115             true_label = pair["true_label"]
116
117             inputs = tokenizer(sentence_a, sentence_b, return_tensors="pt"
                    )
118             outputs = model(**inputs)
119
120             pred_idx = torch.argmax(outputs.logits, dim=1).item()
121             pred_label = "isNext" if pred_idx == 0 else "notNext"
122
123             results.append({
124                 "index": i,
125                 "sentence_a": sentence_a,
126                 "sentence_b": sentence_b,
```

```
                "true_label": true_label,
                "bert_prediction": pred_label,
                "correct": "yes" if pred_label == true_label else "no",
                "pair_source": pair["pair_source"],
            })

    return results


def print_summary(results: List[Dict[str, Any]]) -> None:
    total = len(results)
    correct = sum(1 for r in results if r["correct"] == "yes")
    print("\n=== NSP SUMMARY ===")
    print(f"Accuracy: {correct}/{total} = {correct/total:.4f}")

    for r in results:
        print("\n" + "-" * 100)
        print(f"Pair {r['index']}")
        print("Source:          ", r["pair_source"])
        print("True label:      ", r["true_label"])
        print("BERT prediction: ", r["bert_prediction"])
        print("Correct?:        ", r["correct"])
        print("Sentence A:", r["sentence_a"])
        print("Sentence B:", r["sentence_b"])


top_para_sentences = collect_paragraph_sentence_lists(TOP_TEXT)
covid_para_sentences = collect_paragraph_sentence_lists(COVID_TEXT)

print("TOP paragraphs with >=2 usable sentences:", len(top_para_sentences)
    )
print("COVID paragraphs with >=2 usable sentences:", len(
    covid_para_sentences))

isnext_pairs = build_isnext_pairs_from_top(top_para_sentences)
notnext_pairs = build_notnext_pairs(top_para_sentences,
    covid_para_sentences)

pairs = isnext_pairs + notnext_pairs
results = run_nsp_experiment(pairs)

print_summary(results)
```

#	Sentence A	Sentence B	True label	BERT prediction	Correct?
1	Twenty One Pilots are an American musical duo from Columbus, Ohio.	The group was originally formed in 2009 by lead vocalist Tyler Joseph along with Nick Thomas and Chris Salih, who both left in 2011.	isNext	isNext	Yes
2	The group was originally formed in 2009 by lead vocalist Tyler Joseph along with Nick Thomas and Chris Salih, who both left in 2011.	Since their departure, the lineup has consisted of Joseph and drummer Josh Dun.	isNext	isNext	Yes
3	Since their departure, the lineup has consisted of Joseph and drummer Josh Dun.	The duo is best known for their singles "Stressed Out", "Ride", and "Heathens", which achieved commercial success between 2015 and 2016.	isNext	isNext	Yes
4	The duo is best known for their singles "Stressed Out", "Ride", and "Heathens", which achieved commercial success between 2015 and 2016.	"Stressed Out" earned them a Grammy Award for Best Pop Duo/Group Performance at the 59th Annual Grammy Awards.	isNext	isNext	Yes
5	Twenty One Pilots are an American musical duo from Columbus, Ohio.	The global COVID-19 pandemic began with an outbreak in Wuhan, China, in December 2019.	notNext	notNext	Yes
6	The group was originally formed in 2009 by lead vocalist Tyler Joseph along with Nick Thomas and Chris Salih, who both left in 2011.	It spread to other parts of Asia and then worldwide in early 2020.	notNext	notNext	Yes
7	Since their departure, the lineup has consisted of Joseph and drummer Josh Dun.	The World Health Organization declared the outbreak a public health emergency of international concern on 30 January 2020 and assessed it as having become a pandemic on 11 March.	notNext	notNext	Yes
8	The duo is best known for their singles "Stressed Out", "Ride", and "Heathens", which achieved commercial success between 2015 and 2016.	The WHO declared that the public health emergency caused by COVID-19 had ended in May 2023.	notNext	notNext	Yes

Table 5: Next Sentence Prediction results for 8 sentence pairs.

BERT achieved perfect NSP accuracy in this experiment:

$$\text{Accuracy} = \frac{8}{8} = 1.00$$

This result is strong, but it is also partly due to the design of the experiment. The `isNext` pairs were very natural consecutive sentences from the same paragraph, so they formed coherent local discourse. The `notNext` pairs were taken from clearly unrelated topics, which made them easy negatives. For example, a sentence about the formation of Twenty One Pilots followed by a sentence about the global COVID-19 pandemic is an obvious topic shift, and BERT correctly labeled such pairs as `notNext`.

The experiment still illustrates an important point: BERT's NSP objective does capture discourse-level coherence to some extent. The model was able to distinguish continuation from non-continuation reliably in these examples. However, because the negative pairs were strongly mismatched in topic, this was not a difficult stress test. A harder NSP evaluation would use false pairs that are more topically similar, so that the distinction between "plausibly related" and "actually next" becomes

more subtle. Still, for this homework, the results show that BERT's pretraining objective for sentence-pair reasoning is functioning as expected.

2.3 Understanding Questions

Generative vs. discriminative training. In discriminative training, the main goal is to model the conditional probability $p(Y \mid X)$. This means the model focuses directly on predicting the output label Y from the input X. For example, in sentiment classification, X could be a review sentence and Y could be the label positive or negative. A purely discriminative model is concerned only with drawing the best boundary between different output classes for the given inputs.

In contrast, generative training models the joint probability $p(X, Y)$, or in language modeling often learns structure related to $p(X)$ itself. This means the model is not only learning how to map inputs to labels, but also learning about the statistical structure of the inputs. In BERT-style pretraining, the model learns patterns of word usage, grammar, sentence structure, and context from very large unlabeled corpora. This helps downstream discriminative tasks because the model begins fine-tuning with a strong internal representation of language. Instead of learning language understanding and the target task at the same time from limited labeled data, it first learns language broadly and then specializes. That is why generative pretraining can improve performance on downstream discriminative tasks such as classification.

MLM vs. autoregressive modeling. BERT uses Masked Language Modeling (MLM), where one or more tokens are masked and the model predicts them using both the left and right context. The main advantage of MLM is that it gives the model bidirectional understanding. For example, if a word appears in the middle of a sentence, BERT can use both the words before it and the words after it to infer what belongs in that position. This often leads to strong contextual representations, which are very useful for tasks like classification, question answering, and sentence-pair reasoning.

GPT-style autoregressive modeling works differently. It predicts each token using only the tokens that come before it. The advantage of this approach is that it naturally supports text generation, because the model can generate one token at a time in a left-to-right manner. MLM is not suitable for text generation because it is trained to fill in missing tokens inside a sentence, not to continue text sequentially from left to right. In other words, BERT is excellent at understanding masked context, while GPT is naturally suited for producing new text token by token.

The [CLS] token in BERT and the class token in ViT. The `[CLS]` token in BERT plays a role very similar to the class token in the Vision Transformer from HW9. In both cases, a special learnable token is inserted into the input sequence before the rest of the data tokens. In BERT, `[CLS]` is placed at the beginning of the tokenized sentence. In the Vision Transformer, the class token is placed at the beginning of the patch sequence. During processing through the Transformer layers, this special token attends to the rest of the input and gradually becomes an aggregated representation of the whole sequence.

The common idea is that this token becomes a compact summary representation used for final prediction. In BERT, the final hidden state of `[CLS]` is used for tasks such as classification or sentence-pair prediction. In ViT, the final embedding of the class token is used for image classification. So although one operates on words and the other on image patches, the mechanism is essentially the same: both use a special token to collect global information from the full sequence and pass that representation to a task-specific output layer.

3 Task 3: Fine-tuning BERT for Sentiment Analysis

3.1 Background: From Pretraining to Fine-tuning

3.2 Fine-tuning DistilBERT on SST-2

For this part, I fine-tuned a pretrained `distilbert-base-uncased` model for 3 epochs on the SST-2 dataset using the HuggingFace implementation of `DistilBertForSequenceClassification`.

Listing 4: Fine-tuning script.

```
from __future__ import annotations
import random
from dataclasses import dataclass
from typing import Dict, List, Tuple
import matplotlib.pyplot as plt
import torch
from datasets import load_dataset
from torch.utils.data import DataLoader
from transformers import (DistilBertForSequenceClassification,
                          DistilBertTokenizer)

MODEL_NAME = "distilbert-base-uncased"
MAX_LENGTH = 128
BATCH_SIZE = 32
NUM_EPOCHS = 3
LEARNING_RATE = 2e-5
WEIGHT_DECAY = 0.01
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")


def tokenize_batch(batch, tokenizer):
    return tokenizer(
        batch["sentence"],
        padding="max_length",
        truncation=True,
        max_length=MAX_LENGTH)


def prepare_sst2(tokenizer): #load and tokenize SST-2
    sst2 = load_dataset("glue", "sst2")

    train_ds = sst2["train"].map(lambda batch: tokenize_batch(batch,
        tokenizer), batched=True)
    val_ds = sst2["validation"].map(lambda batch: tokenize_batch(batch,
        tokenizer), batched=True)

    train_ds.set_format("torch", columns=["input_ids", "attention_mask", "
        label"])
    val_ds.set_format("torch", columns=["input_ids", "attention_mask", "
        label"])

    train_loader = DataLoader(train_ds, batch_size=BATCH_SIZE, shuffle=
        True)
    val_loader = DataLoader(val_ds, batch_size=BATCH_SIZE)
```

```
    return sst2, train_ds, val_ds, train_loader, val_loader


@dataclass
class TrainResults:
    train_loss_per_epoch: List[float]
    val_accuracy_per_epoch: List[float]


def train_one_epoch(model, train_loader, optimizer):
    model.train()
    total_loss = 0.0
    for batch in train_loader:
        input_ids = batch["input_ids"].to(DEVICE)
        attention_mask = batch["attention_mask"].to(DEVICE)
        labels = batch["label"].to(DEVICE)

        outputs = model(
            input_ids=input_ids,
            attention_mask=attention_mask,
            labels=labels)
        loss = outputs.loss

        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
        total_loss += loss.item()

    return total_loss / len(train_loader)


def evaluate_accuracy(model, val_loader):
    model.eval()
    correct = 0
    total = 0

    with torch.no_grad():
        for batch in val_loader:
            input_ids = batch["input_ids"].to(DEVICE)
            attention_mask = batch["attention_mask"].to(DEVICE)
            labels = batch["label"].to(DEVICE)

            outputs = model(input_ids=input_ids, attention_mask=
               attention_mask)
            preds = torch.argmax(outputs.logits, dim=1)

            correct += (preds == labels).sum().item()
            total += labels.size(0)
    return correct / total


def fine_tune_distilbert():
    print("Using device:", DEVICE)
```

```
93
94      tokenizer = DistilBertTokenizer.from_pretrained(MODEL_NAME)
95      model = DistilBertForSequenceClassification.from_pretrained(
96          MODEL_NAME, num_labels=2).to(DEVICE)
97
98      sst2, train_ds, val_ds, train_loader, val_loader = prepare_sst2(
            tokenizer)
99
100     optimizer = torch.optim.AdamW(
101         model.parameters(),
102         lr=LEARNING_RATE,
103         weight_decay=WEIGHT_DECAY)
104
105     train_losses = []
106     val_accuracies = []
107
108     for epoch in range(NUM_EPOCHS):
109         avg_loss = train_one_epoch(model, train_loader, optimizer)
110         val_acc = evaluate_accuracy(model, val_loader)
111
112         train_losses.append(avg_loss)
113         val_accuracies.append(val_acc)
114         print(f"Epoch {epoch + 1}: loss = {avg_loss:.4f}")
115         print(f"Val accuracy after epoch {epoch + 1}: {val_acc:.4f}")
116
117     results = TrainResults(train_loss_per_epoch=train_losses,
118                            val_accuracy_per_epoch=val_accuracies)
119
120     return model, tokenizer, results, sst2
121
122 model, tokenizer, results, sst2 = fine_tune_distilbert()
```

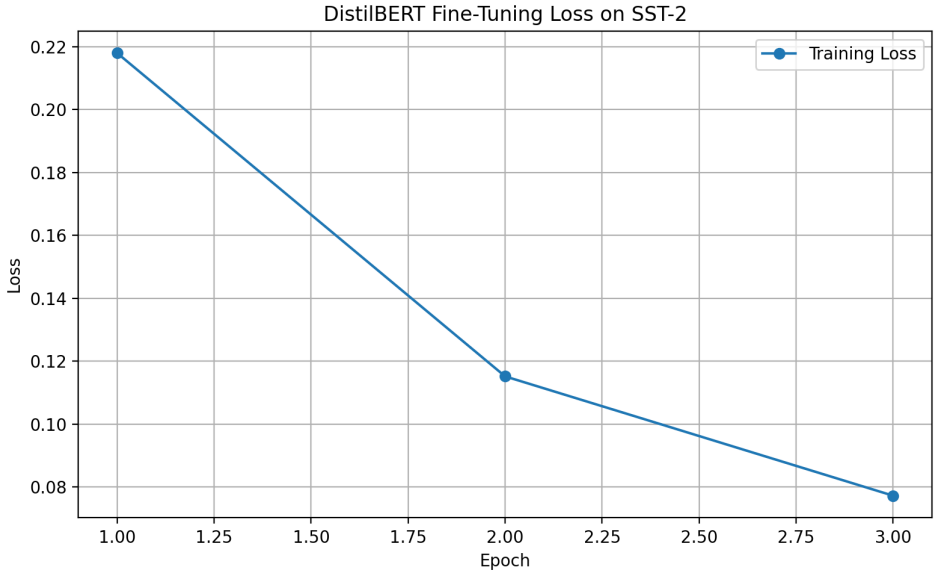

Figure 1: Training loss during fine-tuning of pretrained DistilBERT on SST-2.

The training loss and validation accuracy at the end of each epoch were as follows:

- Epoch 1: loss = 0.2178, validation accuracy = 0.9037
- Epoch 2: loss = 0.1163, validation accuracy = 0.9037
- Epoch 3: loss = 0.0786, validation accuracy = 0.9002

#	Sentence	True Label	Predicted Label	Confidence
1	holden caulfield did it better .	negative	positive	0.5469
2	the script kicks in , and mr. hartley 's distended pace and foot-dragging rhythms follow .	negative	positive	0.5398
3	it 's a charming and often affecting journey .	positive	positive	0.9997
4	unflinchingly bleak and desperate	negative	negative	0.9933
5	allows us to hope that nolan is poised to embark a major career as a commercial yet inventive filmmaker .	positive	positive	0.9985
6	the acting , costumes , music , cinematography and sound are all astounding given the production 's austere locales .	positive	positive	0.9973
7	it 's slow – very , very slow .	negative	negative	0.9989
8	although laced with humor and a few fanciful touches , the film is a refreshingly serious look at young women .	positive	positive	0.9992
9	a sometimes tedious film .	negative	negative	0.9911
10	or doing last year 's taxes with your ex-wife .	negative	negative	0.9684

Table 6: Example validation predictions from fine-tuned pretrained DistilBERT on SST-2.

3.3 Comparison: Pretrained vs. Random Initialization

I compared two DistilBERT models with exactly the same architecture:

- a pretrained DistilBERT model fine-tuned on SST-2
- a DistilBERT model with random initialization trained on SST-2 from scratch

The randomly initialized model was created using `DistilBertConfig` with the same vocabulary size and number of output labels as the pretrained model. The difference between them was not architectural size, but initialization. One started from pretrained language knowledge, while the other started from random weights. The comparison results are shown in Table 7.

Listing 5: From-scratch training script.

```
from __future__ import annotations
from dataclasses import dataclass
from typing import Dict, List, Tuple

import matplotlib.pyplot as plt
import torch
from datasets import load_dataset
from torch.utils.data import DataLoader
from transformers import (
    DistilBertTokenizer,
    DistilBertForSequenceClassification,
    DistilBertConfig)

MODEL_NAME = "distilbert-base-uncased"
MAX_LENGTH = 128
```

```
BATCH_SIZE = 32
NUM_EPOCHS = 3
LEARNING_RATE = 2e-5
WEIGHT_DECAY = 0.01
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")


def train_model(model, train_loader, val_loader, model_name_for_print="
    model"):
    optimizer = torch.optim.AdamW(
        model.parameters(),
        lr=LEARNING_RATE,
        weight_decay=WEIGHT_DECAY)

    train_losses = []
    val_accuracies = []

    for epoch in range(NUM_EPOCHS):
        avg_loss = train_one_epoch(model, train_loader, optimizer)
        val_acc = evaluate_accuracy(model, val_loader)

        train_losses.append(avg_loss)
        val_accuracies.append(val_acc)

        print(f"[{model_name_for_print}] Epoch {epoch + 1}: loss = {
            avg_loss:.4f}")
        print(f"[{model_name_for_print}] Validation accuracy after epoch {
            epoch + 1}: {val_acc:.4f}")

    return TrainResults(
        train_loss_per_epoch=train_losses,
        val_accuracy_per_epoch=val_accuracies)


def compare_scratch():
    print("Using device:", DEVICE)

    tokenizer = DistilBertTokenizer.from_pretrained(MODEL_NAME)
    sst2, train_loader, val_loader = prepare_sst2(tokenizer)

    config = DistilBertConfig(vocab_size=30522,
                              num_labels=2)

    model_scratch = DistilBertForSequenceClassification(config).to(DEVICE)

    scratch_results = train_model(
        model_scratch,
        train_loader,
        val_loader,
        model_name_for_print="scratch")


compare_scratch()
```

Model	Pretrained?	Params	Epoch 1 Acc	Epoch 3 Acc
DistilBERT fine-tuned	Yes	66,955,010	0.9037	0.9002
DistilBERT from scratch	No	66,955,010	0.7993	0.8154

Table 7: Comparison of pretrained versus randomly initialized DistilBERT on SST-2.

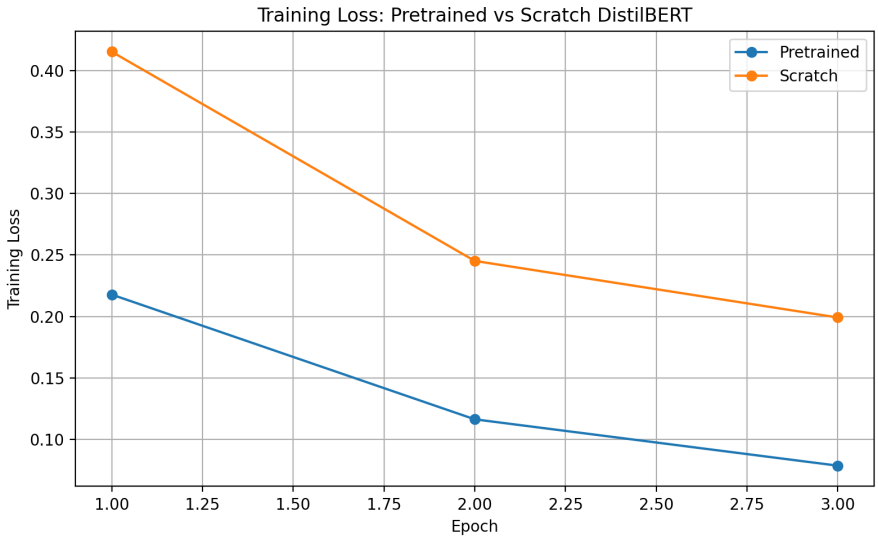

Figure 2: Training loss comparison between pretrained and randomly initialized DistilBERT on SST-2.

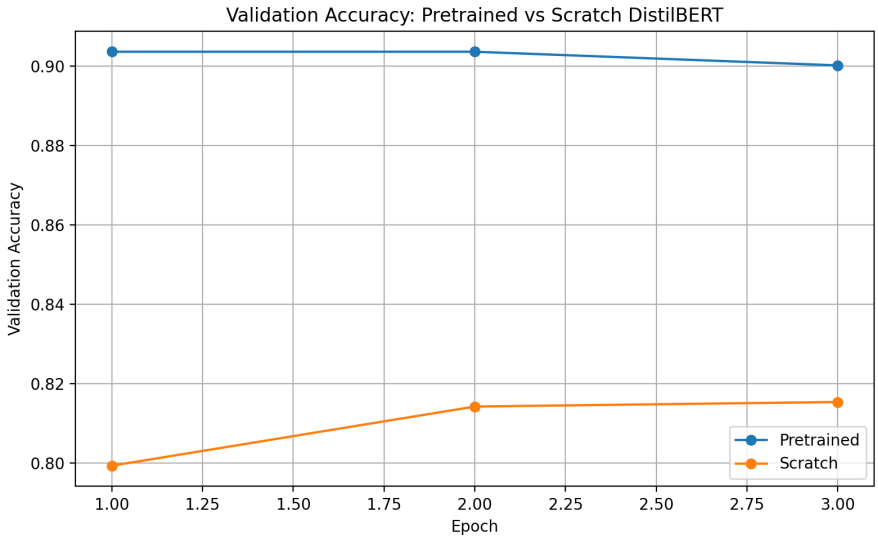

Figure 3: Validation accuracy comparison between pretrained and randomly initialized DistilBERT on SST-2.

3.4 Analysis

The accuracy gap. After 3 epochs, the pretrained DistilBERT model reached a validation accuracy of 0.9002, while the randomly initialized DistilBERT model reached 0.8154. Therefore, the accuracy gap after 3 epochs was

$$0.9002 - 0.8154 = 0.0848$$

which is about 8.5 percentage points. This is a large difference, especially because both models had exactly the same architecture, the same number of parameters, and were fine-tuned on exactly the same SST-2 labeled data. The only real difference was the initialization. The pretrained model started from weights that had already learned a great deal about English during language-model pretraining, while the scratch model started from random weights.

The accuracy advantage comes from what BERT learned before fine-tuning: word meanings, contextual relationships, grammar, common phrases, and general sentence structure. Fine-tuning on SST-2 does not need to teach the pretrained model what language is, it only needs to teach it how to map its existing language representations to positive and negative sentiment labels. In contrast, the randomly initialized model must learn basic language patterns and sentiment discrimination at the same time from only the SST-2 supervision. That is why the pretrained model converged faster, achieved lower loss, and stayed near 90% validation accuracy almost immediately.

Parallel to HW9. The common principle connecting HW9 and HW10 is that pretraining gives a model a strong representation of the input domain before the downstream task begins. In HW9, the pretrained ViT-Base already knew useful visual patterns such as shapes, textures, part-whole structure, and object-level regularities, so it adapted to Food-101 much more effectively than a small visTransformer trained from scratch. In HW10, the pretrained DistilBERT already knew useful linguistic patterns such as syntax, semantics, and contextual word usage, so it adapted to SST-2 much more effectively than the randomly initialized DistilBERT. In both cases, the pretrained model started much closer to a good solution. It did not need to discover the general structure of images or language from the downstream labels alone.

What I observed in both homeworks is that the gap is visible not only in final accuracy, but also in optimization behavior. The pretrained models began with better performance and improved quickly, while the scratch models learned more slowly and remained clearly weaker after the same training budget. So the common lesson is not just that "pretraining helps," but that pretraining provides reusable domain knowledge that downstream supervised datasets usually cannot teach efficiently on their own. That is why the pretrained models in both homeworks were better even though the fine-tuning data was the same.

Would more data close the gap? With 10× more labeled sentiment data, I think the from-scratch model would improve and the gap would likely shrink, but it would probably not disappear completely. Sentiment labels tell the model whether a sentence is positive or negative, but they do not directly teach many other things BERT learned during pretraining, such as fine-grained word meanings, syntax, and common contextual relationships between words. Those language properties are learned much more naturally from massive self-supervised text corpora than from sentiment labels alone. So extra labeled data would help, but the scratch model would still be learning language understanding in a much less efficient way.

With 100× more labeled data, the from-scratch model might get much closer, and it is possible that it could eventually match or even approach the pretrained model more closely, depending on

optimization, compute budget, and training time. However, that would be a very data-inefficient way to get the same knowledge. Pretraining is powerful precisely because it learns broad language structure from unlabeled text, which is much easier to obtain at scale than human sentiment annotations. In that sense, pretraining is not just a shortcut; it is a fundamentally more suitable way to learn general language knowledge. Sentiment labels can teach task-specific decision boundaries, but they are a poor substitute for the full range of linguistic information BERT acquires during pretraining.

4 Bonus Task: Training an Autoregressive Language Model with babyGPT

4.1 Important Setup Notes for Colab

4.2 Understanding Autoregressive Masking

The mask growth. The growing mask controls which token positions are allowed to attend to which earlier positions during autoregressive generation. At the beginning, the model only has the first valid position, so the mask starts as [1]. After the model predicts one more token, another valid position is added, so the mask grows by appending another 1. In this way, the mask expands step by step as the generated sequence becomes longer. This achieves causal decoding: at every step, the model can use only the tokens that already exist, and it cannot look ahead to future positions that have not yet been generated.

The reason the mask starts with length 1 instead of length 0 is that generation cannot begin from an empty visible sequence. The model needs at least one valid position to anchor the first prediction step. A zero-length mask would mean there is no active context at all, so there would be nothing for the decoder to process. Starting with [1] means the first token position is already active, and then each newly generated token extends the visible prefix. Conceptually, the growing mask matches the left-to-right nature of autoregressive language modeling: the usable context grows exactly as the generated sequence grows.

No cross-attention. In the English-Spanish translation Transformer from HW9, the decoder had a cross-attention layer because it needed to condition on a second sequence: the encoder output produced from the source-language sentence. In that setting, the encoder output served as a learned representation of the English input sentence, and the decoder used cross-attention to consult that representation while generating the Spanish translation. So the decoder was not generating from its own prefix alone; it was generating while also attending to an external information source.

In babyGPT, there is no such external source sequence. The model is not translating, summarizing, or answering based on another input sequence. It is only doing next-token prediction on a single stream of text. Therefore, the decoder needs only masked self-attention over its own previously seen tokens. There is no separate encoder output because there is no encoder-side input whose representation must be injected into decoding. In the translation setting, the encoder output carried the source sentence meaning. In the language-model setting, the only available information is the already generated prefix itself, so masked self-attention is enough and cross-attention is unnecessary.

Connection to GPT-3. Yes, the fundamental training mechanism is the same. Both babyGPT and GPT-3 are autoregressive Transformer language models trained to predict the next token from the previous tokens. In both cases, masked self-attention enforces causal structure, and the model

learns by maximizing the probability of the training text one token at a time. So the difference is not in the core objective, but in scale, capacity, and the amount of computation and data that can be used.

What GPT-3 gains from scale is not just "more memory," but a much stronger ability to represent long-range structure, semantic nuance, and many different patterns at once. A very small model like babyGPT can learn local token regularities, common short phrases, and repetitive patterns. GPT-3, by contrast, has enough depth, width, and parameter capacity to model much richer dependencies across tokens and across many domains. Scale allows it to generalize patterns that a tiny model cannot store or compose effectively, even if both models are trained with the same basic next-token objective.

4.3 Training babyGPT on the News Corpus

Listing 6: babyGPT training script.

```
from __future__ import annotations
import os
import sys

from babyGPT import babyGPT

articles_dir = "./saved_articles_dir_12M"
tokenizer_json = "./babyGPT_Examples/112_babygpt_tokenizer_cl_35035.json"

max_seq_length = 30
context_window_size = 25
context_buffer_size = 5
batch_size = 50
embedding_size = 128
num_basic_decoders = 4
num_atten_heads = 8
gradient_accumulation_steps = 1

assert context_window_size + context_buffer_size == max_seq_length

optimizer_params = {
    "beta1": 0.9,
    "beta2": 0.95,
    "epsilon": 1e-8}

num_warmup_steps = 4000
masking = True

path_saved_model = {
    "decoder": "saved_decoder",
    "embedding_generator": "saved_embedding_generator"}

gpt = babyGPT(
    max_seq_length=max_seq_length,
    batch_size=batch_size,
    embedding_size=embedding_size,
    num_atten_heads=num_atten_heads,
    beta1=optimizer_params["beta1"],
```

```
    beta2=optimizer_params["beta2"],
    epsilon=optimizer_params["epsilon"],
    num_warmup_steps=num_warmup_steps,
    masking=masking,
    verify_text_corpus=False,
    path_saved_model=path_saved_model)

xformer = gpt.TransformerFG(
    max_seq_length=max_seq_length,
    embedding_size=embedding_size,
    tokenizer_json=tokenizer_json,
    num_warmup_steps=num_warmup_steps,
    optimizer_params=optimizer_params)

master_decoder = gpt.MasterDecoderWithMasking(
    xformer=xformer,
    num_basic_decoders=num_basic_decoders,
    num_atten_heads=num_atten_heads,
    context_window_size=context_window_size,
    context_buffer_size=context_buffer_size,
    batch_size=batch_size,
    gradient_accumulation_steps=gradient_accumulation_steps,
    masking=masking)

dataloader = gpt.ArticleDatasetWithBufferedContext(
    gpt=gpt,
    tokenizer_json=tokenizer_json,
    context_window_size=context_window_size,
    context_buffer_size=context_buffer_size,
    articles_dir=articles_dir)


gpt.run_code_with_buffered_context_for_training_TransformerFG(
    xformer=xformer,
    master_decoder=master_decoder,
    dataloader=dataloader,
    checkpoint_frequency=4000,
    display_train_loss=False)
```

The loss curve is shown in Figure 4. The logged loss started at 60.0480 and then steadily decreased throughout training. The first few values were 60.0480, 58.1392, 55.0362, 51.5244, and 48.5643, while the last few values were 23.2503, 23.8924, 23.4432, 23.6911, and 23.6952. This shows a strong downward trend overall, even though the curve became noisier and flatter later in training. The large early drop suggests that the model quickly learned some basic token-level regularities of the corpus. Later, the improvement became smaller, which is typical once the model captures the easiest local patterns and starts approaching the limit of what such a small architecture can learn from the data.

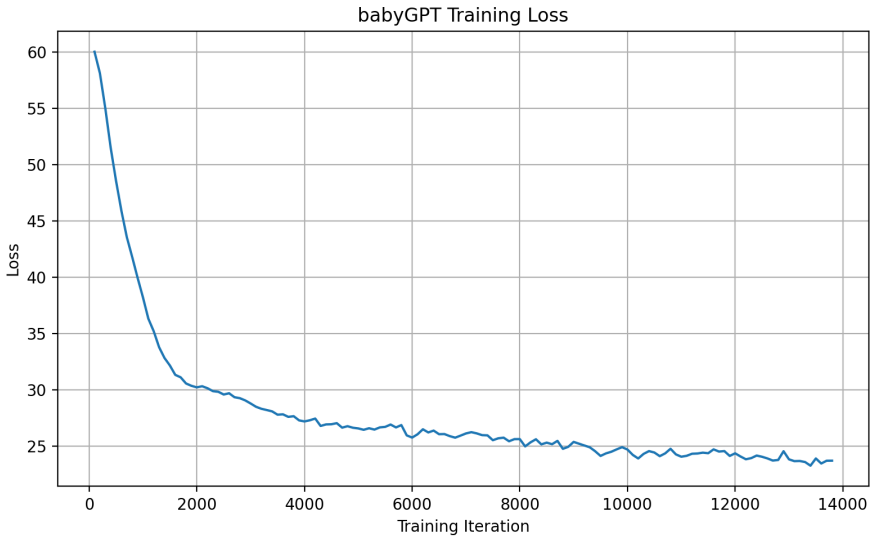

Figure 4: babyGPT training loss over iterations.

The qualitative training trends also changed over time. At very early iterations, the model produced almost meaningless outputs with repeated strange fragments such as `gen`, `sovereignty,` and other random-looking continuations. Later, the outputs became dominated by frequent short words such as `the`, `of`, and `to`, and the model started to repeat short phrase patterns instead of fully random text. To illustrate this progression, Table 8 shows ground-truth versus prediction comparisons at iterations 100, 4000, 8000, and 12000.

Iteration	Ground-truth	Predicted	Observation
100	`O pic . twitter . com / Nl5BYXP6DX --- Mike`	`sovereignty, seri gen gen gen gen gen ... 16- 16- 16- 16-`	At the beginning of training, the model output is essentially noise and does not resemble the target sequence.
4000	`the nearly three-year process of saving for your down`	`the the _ the _ the the _ the the the`	By 4000 iterations, the model no longer produces fully random fragments. It has started repeating common function-word patterns such as `the`.
8000	`little bits of dialogue . In real life , Rebecca`	`_ the s _ _ to _ _ to _ the s _ _ The _ the _ the in and in _ _`	By 8000 iterations, the output still lacks coherence, but it reflects stronger corpus-level regularities and more reuse of common short patterns.
12000	`say in the Lakers' head coaching search , but it's`	`to _ the _ the _ first _ s s of _ _ _ to _ s _ and _ the _ s _`	By 12000 iterations, the model still remains repetitive, but it can now echo prompt-related sports context such as `Lakers` and `head coaching search` before collapsing into repeated short tokens.

Table 8: Ground-truth vs. prediction comparisons showing how babyGPT outputs change over training.

Overall, the training results suggest that babyGPT learned shallow distributional structure but not strong semantic coherence. The falling loss confirms that optimization worked, while the snapshots show that a very small autoregressive model trained for a limited time tends to learn frequent local token patterns much more easily than fluent sentence generation.

4.4 Prompting the Trained Model

After training, I prompted the saved checkpoint using both in-domain and out-of-domain prompts. The in-domain prompts included sports-related examples such as `The Lakers`, `The coach said`, `Stephen Curry`, `In the second half`, and `LeBron James`. The out-of-domain prompts included `The recipe for pasta` and `Astronauts discovered`. I also tried several single-word prompts such as `The`, `First`, `Of`, `first`, and `second`. These were generated using `PromptResponder` with checkpoint index 12000 and `context_buffer_option="all_zeros"`.

The outputs were not fluent, but they were still useful for analysis.

- `The Lakers` → `The Lakers of the first the first the first the first of the`
- `The coach said` → `The coach said the first the first the first the first of the`
- `Stephen Curry` → `Stephen Curry de la la la de la la st la la`
- `The recipe for pasta` → `The recipe for pasta in the first the first the first of`
- `Astronauts discovered` → `Astronauts discovered the first the first the first the first of`
- `The` → `The re the first the first the first of the first of the`

These completions show that the model learned very shallow recurring token patterns. It often repeated high-frequency fragments such as `the first`, `of the`, and `de la la`, and it did not meaningfully adapt the continuation to the domain of the prompt. The sports prompts did not produce genuinely sports-specific continuations, and the out-of-domain prompts behaved similarly. Instead, the model mostly fell back to repetitive short phrases with high local probability.

Even though the generations were weak, the experiment is still informative. The main goal of the bonus task was to understand how a small autoregressive Transformer behaves during training and inference. The loss curve showed that babyGPT learned something meaningful at the optimization level, but the prompt completions revealed the limits of a tiny language model trained under limited resources. This contrast is important, lower training loss does not automatically imply coherent natural-language generation quality, especially for small autoregressive models.

References

[1] S. Mitra, *BME646 and ECE60146: Homework 10*. Available at: https://purdue.brightspace.com/d2l/le/content/1489347/viewContent/21517829/View

[2] C. Hughes, *2026 NFL Draft Notes: Rumors and rumblings around the league, including Jets and Giants*. SNY, April 20, 2026. Available at: https://sny.tv/articles/2026-nfl-draft-notes-rumors-jets-giants

[3] Fundação de Amparo à Pesquisa do Estado de São Paulo, *This common plant could clean microplastics from your drinking water* ScienceDaily, April 20, 2026. Available at: https://www.sciencedaily.com/releases/2026/04/260420014735.htm

[4] Reddit, *What are your favorite slang words or phrases to use regularly?*, April 20, 2026. Available at: https://www.reddit.com/r/Millennials/comments/1dp8kgn/what_are_your_favorite_slang_words_or_phrases_to/

[5] Wikipedia, *Twenty One Pilots*, April 20, 2026. Available at: https://en.wikipedia.org/wiki/Twenty_One_Pilots

[6] Wikipedia, *COVID-19 pandemic*, April 20, 2026. Available at: https://en.wikipedia.org/wiki/COVID-19_pandemic