**BME646 and ECE60146: Homework 10**

**Spring 2026**
**Due Date: Sunday, April 26, 2026, 11:59 pm ET**
**TA: Somosmita Mitra (mitra26@purdue.edu)**

Turn in typed solutions via Gradescope. Post questions to Piazza. Additional instructions can be found at the end. **Late submissions will be accepted with penalty: -10 points per late day, up to 2 days.**

# 1 Introduction

In HW9 you worked with Transformers for sequence-to-sequence translation and image classification. You trained DLStudio's `TransformerFG` on English-Spanish pairs, built a Vision Transformer from scratch on Food-101, and fine-tuned a pretrained ViT. The central theme of that homework was attention: how the $Q \cdot K^T$ mechanism lets a network decide which parts of its input matter most.

This homework shifts from *how* Transformers compute to *what they can learn when turned loose on massive text corpora*. You will work with BERT and GPT, two families of language models that took the Transformer architecture from HW9 and applied it at a scale that changed the field.

The homework has three tasks and an optional bonus.

In Task 1 you will train a Byte Pair Encoding (BPE) tokenizer using Professor Kak's `babyGPT` module, then compare its output against BERT's WordPiece tokenizer. You will see firsthand why a fixed-size vocabulary matters for language modeling and how domain-specific training reshapes what the tokenizer treats as a single token.

In Task 2 you will probe a pretrained BERT model without any fine-tuning. By masking words in sentences and asking BERT to fill them in, you will observe what the model absorbed during unsupervised pretraining on Wikipedia and the BookCorpus. You will also test BERT's Next Sentence Prediction ability, the second objective it was trained on.

In Task 3 you will fine-tune a pretrained DistilBERT on the Stanford Sentiment Treebank (SST-2), a binary movie-review sentiment dataset. You will then train the same architecture from randomly initialized weights and compare the two. This mirrors the from-scratch vs. pretrained comparison you did with Vision Transformers in HW9 Task 4, but now for language.

In the Bonus Task you will train your own autoregressive language model using `babyGPT`. Even on a small corpus with a small model, you will watch

the training loss drop and see the network's predictions evolve from random characters into recognizable (if imperfect) English. You will study the masking mechanism that makes autoregressive training possible - the same mechanism at the heart of every GPT model.

### 1.1 Quick reference: lecture slide numbers

Use this table to look up the lecture material for each topic.

| **HW10 topic** | **Lecture** | **Slides** |
|---|---|---|
| *Tokenization* | | |
| Why tokenization is needed for LLMs | BERT and GPT [1] | 16–17 |
| BPE tokenizer steps | BERT and GPT | 18–27 |
| TrainTokenizer class in babyGPT | BERT and GPT | 24–26 |
| WordPiece tokenizer in BERT | BERT and GPT | 38–42 |
| BPE vs. WordPiece vs. SentencePiece | BERT and GPT | 28–29 |
| *BERT* | | |
| Generative vs. discriminative training | BERT and GPT | 10–14 |
| BERT architecture (Master Encoder) | BERT and GPT | 31–33 |
| BERT input format (CLS, SEP, segments) | BERT and GPT | 34–37 |
| Pre-training: MLM and NSP | BERT and GPT | 43–46 |
| Fine-tuning BERT | BERT and GPT | 50–56 |
| *GPT and babyGPT* | | |
| GPT-2 core idea (autoregressive modeling) | BERT and GPT | 64–68 |
| Autoregressive masking | BERT and GPT | 67–69 |
| MasterDecoderWithMasking code | BERT and GPT | 70–73 |
| GPT-3 scale and few-shot learning | BERT and GPT | 76–81 |
| In-Context Learning | BERT and GPT | 82–88 |
| babyGPT module overview | BERT and GPT | 91–102 |

**Which slides to read for each task:**

| **Task** | **Key slides** |
|---|---|
| Task 1: Tokenization | BERT and GPT 16–29, 38–42 |
| Task 2: Probing pretrained BERT | BERT and GPT 10–14, 31–46 |
| Task 3: Fine-tuning for sentiment | BERT and GPT 50–56 |
| Bonus: babyGPT autoregressive training | BERT and GPT 64–73, 91–102 |

## 2 Setting up your environment

### 2.1 Prerequisites from previous homeworks

Ensure you have:

- Python 3.8+ with PyTorch, NumPy, Matplotlib installed

- A Google Colab account (GPU recommended for Task 3 and the Bonus Task)

### 2.2 Installing required packages

Install the following packages. On Google Colab these install in under a minute:

```
pip install transformers datasets
pip install babygpt lightning blingfire newspaper3k
```

The `transformers` library (by Hugging Face) provides pretrained BERT models and tokenizers. The `datasets` library provides the SST-2 sentiment dataset used in Task 3. The `babyGPT` module (by Professor Kak, version 1.1.4) provides the BPE tokenizer trainer and the autoregressive language model used in Tasks 1 and the Bonus Task. It is available on PyPI under the package name `babygpt`. The `lightning` and `blingfire` packages are dependencies that babyGPT uses internally.

### 2.3 Downloading the babyGPT text corpus and example scripts

Task 1 and the Bonus Task require a text corpus and pretrained tokenizer files that are **not** included in the `pip install`. You must download them separately.

**Step 1: Download the text datasets.** The dataset archive is hosted on the babyGPT page [2]. To download it you can run:

```
wget -q https://engineering.purdue.edu/kak/distBabyGPT/
    datasets_for_babyGPT.tar.gz
tar xzf datasets_for_babyGPT.tar.gz
```

This extracts two directories into the current working directory:

- 
- `saved_articles_dir_12M` (∼18 MB; 12 million Unicode characters) : the smaller corpus you will use for this homework
- `saved_Adrien_News_Articles_56M` (∼89 MB; 56 million Unicode characters): a larger athlete-news corpus

**Step 2: Extract the Examples directory.** The babyGPT source archive includes pretrained tokenizer JSON files and the training/prompting scripts you will adapt. Run:

```
pip download babygpt --no-binary :all: \
    -d /tmp/bg_src
cd /tmp/bg_src && tar xzf babyGPT-*.tar.gz
cp -r /tmp/bg_src/babyGPT-*/Examples \
    ./babyGPT_Examples
```

The `babyGPT_Examples/` directory contains:

- `112_babygpt_tokenizer_50002.json`: pretrained BPE tokenizer (∼50k vocab, trained on athlete news)
- `112_babygpt_tokenizer_cl_35035.json`: cleaned variant (∼35k vocab)
- `create_base_model_with_buffered_context.py`: the main training script
- `interact_with_prompts.py`: the prompting script
- `train_tokenizer.py`: tokenizer training script

**Colab tip:** If you download the datasets and examples to the ephemeral Colab session, they will be lost when the runtime disconnects. To avoid re-downloading, save them to Google Drive instead.

### 2.4 The SST-2 dataset (Task 3)

Task 3 uses the Stanford Sentiment Treebank binary classification split (SST-2) from the GLUE benchmark [6]. The dataset contains roughly 67,000 training sentences and 872 validation sentences, each labeled `positive` or `negative`. It downloads automatically through the `datasets` library:

```
from datasets import load_dataset

sst2 = load_dataset("glue", "sst2")
# sst2["train"]  -> 67349 examples
# sst2["validation"] -> 872 examples
# Each example has keys: "sentence", "label"
# label: 0 = negative, 1 = positive
```

The download is under 10 MB.

## 3 Programming tasks (100 points + 30 bonus)

### 3.1 Task 1: Tokenization (20 points)

**Objective:** Train a BPE tokenizer with babyGPT, then compare its behavior against BERT's WordPiece tokenizer across different text domains.

Understand why LLMs require a fixed-size vocabulary and how the training corpus shapes the tokenizer.

### 3.1.1 Training a BPE tokenizer (7 points)

Using the `TrainTokenizer` class from the `babyGPT` module, train a BPE tokenizer on the `saved_articles_dir_12M` corpus with a target vocabulary size of 10,000 tokens:

```
from babyGPT import *

tokenizer_trainer = TrainTokenizer(
    corpus_directory=
    './saved_articles_dir_12M',
    target_vocab_size=10000,
)
tokenizer_trainer.train_tokenizer()
```

This produces a JSON file containing the learned vocabulary and merge rules. Training may take 15–30 minutes on a CPU runtime.

You will also use the **pretrained tokenizer** that ships with babyGPT (the JSON file with a ∼50,000 token vocabulary trained on a larger athlete-news corpus). See the babyGPT documentation page for its filename and location.

**Deliverables:**

- The command or code you used to train the tokenizer
- The actual vocabulary size in your trained tokenizer JSON
- Three example words that your tokenizer keeps whole and three that it splits into subwords. Explain why, based on what you know about the training corpus.

### 3.1.2 Comparing tokenizers across domains (13 points)

Prepare three groups of sentences, **at least 3 sentences per group**, each sentence at least 25 words long:

1. **Sports news** (in-domain for babyGPT): sentences about athletes, games, scores. You may pull these from any online sports news source - cite your source.
2. **Scientific or medical text** (out-of-domain for babyGPT): sentences from a Wikipedia science article, a PubMed abstract, or similar. Cite your source.

3. **Informal internet text** (out-of-domain for both): tweets, Reddit posts, or chat messages with slang, abbreviations, hashtags, and emoji descriptions. Cite your source or compose them yourself.

For each sentence, tokenize it three ways:

1. **Word-level:** plain `split()` on whitespace
2. **BERT WordPiece:** using `DistilBertTokenizer.from_pretrained("distilbert-base-uncased")`
3. **babyGPT BPE:** using the pretrained ∼50k tokenizer that ships with babyGPT (loaded via `PreTrainedTokenizerFast`)

```
from transformers import (
    DistilBertTokenizer,
    PreTrainedTokenizerFast)

bert_tok = DistilBertTokenizer.from_pretrained(
    "distilbert-base-uncased")
baby_tok = PreTrainedTokenizerFast(
    tokenizer_file=
    "babyGPT_Examples/"
    "112_babygpt_tokenizer_50002.json")

text = "Your sentence here"

word_tokens = text.split()
bert_tokens = bert_tok.tokenize(text)
baby_ids = baby_tok.encode(text)
baby_decoded = baby_tok.decode(baby_ids)
```

**Deliverables:**

- A table for each sentence group showing: the sentence, token count for each method, and the actual tokens produced by BERT and babyGPT (truncate if very long, but show enough to illustrate differences)
- Answer these questions (1–2 paragraphs each):
  - On which sentence group does babyGPT produce the fewest tokens? Why does this happen, given what corpus it was trained on?
  - BERT lowercases all input before tokenizing. What are the consequences of this for tasks where capitalization carries meaning (e.g., named entities, abbreviations)?

– Slide 17 explains why a fixed vocabulary size is necessary for language modeling. In your own words, explain the connection between the vocabulary size and the final softmax layer of an autoregressive model.

**Grading:**

- Tokenizer training with examples (7 points)
- Cross-domain comparison tables and written answers (13 points)

## 3.2 Task 2: Probing pretrained BERT (35 points)

**Objective:** Without any fine-tuning, use BERT's two pretraining objectives - Masked Language Modeling (MLM) and Next Sentence Prediction (NSP), to see what the model learned from ingesting the BookCorpus and English Wikipedia. The goal is to build intuition for what unsupervised generative pretraining actually captures.

### 3.2.1 Masked Language Modeling: what does BERT know? (15 points)

BERT was pretrained to predict randomly masked tokens (Slides 43–44). You can query this ability directly using the Hugging Face `pipeline`:

```
from transformers import pipeline

mlm = pipeline("fill-mask",
    model="bert-base-uncased")

results = mlm("The capital of France is [MASK].")
for r in results:
    print(f"{r['token_str']:>12s}  "
          f"{r['score']:.4f}")
```

This returns the top-5 predictions for the masked position, each with a probability score.

Design **10 masked sentences** that probe different kinds of knowledge. Include at least two sentences from each of the following categories:

1. **Factual knowledge:** geography, science, history (e.g., "The chemical symbol for gold is [MASK].")
2. **Syntactic knowledge:** verb agreement, pronoun resolution (e.g., "The dogs in the park [MASK] chasing a squirrel.")

3. **Commonsense or associative:** things BERT should "know" from reading a lot of text (e.g., "She put on her coat because it was [MASK] outside.")

4. **Adversarial or tricky:** sentences where you expect BERT to struggle - rare words, ambiguous context, or recent slang that postdates BERT's 2018 training corpus

For each sentence, record the top-5 predictions and whether the correct answer appears among them.

**Deliverables:**

- A table with columns: Sentence (with [MASK]), Category, Top-5 predictions, Correct answer, Correct in top-5? (yes/no)

- A paragraph discussing patterns you observed. Which category did BERT handle best? Where did it fail, and why might that be?

#### 3.2.2 Next Sentence Prediction (10 points)

BERT was also pretrained on Next Sentence Prediction: given two sentences, decide whether the second actually follows the first in the original text (Slides 44–45). The demo code on Slides 51–53 of the lecture shows how to test this.

Construct **8 sentence pairs**: 4 that are genuine continuations and 4 that are not. For the genuine pairs, pick two consecutive sentences from a Wikipedia article or news story. For the false pairs, combine sentences from unrelated topics.

```
from transformers import (
    BertTokenizer,
    BertForNextSentencePrediction)
import torch

tokenizer = BertTokenizer.from_pretrained(
    'bert-base-uncased')
model = BertForNextSentencePrediction\
    .from_pretrained('bert-base-uncased')

sentence_A = "Purdue University is in Indiana."
sentence_B = "It was founded in 1869."

inputs = tokenizer(
    sentence_A, sentence_B,
    return_tensors='pt')
```

```
with torch.no_grad():
    outputs = model(**inputs)
prediction = torch.argmax(outputs.logits, dim=1)
# 0 = isNext, 1 = notNext
```

**Deliverables:**

- A table with columns: Sentence A, Sentence B, True label (isNext / notNext), BERT prediction, Correct? (yes/no)
- Overall accuracy (e.g., 6/8 correct)
- 3–4 sentences discussing the results. Did BERT get any wrong? If so, what made those pairs hard?

#### 3.2.3 Understanding questions (10 points)

Study the lecture slides and answer each question in 1–2 paragraphs:

- **Generative vs. discriminative training:** Slides 10–13 explain how generative training differs from discriminative training. In your own words, explain the difference using the notation $p(Y|X)$ vs. $p(X, Y)$. Why does generative pretraining help with downstream discriminative tasks like classification?
- **MLM vs. autoregressive:** BERT uses Masked Language Modeling (predict a masked token from its full bidirectional context), while GPT uses autoregressive modeling (predict each token from only the preceding tokens). What is the advantage of each approach? Why is MLM not suitable for text generation?
- **The [CLS] token in BERT:** In HW9 Task 2 you studied the class token in the Vision Transformer. BERT uses a nearly identical mechanism: a special [CLS] token whose embedding at the output of the final Transformer block is used for classification (Slide 36). Compare the role of [CLS] in BERT with the class token in DLStudio's `visTransformer`. What do they have in common?

**Grading:**

- MLM probing with table and discussion (15 points)
- NSP experiment with table and discussion (10 points)
- Written answers to the three questions (10 points)

### 3.3 Task 3: Fine-tuning BERT for sentiment analysis (45 points)

**Objective:** Fine-tune a pretrained DistilBERT model on the SST-2 sentiment dataset, then train the same architecture from random weights and compare.

#### 3.3.1 Background: from pretraining to fine-tuning

Slides 50–56 describe how BERT's pretrained weights can be adapted to specific tasks with a small amount of supervised training. The key insight, identical to what you saw with ViT in HW9, is that unsupervised pretraining on a large generic corpus gives the model a strong initialization. Fine-tuning then adjusts those weights for a particular task using far less labeled data than training from scratch would require.

DistilBERT is a smaller, faster version of BERT (66 million parameters vs. BERT-base's 110 million) that retains roughly 97% of BERT's performance [7]. We use it here because it trains faster on Colab's free GPU tier.

#### 3.3.2 Fine-tuning DistilBERT on SST-2 (20 points)

Fine-tune `distilbert-base-uncased` on the SST-2 training split and evaluate on the validation split:

```
from transformers import (
    DistilBertTokenizer,
    DistilBertForSequenceClassification)
from datasets import load_dataset
import torch
from torch.utils.data import DataLoader

device = torch.device(
    "cuda" if torch.cuda.is_available()
    else "cpu")

tokenizer = DistilBertTokenizer.from_pretrained(
    "distilbert-base-uncased")
model = DistilBertForSequenceClassification\
    .from_pretrained(
        "distilbert-base-uncased",
        num_labels=2)
model = model.to(device)

sst2 = load_dataset("glue", "sst2")
```

```
21
22 def tokenize_fn(batch):
23     return tokenizer(
24         batch["sentence"],
25         padding="max_length",
26         truncation=True,
27         max_length=128)
28
29 train_ds = sst2["train"].map(
30     tokenize_fn, batched=True)
31 val_ds = sst2["validation"].map(
32     tokenize_fn, batched=True)
33
34 train_ds.set_format("torch",
35     columns=["input_ids",
36              "attention_mask", "label"])
37 val_ds.set_format("torch",
38     columns=["input_ids",
39              "attention_mask", "label"])
40
41 train_loader = DataLoader(
42     train_ds, batch_size=32, shuffle=True)
43 val_loader = DataLoader(
44     val_ds, batch_size=32)
45
46 optimizer = torch.optim.AdamW(
47     model.parameters(),
48     lr=2e-5, weight_decay=0.01)
49
50 # Train for 3 epochs
51 for epoch in range(3):
52     model.train()
53     total_loss = 0
54     for batch in train_loader:
55         input_ids = batch["input_ids"].to(device)
56         mask = batch["attention_mask"].to(device)
57         labels = batch["label"].to(device)
58         outputs = model(
59             input_ids=input_ids,
60             attention_mask=mask,
61             labels=labels)
62         loss = outputs.loss
63         loss.backward()
64         optimizer.step()
65         optimizer.zero_grad()
66         total_loss += loss.item()
67     avg_loss = total_loss / len(train_loader)
68     print(f"Epoch {epoch}: loss = {avg_loss:.4f}")
69
```

```
    # Evaluate
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for batch in val_loader:
            input_ids = batch["input_ids"]\
                .to(device)
            mask = batch["attention_mask"]\
                .to(device)
            labels = batch["label"].to(device)
            outputs = model(
                input_ids=input_ids,
                attention_mask=mask)
            preds = torch.argmax(
                outputs.logits, dim=1)
            correct += (preds == labels).sum()\
                .item()
            total += labels.size(0)
    print(f"  Val accuracy: {correct/total:.4f}")
```

With pretrained DistilBERT on SST-2, you should reach roughly 88–92% validation accuracy within 3 epochs. Each epoch takes about 10–15 minutes on a Colab T4.

**Deliverables:**

- Your fine-tuning script
- Training loss curve over 3 epochs
- Validation accuracy after each epoch
- Final validation accuracy
- 10 example sentences from the validation set with their true labels, predicted labels, and model confidence scores. Include at least 2 misclassified examples (if any).

#### 3.3.3 Comparison: pretrained vs. random initialization (12 points)

Now train the *same* DistilBERT architecture from scratch, without pretrained weights:

```
from transformers import (
    DistilBertConfig,
    DistilBertForSequenceClassification)

config = DistilBertConfig(
```

```
    vocab_size=30522,
    num_labels=2)

model_scratch = \
    DistilBertForSequenceClassification(config)
model_scratch = model_scratch.to(device)

# Train with the SAME hyperparameters
# (lr=2e-5, 3 epochs, batch_size=32)
# and record loss and accuracy each epoch
```

**Deliverables:**

- A comparison table:

| Model | Pretrained? | Params | Epoch 1 Acc | Epoch 3 Acc |
|---|---|---|---|---|
| DistilBERT fine-tuned | Yes | ? | ? | ? |
| DistilBERT from scratch | No | ? | ? | ? |

- Training loss curves for both models on the same plot
- Validation accuracy curves for both models on the same plot

#### 3.3.4 Analysis (13 points)

**Report** (answer each in 1–2 paragraphs):

- **The accuracy gap:** How large is the difference in validation accuracy between the pretrained and randomly initialized models after 3 epochs? The pretrained model had access to exactly the same labeled training data as the from-scratch model during fine-tuning. Where does the accuracy advantage come from?
- **Parallel to HW9:** In HW9 Task 4 you compared a from-scratch visTransformer (small, no pretraining) against a pretrained ViT-Base fine-tuned on the same Food-101 subset. You saw a large accuracy gap there too. What is the common principle connecting these two experiments? State it in your own words, not as a vague generality, but grounded in what you observed in both homeworks.
- **Would more data close the gap?** SST-2 has about 67,000 training sentences. If you had 10× more labeled sentiment data, would the from-scratch model eventually catch up to the pretrained one? What

about 100× more? Think about what BERT learned during pretraining (word meanings, grammar, common phrases) and whether those things can be learned from sentiment labels alone.

**Grading:**

- Fine-tuning script, loss/accuracy curves, example predictions (20 points)
- Comparison table and paired plots (12 points)
- Analysis paragraphs (13 points)

# 4 Bonus Task: Training an Autoregressive Language Model with babyGPT (30 Points)

**This task is entirely optional.** Points earned here are added on top of the 100-point base from Tasks 1, 2, and 3.

**Objective:** Train a small GPT-style language model using Professor Kak's `babyGPT` module, observe the training dynamics, prompt your trained model, and understand the autoregressive masking mechanism that makes this kind of learning possible.

## 4.1 Important setup notes for Colab

**DDP strategy.** babyGPT uses PyTorch Lightning with a distributed training strategy that crashes on Colab's single-GPU runtime. Fix it by replacing

```
# change this
strategy =  'ddp_find_unused_parameters_true'
#to this
strategy = 'auto'
```

**Training from the Colab terminal.** You may find it more reliable to run the babyGPT training script from the Colab terminal (click the terminal icon in the left sidebar) rather than from a notebook cell. Lightning's progress output works better in a terminal, and a long-running `nohup` process will survive even if you navigate away from the tab.

## 4.2 Understanding autoregressive masking (5 points)

Before training, study the `MasterDecoderWithMasking` class (Slides 70–73 of the lecture and the code on Slides 72–73).

**Report** (answer each in 1–2 paragraphs):

- **The mask growth:** On Slide 72, Line (E) initializes the mask as a single-element vector `[1]`, and Line (R) appends a 1 after each prediction step. Explain what this growing mask achieves. Why does the mask start with length 1 rather than length 0?

- **No cross-attention:** Compare the `BasicDecoderWithMasking` on Slide 73 with the `BasicDecoder` used for English-Spanish translation in HW9 (which you studied in the Transformers lecture). The key structural difference is that the language-model decoder has no cross-attention layer. Why not? What played the role of the "encoder output" in the translation setting, and why is there no equivalent here?

- **Connection to GPT-3:** Slide 78 lists the GPT-3 configuration: 96 layers, embedding size 12,288, 96 attention heads, 175 billion parameters. The babyGPT model you will train uses 4 layers and embedding size 128. Aside from the obvious size difference, is the fundamental training mechanism the same? What does GPT-3 gain from scale that a small model cannot compensate for?

### 4.3 Training babyGPT on the news corpus (20 points)

**Non-NaN losses are required for credit.** You must demonstrate a working model whose training loss decreases over time. If your loss is `nan`, you will not receive points for this section. See the NaN troubleshooting advice in the FAQ.

Using the `saved_articles_dir_12M` corpus and the **cleaned** pretrained tokenizer, train a babyGPT language model. The main training script is `create_base_model_with_buffered_context.py` in the `babyGPT_Examples/` directory. Copy it into your working directory and set the following parameters:

```
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
# NOTE: context_window_size + context_buffer_size
#       must equal max_seq_length

```

```
batch_size     = 50
embedding_size = 128
num_basic_decoders = 4
num_atten_heads    = 8
gradient_accumulation_steps = 1
```

The script prints loss and ground-truth vs. prediction comparisons every 100 iterations, and saves checkpoints every 4,000 iterations to `checkpoint_dir/`. Copy `checkpoint_dir/` to Google Drive periodically.

**Deliverables:**

- Your training script with the parameters you used
- Training loss values (a plot, or the printed loss at each 100-iteration checkpoint)
- At least two ground-truth vs. prediction comparisons showing how predictions change over training
- A paragraph describing the training behavior: Does the loss decrease? Do the predictions become more coherent? If you encountered `nan` losses, describe what you tried.

### 4.4 Prompting your trained model (5 points)

Use babyGPT's `PromptResponder` class to generate text from your trained model. The prompting script is `interact_with_prompts.py` in the `babyGPT_Examples/` directory. Copy it and set the same configuration parameters you used during training. You must also set the `checkpoint_index` to the iteration number of the checkpoint you want to load (check `checkpoint_dir/` for available checkpoints). Try **at least 5 prompts**, including:

1. Two prompts on topics that appear in the training corpus (sports, athletes)
2. Two prompts on topics that do **not** appear in the training corpus (cooking, space exploration, or anything unrelated to sports news)
3. One prompt that is a single common word (e.g., "The")

For each prompt, record the generated text (even if it is gibberish or only a few words).

**Deliverables:**

- The 5+ prompts and their generated responses

- A paragraph analyzing the results: Is the model noticeably better at in-domain prompts than out-of-domain ones? Does the output resemble English, or is it mostly noise? How does this connect to the limited scale of training compared to GPT-3?

**Grading:**

- Autoregressive masking explanations (5 points)
- Training with non-NaN loss and predictions (20 points)
- Prompting experiment and analysis (5 points)

# 5 Submission instructions

## 5.1 What to submit in the report

**Read this section carefully. Incomplete submissions will lose points. Every item listed below must be present in your submission.**

1. **Task 1: Tokenization (20 points)**
   (a) **Code:** Tokenizer training script
   (b) **Text:** Vocabulary size, example words kept whole / split, and explanations
   (c) **Table:** Cross-domain tokenization comparison (3 sentence groups $\times$ 3 tokenizers)
   (d) **Text:** Answers to the three tokenization questions
2. **Task 2: Probing pretrained BERT (35 points)**
   (a) **Code:** MLM probing script
   (b) **Table:** 10 masked sentences with top-5 predictions and correctness
   (c) **Text:** MLM discussion paragraph
   (d) **Code:** NSP experiment script
   (e) **Table:** 8 sentence pairs with predictions and accuracy
   (f) **Text:** NSP discussion
   (g) **Text:** Generative vs. discriminative explanation
   (h) **Text:** MLM vs. autoregressive comparison

(i) **Text:** [CLS] token comparison with HW9 visTransformer class token

3. **Task 3: Fine-tuning BERT for sentiment (45 points)**

   (a) **Code:** Fine-tuning script

   (b) **Figure:** Training loss curve

   (c) **Text:** Validation accuracy per epoch and final accuracy

   (d) **Text:** 10 example predictions with confidence scores

   (e) **Code:** From-scratch training script

   (f) **Table:** Comparison table (pretrained vs. scratch)

   (g) **Figure:** Paired loss curves and paired accuracy curves

   (h) **Text:** Accuracy gap discussion

   (i) **Text:** Parallel to HW9 ViT comparison

   (j) **Text:** Would more data close the gap?

4. **Bonus Task: babyGPT autoregressive training (30 bonus points, optional)**

   (a) **Text:** Mask growth explanation

   (b) **Text:** No cross-attention explanation

   (c) **Text:** Connection to GPT-3 scale

   (d) **Code:** Training script with parameters

   (e) **Figure:** Training loss curve (must show decreasing, non-NaN loss)

   (f) **Text:** Ground-truth vs. prediction comparisons

   (g) **Text:** Training behavior observations

   (h) **Text:** 5+ prompts with generated responses

   (i) **Text:** Prompting analysis paragraph

### 5.2 Code files to submit

1. `tokenizer_comparison.py`: Tokenizer training and cross-domain comparison code

2. `bert_probing.py`: MLM and NSP probing experiments

3. `babygpt_train.py`: babyGPT training and prompting script **(only if attempting the Bonus Task)**
4. `sentiment_finetune.py`: DistilBERT fine-tuning and from-scratch comparison
5. `requirements.txt`: Dependencies (must include `transformers`, `datasets`, `babygpt`)
6. `README.md`: Instructions to run your code, including Colab setup steps

## 5.3 Autograder interface requirements

The autograder will import your `sentiment_finetune.py` file and call the following function. If the file name or function signature does not match exactly, the autograder will fail and you will lose points.

### 5.3.1 Required file names

Submit exactly this file name (case-sensitive):

- `sentiment_finetune.py`

### 5.3.2 `sentiment_finetune.py`

Must contain:

- `predict_sentiment(texts, model, tokenizer)`: Takes a list of strings, a `DistilBertForSequenceClassification` model, and a `DistilBertTokenizer`. Returns a tuple of two lists: integer labels (`0` for negative, `1` for positive) and confidence scores (the softmax probability of the predicted class).

**The autograder will run:**

```
from sentiment_finetune import predict_sentiment
from transformers import (
    DistilBertTokenizer,
    DistilBertForSequenceClassification)

tokenizer = DistilBertTokenizer.from_pretrained(
    "distilbert-base-uncased")
# Assume model has been loaded from a checkpoint
model = DistilBertForSequenceClassification\
    .from_pretrained("./checkpoint")

```

```
texts = [
    "This movie was absolutely wonderful.",
    "A complete waste of time.",
    "The acting was decent but the "
    "plot made no sense.",
]

labels, scores = predict_sentiment(
    texts, model, tokenizer)

assert isinstance(labels, list)
assert isinstance(scores, list)
assert len(labels) == 3
assert len(scores) == 3
assert all(l in [0, 1] for l in labels)
assert all(0.0 <= s <= 1.0 for s in scores)
```

#### 5.3.3 Autograder summary

| File | Required names | Autograder tests |
|---|---|---|
| `sentiment_finetune.py` | `predict_sentiment` | Label type/range, score range, list length, basic predictions |

**Important notes:**

- Do **not** rename the file or function.
- Do **not** put executable code at the module level. Guard scripts with `if __name__ == '__main__':` blocks.
- **Cite all source code** you borrow or adapt from Hugging Face examples, babyGPT, or DLStudio.

### 5.4 Code quality requirements

- Well-commented code explaining what each section does
- Meaningful variable and function names
- Docstrings for all classes and functions
- Code should run without modification (given correct paths and model downloads)
- Follow PEP 8 style guidelines

### 5.5 Report formatting

- **Include code snippets in the report for every task.** For each task, show the code you wrote alongside the outputs it produced. If the code snippet is not present, you will lose points on that task.
- Place output directly next to the corresponding code block. Avoid placing code and output in separate sections.
- Use figures and tables with captions
- Number equations, figures, and tables
- Use proper citations when referencing papers or course materials

### 5.6 Additional guidelines

- Convert Jupyter notebooks to .py files. **Do NOT submit .ipynb**
- If submitting late, submit only once
- **Do NOT submit dataset files or model weights**, only code
- Your code and report must be your own work
- Document your compute setup (CPU/GPU, estimated training times) in your README

## 6 Frequently asked questions (FAQ)

**Q: The babyGPT tokenizer training is slow.**

A: On a CPU runtime, training a 10,000-token vocabulary on the 12M corpus takes roughly 15–30 minutes. This is expected. You do not need a GPU for tokenizer training. If it seems stuck, it is likely still running - the script prints progress every 100 iterations.

**Q: I can't install babyGPT on Colab.**

A: Run `pip install babygpt` (note: all lowercase) in a code cell. This installs version 1.1.4 or later from PyPI. You also need `pip install lightning blingfire newspaper3k` for its dependencies. If you get conflicts, try `pip install babygpt --no-deps` and install missing dependencies individually. Remember that the datasets and example scripts must be downloaded separately (see Section 2.3).

**Q: My babyGPT training produces mostly gibberish after training.**

A: That is expected for a model this small trained on this little data. The purpose of the Bonus Task is to observe the training mechanism and see incremental improvement, not to build a useful language model. Even GPT-2 used 1.5 billion parameters and 40 GB of training data. Report whatever output you get and analyze it honestly.

**Q: My babyGPT training loss is `nan` from the first iteration.**

A: This usually means the output layer is too large relative to the embedding size. Use the **cleaned** tokenizer (`112_babygpt_tokenizer_cl_35035.json`, ∼35k vocab) instead of the 50k variant. The 50k tokenizer creates a `Linear(128, 50002)` output layer that dominates the model and causes numerical instability. If losses are still `nan` with the 35k tokenizer, try changing the parameters.

**Q: babyGPT crashes with a DDP or "unused parameters" error on Colab.**

A: babyGPT uses a distributed training strategy that does not work on Colab's single GPU. You must change the strategy to `'auto'` before training. See Section 4.1 for details.

**Q: Should I run babyGPT training from a notebook cell or the terminal?**

A: The Colab terminal is more reliable for long training runs. Lightning's output displays better in a terminal, and you can use `nohup` to keep the process running if you navigate away. See Section 4.1 for the recommended commands.

**Q: The BERT MLM pipeline returns predictions I didn't expect.**

A: BERT was trained on text from before 2019. It will not know about recent events, people, or slang. It may also reflect biases present in its training data. Report and discuss unexpected results rather than trying to force "correct" outputs.

**Q: My from-scratch DistilBERT barely beats random (50%) after 3 epochs.**

A: That is exactly the point. A 66-million-parameter model trained from random initialization on 67k sentences does not have enough data or training time to learn both language understanding and sentiment simultaneously. This is why pretraining matters. Report your results.

**Q: Can I use a different model for Task 3?**

A: You may substitute `bert-base-uncased` for DistilBERT if you prefer, but DistilBERT is faster to train on Colab. Do **not** use a non-BERT model

(e.g., GPT-2, RoBERTa) since the homework is specifically about BERT.

**Q: Can I train for more than 3 epochs?**

A: Yes. If your Colab session has time remaining, feel free to train longer and report the additional results. The minimum requirement is 3 epochs for Task 3. For the Bonus Task, train as long as your Colab session allows and report whatever results you obtain.

**Q: The pretrained babyGPT tokenizer JSON is not where I expected it.**

A: The tokenizer JSON files are **not** installed by `pip install babygpt`. They are in the source archive's `Examples/` directory. Follow the extraction steps in Section 2.3 (Step 2) to get them into a `babyGPT_Examples/` directory. The file you want is `112_babygpt_tokenizer_50002.json` (the ∼50k vocab tokenizer) or `112_babygpt_tokenizer_cl_35035.json` (the cleaned ∼35k variant).

**Q: My Colab session disconnected during babyGPT training.**

A: The training script saves checkpoints automatically every 4,000 iterations to `checkpoint_dir/`. If you copied that directory to Google Drive before the disconnect, you can resume by pointing the script to the latest checkpoint. If you did not save to Drive, you will need to restart training. Copy `checkpoint_dir/` to Drive periodically.

**Q: Can I work in a group?**

A: No. This is individual work.

**Q: Can I reuse code from previous homeworks?**

A: Yes. You are encouraged to reuse plotting utilities, data loading patterns, etc.

## References


[1] Avinash Kak, *Transformer Based Learning for BERT and GPT Language Models*. Available at: https://engineering.purdue.edu/kak/pdf-kak/LLMLearning.pdf

[2] Avinash Kak, *babyGPT Module*. Available at: https://engineering.purdue.edu/kak/distBabyGPT/

[3] Avinash Kak, *DLStudio Platform*. Available at: https://engineering.purdue.edu/kak/distDLS/

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, *BERT: Pre-Training of Deep Bidirectional Transformers for Language*

*Understanding*, NAACL, 2019. Available at: https://arxiv.org/abs/1810.04805

[5] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever, *Language Models are Unsupervised Multitask Learners*, OpenAI, 2019. Available at: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

[6] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman, *GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding*, ICLR, 2019. Available at: https://arxiv.org/abs/1804.07461

[7] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf, *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*, NeurIPS Workshop, 2019. Available at: https://arxiv.org/abs/1910.01108