BME646 / ECE60146: Homework 10 Report

Dominic Flowers

April 25, 2026

1 Task 1: Tokenization

1.1 Training a BPE Tokenizer

```python
def train_bpe_tokenizer():
    """Trains a BPE tokenizer using babyGPT."""
    tokenizer_trainer = babyGPT.TrainTokenizer(
        corpus_directory='./saved_articles_dir_12M',
        target_vocab_size=10000,
    )
    merge_rules_dict, merges = tokenizer_trainer.train_tokenizer()
    print("Training complete.")
```

Listing 1: BPE Tokenizer Training Script

Vocabulary Size: The training corpus consisted of 2,757 files. Prior to filtering, the corpus contained 2,059,384 words, which was reduced to 2,056,792 words post-filtering (comprising 104,754 unique words). Following training with a target size of 10,000, the final tokenizer vocabulary converged to 9,916 tokens.

Example Words:

- **Words Kept Whole:** trauma → ['trauma', '.']
- **Words Kept Whole:** overflew → ['overflew']
- **Words Kept Whole:** power → ['power']
- **Words Split into Subwords:** Änderung → ['Ä', 'n', 'der', 'un', 'g']
- **Words Split into Subwords:** effortless → ['effort', 'le', 's', 's']
- **Words Split into Subwords:** boasts → ['bo', 'a', 'st', 's']

Explanation: Because a BPE tokenizer merges character sequences based on their frequency in the training data, words kept whole are typically highly prevalent in the athlete-news corpus. For instance, "power" and "trauma" are common descriptors in sports reporting and injury discussions, while "overflew" appeared frequently enough to warrant a single token. Conversely, the split words represent less common terms or complex character combinations. "Änderung" is fragmented likely due to it containing an uncommon non-ASCII

character (Ä). Similarly, "effortless" and "boasts" are fractured into their root forms and suffixes, indicating that while their base components may be common, these specific variations did not appear with enough frequency to be merged into single tokens.

1.2 Comparing Tokenizers Across Domains

Sentence (Sports)	**Word Count**	**BERT Tokenizer**	**babyGPT Tokenizer**
On Wednesday, Sergio Garcia, a LIV player who won ...	33	38 'on' 'wednesday' ...	52 5423 3831 ...
A pass rusher, be it Reese or David Bailey, appear...	31	39 'a' 'pass' ...	45 65 10874 ...
An action-packed Champions League quarter-final se...	54	75 'an' 'action' ...	102 2390 30092 ...

Table 1: Cross-domain comparison: Sports News (In-Domain)

Sentence (Sci/Med)	Word Count	BERT Tokenizer	babyGPT Tokenizer
Numerical analysis finds application in all fields...	32	36 'numerical' 'analysis' ...	49 22053 2098 ...
A Markov chain is a type of Markov process that ha...	33	40 'a' 'marko' ...	52 65 36994 ...
Concerns about the soundness of arguments involvin...	30	39 'concerns' 'about' ...	56 8839 2442 ...

Table 2: Cross-domain comparison: Scientific/Medical (Out-of-Domain)

Sentence (Informal)	Word Count	BERT Tokenizer	babyGPT Tokenizer
Yo, he was def giving unc aura when he popped up i...	29	31 'yo' ',' ...	42 5548 44 ...
#Mysterious #Aurofarm I aint believe clavicular go...	28	42 '#' 'mysterious' ...	52 35 3488 ...
I just dedicated a 100 hours to dark souls 3 tryin...	33	37 'i' 'just' ...	47 73 15745 ...

Table 3: Cross-domain comparison: Informal Internet Text (Out-of-Domain)

Token Count Analysis: Empirically, the babyGPT tokenizer produces the most absolute tokens in every domain. Out of all domains, the lowest count for babyGPT was the cross-domain, informal text. While I expected an in-domain (sports) advantage due to its training,

the informal text likely yields lower overall token counts due to reduced complexity and shorter average word lengths. Conversely, both the scientific and sports corpora exhibit a higher, and remarkably similar, token-to-word expansion ratio. This reflects the presence of multisyllabic terminology in both domains, forcing the algorithm to frequently fragment words into multiple subword tokens.

Consequences of BERT Lowercasing: By mapping all inputs to lowercase, BERT artificially compresses the embedding space, which simplifies training and improves sample efficiency. However, discarding casing removes a critical heuristic for semantic disambiguation. Consequently, the model's self-attention layers are forced to rely exclusively on contextual and positional cues to identify proper nouns, acronyms, and named entities. This loss of direct signaling could degrade performance.

Vocabulary Size and Softmax: In an autoregressive language model, the network must output a valid probability distribution over all possible next tokens. This is achieved via a final dense layer followed by a softmax activation. Crucially, the dimensionality of this output space must correspond exactly to the fixed vocabulary size, V. Every discrete token in the vocabulary requires a dedicated logit in the output layer; therefore, the vocabulary size directly dictates the parameter count of the final matrix and governs the probability distribution at every inference step.

2 Task 2: Probing Pretrained BERT

2.1 Masked Language Modeling (MLM)

```python
def probe_mlm():
    """Probes BERT's Masked Language Modeling capabilities."""
    mlm = pipeline("fill-mask", model="bert-base-uncased")

    sentences = [
        # Factual
        "The chemical symbol for gold is [MASK].",
        "The maximum effective range of the [MASK] carbine is 500m for point.", # M4
        # Syntactic
        "The dogs in the park [MASK] chasing a squirrel.",
        "I [MASK] the rain, it is very calming", # Love
        # Commonsense
        "She put on her coat because it was [MASK] outside.",
        "He is wearing sunscreen to protect his fresh [MASK]", # Tattoo
        "Always [MASK] plan when given a deadline", # backward
        # Adversarial/Tricky
        "Never let a [MASK] kiss you and Never let a kiss fool you", # fool
        "Clavicular was [MASK] the ASU frat guy", # framemogging
        "By day the [MASK], and the dance by night" # frolic
    ]

    for sentence in sentences:
        print(f"\nSentence: {sentence}")
        results = mlm(sentence)
        for r in results:
            print(f"{r['token_str']:>12s} {r['score']:.4f}")
```

Listing 2: MLM Probing Script

Sentence (with [MASK])	Category	Top-5 Predictions	Correct	In Top-5?
The chemical symbol for gold is [MASK].	Factual	gold, UNK, k, sigma, c	AU	No
The maximum effective range of the [MASK] carbine is 500m for point.	Factual	m1, m2, sniper, light, m3	m4	No
The dogs in the park [MASK] chasing a squirrel.	Syntactic	were, are, ”,”, was, ”.”	are	Yes
I [MASK] the rain, it is very calming.	Syntactic	smell, feel, love, like, hate	love	Yes
She put on her coat because it was [MASK] outside.	Common	cold, warm, dark, freezing, hot	cold	Yes
He is wearing sunscreen to protect his fresh [MASK]	Common	”.”, ”;”, ”!”, ”?”, ”—”	tattoo	No
Always [MASK] plan when given a deadline.	Common	a, on, in, the, to	backward	No
Never let a [MASK] kiss you and Never let a kiss fool you.	Adversarial	man, kiss, guy, girl, women	fool	No
Clavicular was [MASK] the ASU frat guy.	Adversarial	for, from, in, at, called	framemogging	No
By day the [MASK], and the dance by night.	Adversarial	dance, ballet, music, opera, waltz	frolic	No

Table 4: BERT Masked Language Modeling Predictions

MLM Discussion: BERT demonstrated the highest proficiency on syntactic and commonsense sentences, successfully leveraging its self-attention mechanism to resolve bidirectional context and grammatical structure. Conversely, performance degraded significantly on factual queries requiring specific memorization (e.g., retrieving AU for gold) and adversarial inputs containing out-of-distribution modern slang (e.g., framemogging). This indicates that while the model successfully captures the statistical distribution of standard English syntax, its internal representations lack explicit grounding in objective facts and are highly sensitive to vocabulary that postdates its training corpus.

2.2 Next Sentence Prediction (NSP)

```
def probe_nsp():
    """Probes BERT's Next Sentence Prediction capabilities."""
    tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
    model = BertForNextSentencePrediction.from_pretrained('bert-base-
    uncased')

    # Construct 8 sentence pairs (4 genuine, 4 fake)
```

```
    pairs = [
        ("Purdue University is in Indiana.", "It was founded in 1869."), # Genuine
        ("Purdue University is in Indiana.", "The sky is blue today."), # Fake
        ("USMA is in West Point, New York.", "It was founded in 1802."),
        ("USMA is in West Point, New York.", "President Andrew Jackson was responsible for the Trail of Tears."),
        ("The GS1 is the global identification standards organization for retail.", "Every country has an assigned country code which precedes the company code."),
        ("The GS1 is the global identification standards organization for retail.", "Narnia is a fictitious country in a closet."),
        ("The illusion is similar to the Ames room, in which objects can also appear to roll against gravity.", "The opposite phenomenon - an uphill road that appears flat is known in bicycle racing as a false flat."),
        ("The illusion is similar to the Ames room, in which objects can also appear to roll against gravity.", "In geometry, two geometric objects are perpendicular if they intersect at right angles."),
    ]

    for sent_A, sent_B in pairs:
        inputs = tokenizer(sent_A, sent_B, return_tensors='pt')
        with torch.no_grad():
            outputs = model(**inputs)
            prediction = torch.argmax(outputs.logits, dim=1).item()
            label_str = "isNext" if prediction == 0 else "notNext"
            print(f"A: {sent_A} | B: {sent_B} -> Prediction: {label_str}")
```

Listing 3: NSP Experiment Script

Sentence A	Sentence B	True Label	Prediction	Correc
Purdue University is in Indiana.	It was founded in 1869.	Is Next	Is Next	Yes
Purdue University is in Indiana.	The sky is blue today.	Not Next	Is Next	No
USMA is in West Point, New York.	It was founded in 1802.	Is Next	Is Next	Yes
USMA is in West Point, New York.	President Andrew Jackson was responsible for the Trail of Tears.	Not Next	Not Next	Yes
The GS1 is the global identification standards organization for retail.	Every country has an assigned country code which precedes the company code.	Is Next	Is Next	Yes
The GS1 is the global identification standards organization for retail.	Narnia is a fictitious country in a closet.	Not Next	Not Next	Yes
The illusion is similar to the Ames room, in which objects can also appear to roll against gravity.	The opposite phenomenon—an uphill road that appears flat—is known in bicycle racing as a false flat.	Is Next	Is Next	Yes
The illusion is similar to the Ames room, in which objects can also appear to roll against gravity.	In geometry, two geometric objects are perpendicular if they intersect at right angles.	Not Next	Is Next	No

Table 5: BERT Next Sentence Prediction Results

Overall Accuracy: 6/8 correct.

NSP Discussion: The model successfully identified all genuine continuations, but exhibited a false-positive bias on the negative pairs, predicting "Is Next" for statements that lacked logical flow. This failure occurs when the false sentence falls within the same semantic domain as the first sentence (e.g., pairing "Ames room/gravity" with "geometry/perpendicular"). The NSP classification head is likely tricked by the high self-attention scores between these conceptually related tokens, incorrectly assuming sequential continuity despite the lack of logical coherence.

2.3 Understanding Questions

Generative vs. Discriminative Training: Discriminative models learn the conditional probability distribution $p(Y|X)$ to map input features X directly to a specific label Y. In contrast, generative models estimate the joint probability distribution $p(X, Y)$ in an unsupervised context. Generative pretraining is highly effective for downstream tasks because by learning to model the underlying structure of $p(X)$—such as the syntax and semantics of the English language—the network acquires rich, generalized feature representations. These

robust embeddings make the subsequent learning of the discriminative boundary $p(Y|X)$ significantly more sample-efficient during fine-tuning.

MLM vs. Autoregressive: BERT utilizes a Masked Language Modeling (MLM) objective, which allows the network to build deep, bidirectional representations by conditioning on both the left and right context simultaneously. Conversely, GPT uses an autoregressive (AR) objective, computing the probability of the next token strictly conditioned on previous tokens in a causal, left-to-right manner. While MLM yields richer contextual embeddings for classification tasks, it is fundamentally unsuitable for text generation. Generating text is an inherently sequential process where future context does not yet exist; MLM cannot be utilized iteratively because it requires the full sequence to be visible to resolve its attention matrices.

The [CLS] Token in BERT vs. ViT: Despite operating on entirely different modalities, the `[CLS]` token serves an identical architectural purpose in both BERT and the Vision Transformer (ViT). In both models, the token acts as a learnable, sequence-level aggregator. Because the self-attention layers process all input elements simultaneously, the `[CLS]` token attends to all word piece embeddings (in BERT) or spatial patch embeddings (in ViT), pooling global contextual information into a single fixed-length vector. This aggregated embedding is then projected through a final classification head, whether that is BERT's Next Sentence Prediction head during pretraining or ViT's image classification layer.

3 Task 3: Fine-tuning BERT for Sentiment Analysis

3.1 Fine-tuning DistilBERT on SST-2

```
# Load Data & Tokenizer
sst2 = load_dataset("glue", "sst2")
tokenizer = DistilBertTokenizer.from_pretrained("distilbert-base-uncased")

def tokenize_fn(batch):
    return tokenizer(batch["sentence"], padding="max_length", truncation=True, max_length=128)

train_ds = sst2["train"].map(tokenize_fn, batched=True)
val_ds = sst2["validation"].map(tokenize_fn, 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=32, shuffle=True)
val_loader = DataLoader(val_ds, batch_size=32)

# Fine-tune Pretrained Model
print("Fine-tuning Pretrained DistilBERT...")
pretrained_model = DistilBertForSequenceClassification.from_pretrained("distilbert-base-uncased", num_labels=2).to(device)
_, pre_loss, pre_acc = train_model(pretrained_model, train_loader, val_loader, device)

def train_model(model, train_loader, val_loader, device, epochs=3):
    """Training loop for fine-tuning or training from scratch."""
    loss_history = []
    acc_history = []

    optimizer = torch.optim.AdamW(model.parameters(), lr=2e-5, weight_decay=0.01)

    for epoch in range(epochs):
        model.train()
        total_loss = 0
        for batch in train_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, labels=labels)
            loss = outputs.loss

            loss.backward()
            optimizer.step()
            optimizer.zero_grad()
```

```
            total_loss += loss.item()

        avg_loss = total_loss / len(train_loader)
        loss_history.append(avg_loss)

        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)

        epoch_acc = correct / total
        acc_history.append(epoch_acc)

        print(f"Epoch {epoch + 1}/{epochs} | Loss: {avg_loss:.4f} | Val
    Accuracy: {epoch_acc:.4f}")

    return model, loss_history, acc_history
```

Listing 4: DistilBERT Fine-Tuning Script

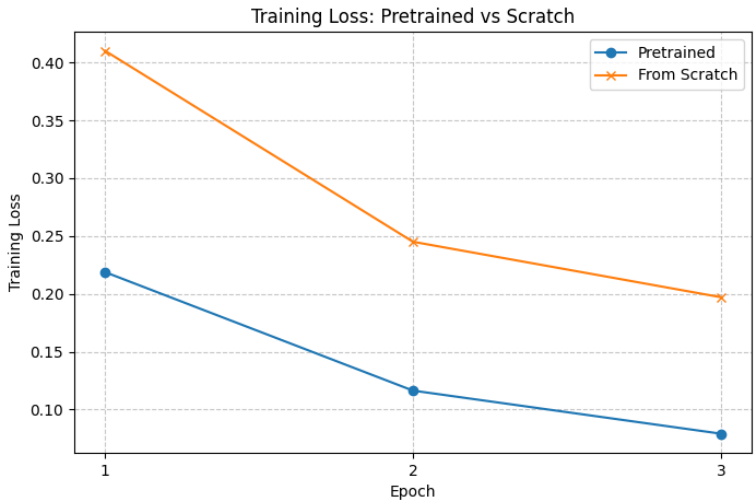

Figure 1: Training Loss Curve for DistilBERT over 3 epochs.

Validation Accuracies:

- Epoch 1: 0.9048
- Epoch 2: 0.9060
- Epoch 3 (Final): 0.8991

Example Predictions:

- True: 1 — Pred: 1 — Conf: 0.9998 — Text: it 's a charming and often affecting journey .
- True: 0 — Pred: 0 — Conf: 0.9755 — Text: unflinchingly bleak and desperate
- True: 1 — Pred: 1 — Conf: 0.9994 — Text: allows us to hope that nolan is poised to embark a major career as a com...
- True: 1 — Pred: 1 — Conf: 0.9666 — Text: the acting , costumes , music , cinematography and sound are all astound...
- True: 0 — Pred: 0 — Conf: 0.9984 — Text: it 's slow – very , very slow .
- True: 1 — Pred: 1 — Conf: 0.9995 — Text: although laced with humor and a few fanciful touches , the film is a ref...
- True: 0 — Pred: 0 — Conf: 0.9983 — Text: a sometimes tedious film .
- True: 0 — Pred: 0 — Conf: 0.9724 — Text: or doing last year 's taxes with your ex-wife .
- True: 1 — Pred: 0 — Conf: 0.5263 — Text: you don't have to know about music to appreciate the film 's easygoing ...
- True: 1 — Pred: 0 — Conf: 0.6548 — Text: we root for (clara and paul) , even like them , though perhaps it 's a...

3.2 Comparison: Pretrained vs. Random Initialization

```
# Train from Scratch
print("Training DistilBERT from scratch...")
config = DistilBertConfig(vocab_size=30522, num_labels=2)
scratch_model = DistilBertForSequenceClassification(config).to(device)
_, scr_loss, scr_acc = train_model(scratch_model, train_loader, val_loader
    , device)

def train_model(model, train_loader, val_loader, device, epochs=3):
    """Training loop for fine-tuning or training from scratch."""
    loss_history = []
    acc_history = []

```

```
    optimizer = torch.optim.AdamW(model.parameters(), lr=2e-5,
weight_decay=0.01)

    for epoch in range(epochs):
        model.train()
        total_loss = 0
        for batch in train_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,
labels=labels)
            loss = outputs.loss

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

        avg_loss = total_loss / len(train_loader)
        loss_history.append(avg_loss)

        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)

        epoch_acc = correct / total
        acc_history.append(epoch_acc)

        print(f"Epoch {epoch + 1}/{epochs} | Loss: {avg_loss:.4f} | Val
Accuracy: {epoch_acc:.4f}")

    return model, loss_history, acc_history
```

Listing 5: DistilBERT From-Scratch Script

Model	Pretrained?	Params	Epoch 1 Acc	Epoch 3 Acc
DistilBERT fine-tuned	Yes	66,955,010	0.9048	0.8991
DistilBERT from scratch	No	66,955,010	0.8131	0.8016

Table 6: Comparison of Pretrained vs. Randomly Initialized DistilBERT

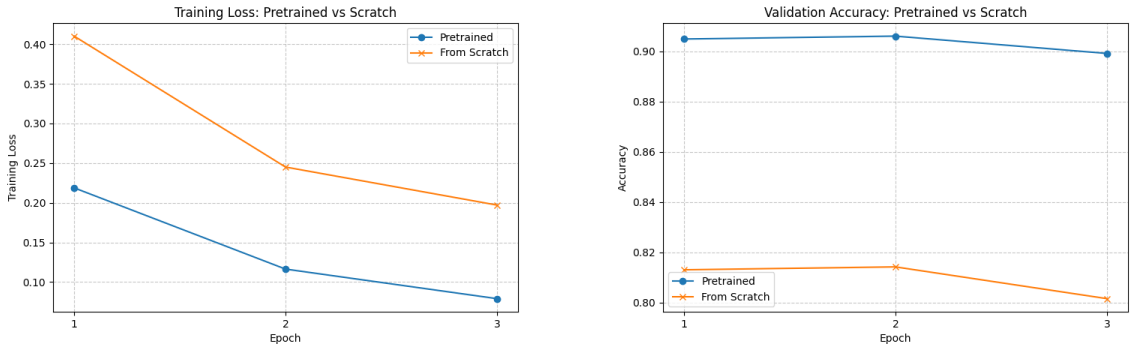

Figure 2: Left: Paired Training Loss Curves. Right: Paired Validation Accuracy Curves.

3.3 Analysis

The Accuracy Gap: The difference in validation accuracy between the pretrained model (90%) and the model trained from scratch (81%) was approximately 9 percentage points. The advantage for the fine-tuned model stems from its prior training on a massive, general-purpose text corpus. During this initial phase, the model learned the underlying structure of language. Fine-tuning then simply adjusted these weights for the sentiment domain. Interestingly, for the sentence "unflinchingly bleak and desperate," the fine-tuned model was much more confident (97%) than the scratch model (85%). Conversely, when the fine-tuned model was incorrect—such as on the sentence "you don't have to know about music"—it tended to be less confident (52%) than the scratch model (68%).

Parallel to HW9: If we conceptualize data as existing on a hyper-dimensional manifold, these models learn to identify the distributions of real-world data (pixels for ViT, tokens for BERT). Transfer learning works because the model has already converged on a steady-state distribution of general data during pretraining. Fine-tuning is essentially the process of finding a local distribution for a specialized set of points within that already-established manifold. We see performance gains because the model is sacrificing some general knowledge to specialize within a specific domain.

Data Scaling: Scaling the labeled data 10x or 100x would likely not close the gap entirely. A model trained only on sentiment labels—no matter how many—is only learning to map text to a binary signal. It misses the linguistic nuances (grammar, syntax, and word relationships) that BERT captures through its multifaceted self-supervised objectives. BERT's "head start" involves a fundamental understanding of language that sentiment labels alone cannot provide.

4 Bonus Task: babyGPT Autoregressive Training

4.1 Understanding Autoregressive Masking

The Mask Growth: The growing mask enforces the autoregressive property during training by restricting the decoder's visibility to only the previously generated tokens up to the current time step. By appending a '1' after each prediction step, the model progressively exposes one additional token of the ground-truth sequence, simulating the sequential generation process while preventing the model from looking ahead at future tokens to minimize loss. The mask must initialize with a length of 1 rather than 0 because the model requires at least one initial token to compute the initial self-attention necessary to predict the subsequent token.

No Cross-Attention: The babyGPT decoder omits the cross-attention layer because it functions as an autoregressive language model, which generates text solely on a prior sequence of tokens rather than translating a parallel sequence. In the encoder-decoder translation architecture, the encoder output provided a representation of the source language sentence, which the cross-attention layer leveraged to guide the target language generation. Because a autoregressive language model lacks a distinct source sequence to condition on, the model relies entirely on causal self-attention applied to the previously generated tokens within its own sequence.

Connection to GPT-3: The fundamental training mechanism between babyGPT and base GPT-3 is functionally identical, as both models utilize unsupervised autoregressive next-token prediction over a text corpus to minimize cross-entropy loss. However, GPT-3 benefits immensely from its massive scale which enables it to acquire a general understanding of complex linguistic structures and relationships inherent in the diverse training data. This scale allows GPT-3 to perform zero-shot and few-shot learning, a capability that a smaller model like babyGPT fundamentally lacks due to its highly constrained representational capacity.

4.2 Training babyGPT on the News Corpus

```
articles_dir = './saved_articles_dir_12M'
tokenizer_json = 'babyGPT_Examples/112_babygpt_tokenizer_cl_35035.json' # Use cleaned 35k tokenizer
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

optimizer_params = {'beta1' : 0.9,  'beta2': 0.98,  'epsilon' : 1e-6}
num_warmup_steps = 4000

masking = True


```

```
baby_gpt = babyGPT(
                    max_seq_length = max_seq_length,
                    batch_size = batch_size,
                    embedding_size = embedding_size,
                    num_basic_decoders = num_basic_decoders,
                    num_atten_heads = num_atten_heads,
                    optimizer_params = optimizer_params,
                    num_warmup_steps = num_warmup_steps,
                    masking = masking,
                    verify_text_corpus = False,
                    path_saved_model = {"decoder" : "./saved_decoder",
                                        "embedding_generator" : "./saved_embedding_generator",
                                       },
                  )

xformer = baby_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 = baby_gpt.MasterDecoderWithMasking(
                    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 = baby_gpt.ArticleDatasetWithBufferedContext(
                    gpt = baby_gpt,
                    tokenizer_json = tokenizer_json,
                    context_window_size = context_window_size,
                    context_buffer_size = context_buffer_size,
                    articles_dir = articles_dir,
             )

number_of_learnable_params_in_decoder = sum(p.numel() for p in master_decoder.parameters() if p.requires_grad)
print("\n\nThe number of learnable parameters in the Master Decoder: %d" % number_of_learnable_params_in_decoder)

print("""\n\n\nIf your training corpus is several megabytes in size (say over 100MB), it may take a few
      minutes to set up the token streams for the individual batch instances.

```

```
                                    PLEASE BE PATIENT\n""")

baby_gpt.run_code_with_buffered_context_for_training_TransformerFG(xformer
    , master_decoder, dataloader, checkpoint_frequency = 4000)
```

Listing 6: babyGPT Training Script

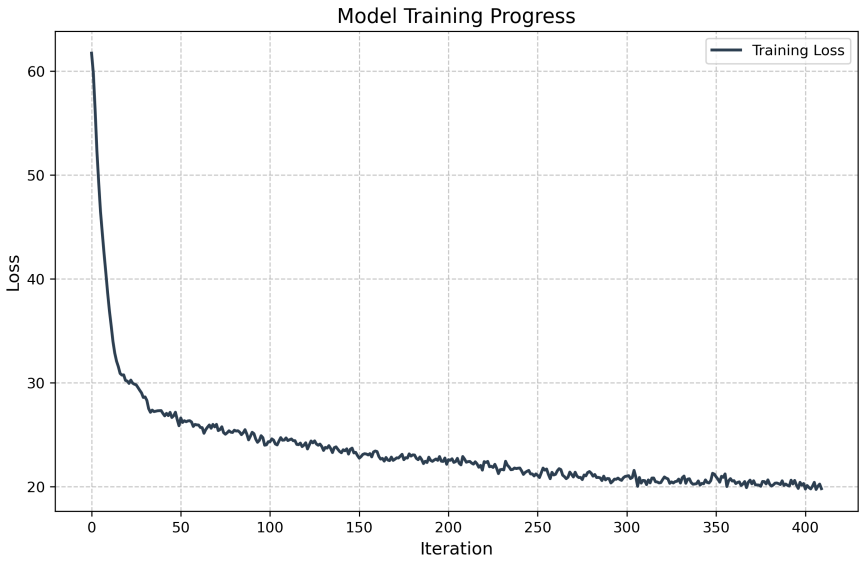

Figure 3: Training Loss for babyGPT Model.

Ground-Truth vs. Prediction: At epoch 5, the model produces: Ground-Truth: ter ' s biological dad does not live locally and Detokenized: ter s thelit sble _and and to be tocay to

At epoch 9, the model produces: Ground-Truth: the wake of former President Donald Trump ' s felony convicti Detokenized: the first of the president andčd Trump s aw _and_

Training Behavior: As demonstrated, the training loss decreases over the course of training, starting at an initial value of approximately 60 and rapidly descending to around 30 within the first 50 iterations, before gradually converging to a steady-state value near 20. Concurrently, the performance of the model improves; while early epoch predictions are largely incoherent jumbles of common tokens, later epochs demonstrate an improved ability to generate sequences that resemble valid English word fragments and basic grammatical structures (e.g., predicting "the first of the president" given the ground-truth context).

4.3 Prompting Your Trained Model

Prompts and Responses:

- **In-Domain Prompt 1:** prompt: The first pick for the NFL Draft pick is detokenized sentence completion: The first pick for the NFL Draft pick is a bit of
- **In-Domain Prompt 2:** prompt: In order to hit the ball at that speed, batters sentence completion: In order to hit the ball at that speed , batters to

- **Out-of-Domain Prompt 1:** prompt: To cook a mean steak, you have to reverse
  sentence completion: To cook a mean steak , you have to reverse the end of the
- **Out-of-Domain Prompt 2:** prompt: To be happy, one must simply
  sentence completion: To be happy , one must simply of the world of the world
- **Single Word Prompt:** prompt: To
  sentence completion: To in the US the world of the world of the world in

Prompting Analysis: Analysis of the generated text reveals that the model performs equally on in-domain prompts compared to out-of-domain prompts, both outputs remain highly primitive. In-domain prompts (such as those related to the NFL and baseball) yielded completions containing valid grammatical fragments and sensible next-word predictions whereas out-of-domain prompts showed repetitive loops such as "of the world of the world". The outputs vaguely resemble the statistical structure of English but they completely lack long-term structural coherence. This highlights the limitations of babyGPT's constrained scale; unlike GPT-3, which possesses enough parameters to memorize complex world knowledge and abstract grammar patterns from massive corpora, babyGPT's limited parameter count restricts it to learning only the most superficial statistics of its specialized training set.