Ashesi University

Prosit 1: Building and Adapting Language Models

Technical Report

ICS554: Natural Language Processing

Author:

Elijah Kwaku Adutwum Boateng

February 4, 2026

Contents

1 Section A

1.1 Language Models and Their Use

At a fundamental level, language models are an attempt to endow machines with a workable form of linguistic competence. In human communication, understanding is revealed through some form of appropriate response: given a context, a competent speaker has expectations about what is likely, plausible, or surprising to say next. A language model adopts this same principle and formalizes it in probabilistic terms. Rather than claiming deep semantic understanding (which remains a philosophical debate), it measures comprehension through the ability to assign sensible expectations to linguistic continuations.

This intuition leads to a probabilistic formulation or model. Given a sequence of tokens $w_1, w_2, \ldots, w_T$, a language model assigns a joint probability $P(w_1, w_2, \ldots, w_T)$. The probabilities are not binary judgments of correctness, but a sort of graded sense of plausibility. For example, the sentence *"I went to the bank to table the watch"* should be assigned a much lower probability than *"I went to the bank to sit by the river"*, because the latter aligns more closely with common linguistic usage and background knowledge, although your immediate expectation of what should follow *I went to the bank* might not be about sitting by a river. If a model consistently makes such distinctions, it demonstrates a functional form of language understanding.

Under this perspective, next-token prediction seems the smallest measurable unit of linguistic expectation. Predicting what comes next, given context, is sufficient to encode syntax, semantics, and common techniques or regularities of language usage. Training a language model, therefore, consists of adjusting its parameters to maximize the likelihood of observed text, aligning the model's internal expectations with those implicit in the natural language. This probabilistic foundation underlies modern applications such as predictive text, machine translation, speech recognition, and conversational agents, where appropriate action depends on anticipating plausible continuations.

1.2 N-gram Models and Their Working Principles

N-gram models are among the earliest and most interpretable approaches to language modeling. They rely on the *Markov assumption*, which states that the probability of a token depends only on a fixed number of preceding tokens. Specifically, an n-gram model approximates the full joint distribution as

$$P(w_1, \ldots, w_T) \approx \prod_{t=1}^{T} P(w_t \mid w_{t-n+1}, \ldots, w_{t-1}). \tag{1}$$

For example, in a trigram model ($n = 3$), the probability of the word w_t is conditioned only on the two immediately preceding words.

Parameter estimation in n-gram models is performed using maximum likelihood estimation (MLE). For a trigram model, the conditional probability is computed as

$$P(w_t \mid w_{t-2}, w_{t-1}) = \frac{\text{Count}(w_{t-2}, w_{t-1}, w_t)}{\text{Count}(w_{t-2}, w_{t-1})}. \tag{2}$$

This counting-based formulation makes n-gram models transparent and easy to debug, which is particularly valuable in low-resource settings where model behavior must be carefully inspected.

However, n-gram models suffer from two fundamental limitations. First, their context window is fixed and short; long-range dependencies that span many words are entirely ignored. Second, the number of possible n-grams grows exponentially with n, leading to severe data sparsity. In practice, this makes models beyond trigrams unreliable unless trained on extremely large corpora. These limitations motivate the transition from purely count-based models to neural approaches that can represent broader context more efficiently.

1.3 Evaluation of Language Models

Once a language model has been trained, the natural question is whether it has actually learned sensible expectations about language. Since the model itself is probabilistic, its evaluation should reflect how well these expectations align with real, unseen text. The most common intrinsic metric used for this purpose is *perplexity*, which measures how surprised the model is by data it did not observe during training.

Formally, perplexity is defined as

$$\mathrm{PP} = \exp\left(-\frac{1}{T}\sum_{t=1}^{T}\log P(w_t \mid w_1, \dots, w_{t-1})\right). \tag{3}$$

Rather than viewing this purely as a mathematical statement, it is helpful to interpret perplexity with some intuition. A perplexity of k can be understood as the model behaving as though it is choosing among k equally likely options at each step. Lower perplexity, therefore, indicates that the model's expectations are sharper and better aligned with the structure of the language.

However, perplexity alone does not tell the full story. A model may achieve low perplexity but still behave poorly when integrated into real world systems. In practice, a model can assign high probability to locally plausible words(based on the training set) while producing outputs that are repetitive and uninformative. For these reasons and more, language models are also evaluated extrinsically through specific tasks or applications that reflect its practical use. For example, in machine translation, performance may be measured using BLEU scores, while in speech systems, word error rate (WER) is more appropriate. In all cases, a evaluation criteria or process that separates training, validation, and test data to ensure that reported results reflect genuine generalization rather than memorization.

1.4 Handling Missing and Unseen Words During Inference

A practical issue that arises in n-gram language modeling is the problem of unseen events. Because maximum likelihood estimation is based entirely on observed counts, any n-gram that does not appear in the training data is assigned a probability of zero. This is means that a single unseen n-gram is sufficient to reduce the probability of an otherwise reasonable sentence to zero.

Smoothing techniques are introduced to address this issue by redistributing probability mass from frequent events to rare or unseen ones. One of the simplest approaches is the add-k smoothing, in which counts are adjusted as

$$P(w_t \mid w_{t-1}) = \frac{\mathrm{Count}(w_{t-1}, w_t) + k}{\mathrm{Count}(w_{t-1}) + kV}, \tag{4}$$

where V denotes the vocabulary size. While this guarantees non-zero probabilities, it often assigns too much probability to implausible events, particularly when the vocabulary is large. For example, suppose $\mathrm{Count}('the') = 100$, the vocabulary size is $V = 10{,}000$, and $k = 1$. An unseen word 'prosit' then receives

$$P('prosit' \,|'\, the') = \frac{1}{100 + 10{,}000} \approx 10^{-4}, \tag{5}$$

which may exceed the probability of genuinely rare but plausible words with small empirical counts. As V increases, this uniform allocation increasingly distorts the probability distribution.

More effective approaches recognize that different context lengths carry different levels of reliability. Backoff methods use higher-order n-gram statistics when they are available, but fall back to lower-order models when they are not. Interpolation adopts a clever strategy of combining multiple context lengths simultaneously:

$$P(w_t \mid w_{t-2}, w_{t-1}) = \lambda_3 P(w_t \mid w_{t-2}, w_{t-1}) + \lambda_2 P(w_t \mid w_{t-1}) + \lambda_1 P(w_t), \tag{6}$$

with $\lambda_1 + \lambda_2 + \lambda_3 = 1$.

Conceptually, smoothing does not alter the objective of language modeling. The goal remains to assign sensible probabilities to linguistic continuations. How smoothing us is that, it allows the model to express uncertainty cleverly or maybe mercifully instead of failing outright when confronted with unseen input. This emphasis on robustness anticipates later neural approaches, where uncertainty is handled implicitly through learned representations rather than explicit count manipulation.

1.5 Large Language Models (LLMs) and What They Are Used For

Large language models grow out of the same basic idea as earlier language models: that a useful notion of linguistic competence can be captured through expectation. Given some context, the model should have a sense of what kinds of words, phrases, or continuations would reasonably follow. In this sense, LLMs do not introduce a new objective. They are still trained to predict the next token in a sequence, using probability as a way of expressing how plausible different continuations are.

What changes in large language models is the scale at which this expectation-building process works. With very large parameter counts, massive training corpora, and larger context windows, these models are able to accumulate and refine expectations across a much wider range of linguistic situations. Over time, this allows them to internalize patterns that are difficult to capture with smaller models, such as long-range dependencies, subtle semantic cues, and regularities that span sentences or even entire documents. The result is a model whose predictions reflect not just local word co-occurrences, but a broader sense of how language tends to unfold in practice.

Because these learned expectations are general rather than task-specific, LLMs can be applied to many different problems without being redesigned each time. The same underlying model can generate text, answer questions, summarize documents, translate between languages, or engage in dialogue, simply by being conditioned on different prompts. In effect, what makes LLMs powerful is not that they have learned individual tasks, but that they have learned a flexible, scalable set of expectations about language itself.

1.6 Typical Architectures for Large Language Models

As language models grew larger, the main question gradually shifted from whether probability was the right foundation to how much context a model could realistically take into account, and how efficiently it could do so. Earlier statistical models relied on short, explicitly defined contexts as discussed in the ngram section, while early neural models attempted to compress growing histories into a single evolving representation (the case of RNNs). These approaches worked to some extent, but as datasets and model sizes increased, their limitations became more apparent, particularly in terms of training stability and scalability.

The Transformer architecture addresses this issue by cleverly redesigning how context is accessed. Instead of summarizing past information into a single state, Transformers allow each token to look back over the entire available context and decide, in a data-driven way, which parts matter most for the current prediction. This is achieved through self-attention, which computes relationships between tokens directly rather than passing information step by step through time. As a result, long-range dependencies become easier to model, and training can be parallelized efficiently, which is crucial at the scale required by modern LLMs.

Within the Transformer framework, different architectural variants emerge depending on how the next-token prediction objective is used. Decoder-only models generate text autoregressively, producing one token at a time while continuously updating their expectations based on everything generated so far. This makes them especially well suited for open-ended generation and interactive settings. Encoder-only models, by contrast, focus on building rich contextual representations of entire sequences, which is useful for understanding-oriented tasks. Encoder–decoder models sit somewhere in between, learning to map one sequence to another in a structured way. Across all these architectures, the underlying principle remains the same: the model is learning how to use context more effectively, not redefining what it means to model language.

At a more technical level, the architectural differences between these models come down to how attention is structured and where queries, keys, and values are drawn from. In self-attention, each token produces a query (Q), key (K), and value (V) vector, and relevance is computed by comparing queries against keys to decide how information should be aggregated from the values. When all three(Q, K, and V) are derived from the same sequence, the model is performing self-attention, learning relationships within that sequence itself.

In decoder-only architectures, this self-attention is further constrained by a causal, or masked, attention mechanism. The attention mask ensures that each token can only attend to earlier tokens in the sequence, preventing access to future information during training and inference. This constraint aligns naturally with the next-token prediction objective, since the model must form expectations based solely on what has already been generated.

Encoder-only models also rely on self-attention, but without causal masking. Every token is allowed to attend to every other token in the sequence, which makes these models well suited for building holistic representations of input text rather than generating it sequentially. Encoder–decoder architectures introduce an additional component: cross-attention. In this case, queries come from the decoder, while keys and values are provided by the encoder's representations of the input sequence. This allows the decoder to condition its predictions not only on previously generated tokens, but also on a structured representation of an external input.

Viewed this way, the architectural differences are less about changing the learning objective and more about controlling how information flows through attention. Masked self-attention enforces temporal ordering, unmasked self-attention supports global understanding, and cross-attention enables conditional generation. Large language models like the GPTs primarily adopt the decoder-only variant because masked self-attention scales cleanly with data and model size while remaining aligned with the probabilistic next-token prediction task idea.

1.7 LLM Decoding

Once a language model has learned to assign probabilities to possible next tokens, decoding is the process of turning those probabilities into an actual sequence of words. At each step, the model produces a probability distribution over the vocabulary conditioned on the context so far. Decoding is simply the decision rule that determines which token is selected from that distribution and appended to the sequence. The model itself does not generate text; it only expresses expectations so we use decoding to decide how these should be turned into word sequences.

This distinction is important because the learned model and the generated text are not the same thing. The same trained LLM can produce very different outputs depending on how decoding is performed, even though the underlying probabilities remain unchanged. In this sense, decoding can be thought of as decision-making under uncertainty: given a graded sense of what is plausible, the system must choose how conservative or exploratory it should be when committing to a specific continuation.

1.8 Decoding Strategies Used in LLMs

The simplest decoding strategy is greedy decoding, where the model selects the most probable token at each step. While this aligns closely with maximizing local likelihood, it often leads to repetitive text, because the model continually follows the single most obvious path. Beam search extends this idea by maintaining multiple high-probability candidate sequences in parallel, allowing the model to consider alternative continuations before committing. This improves global coherence but still tends to favor safe, high-probability phrasing.

There is also sampling strategies. Sampling-based strategies take a different approach by treating the model's probability distribution as something to sample from rather than optimize greedily. Techniques such as temperature scaling adjust how sharp or flat the distribution is, controlling how strongly the model prefers high-probability tokens. Top-k and nucleus (top-p) sampling further restrict sampling to a subset of plausible candidates, balancing coherence with diversity. In practice, these methods acknowledge that language often admits multiple reasonable continuations, and decoding becomes a way of exploring that space rather than collapsing it prematurely to a single most-likely outcome.

So for example, consider a trained language model given the prompt *"Dennis is a great"*. At this point, the model produces a probability distribution over possible next tokens, for example:

$$P(\text{lecturer}) = 0.35, \quad P(\text{teacher}) = 0.30, \quad P(\text{researcher}) = 0.20, \quad P(\text{idea}) = 0.10, \quad P(\text{car}) = 0.05.$$

Under greedy decoding, the model selects the most probable token, producing *"Dennis is a great lecturer"*. Repeating this process at each step often leads to fluent but predictable continuations.

With beam search, the model keeps multiple high-probability continuations in parallel. For instance, it may retain both *"Dennis is a great lecturer"* and *"Dennis is a great teacher"* as candidate sequences, extending each before choosing the overall most likely sequence. This improves global coherence but still favors safe, high-probability phrasing.

Under sampling-based decoding, the next token is drawn from the probability distribution rather than selected deterministically. With temperature scaling or top-k sampling, the model might produce *"Dennis is a great researcher"* or *"Dennis is a great teacher"*, reflecting alternative but still plausible continuations. In this case, decoding allows the model to explore multiple reasonable completions rather than committing immediately to a single most likely outcome.

1.9 LLM Pre-training: Learning Expectations at Scale

Pre-training is the stage at which a language model acquires its general understanding of how language behaves or that observable linguistic competence I spoke about earlier. At this point, the model is not optimized for any specific task, but is simply exposed to very large collections of text and trained to predict the next token given some context. Through this repeated exposure, the model gradually internalizes regularities of language, including syntax, common patterns of reasoning, and broad associations between concepts. In effect, pre-training allows the model to build a wide base of expectations about what tends to follow what in natural language.

This stage became necessary because learning useful language representations from scratch for every task was not practical nor efficient. Without pre-training, a model fine-tuned on limited task-specific data would struggle to form basic expectations about the language structure itself and usage. Pre-training therefore provides a shared linguistic foundation, allowing fine-tuning to focus on shaping behavior rather than relearning language itself. Example, a ghanaian child who does not understand English cannot be taught Mathematics using the English language. The knowledge of maths is transferred or exchanged through the english language.

Technically, pre-training is performed by minimizing the negative log-likelihood of observed text, or equivalently, by maximizing the probability the model assigns to the training data. The same probabilistic objective used in smaller language models applies here, but at a much larger scale, with long context windows and billions of parameters. Because the training data spans many domains and styles, the resulting model learns representations that are general rather than task-specific.

Importantly, pre-training alone does not make a model directly useful in interactive settings. A pre-trained model may generate fluent text, but it does not naturally follow instructions or distinguish between helpful and unhelpful responses. Pre-training therefore provides a foundation of linguistic expectations, which must be further shaped before the model can be reliably deployed.

1.10 Instruction Tuning: From Completion to Following Instructions

Instruction tuning addresses a mismatch between what pre-trained language models learn and how they are expected to behave in practice. A pre-trained model is optimized to continue text, not to respond to requests, questions, or commands. Instruction tuning adapts the model to this setting by fine-tuning it on curated datasets of instruction–response pairs, teaching it how to condition its predictions on explicit user intent (the classical supervised dataset styles).

During instruction tuning, the underlying objective remains next-token prediction, but the training data is structured so that instructions and desired responses appear consistently together. Over time, the model learns to associate certain patterns of input with appropriate forms of output. Conceptually, instruction tuning does not add new knowledge to the model; instead, it reorganizes existing expectations so that the model responds in a way that aligns more naturally with human interaction.

1.11 LLM Alignment: Motivation and Mechanism

Once a language model has learned broad expectations about language from large-scale data, an important question arises: are these expectations aligned with what humans actually want from the system? A pre-trained or even instruction-tuned model may be fluent and knowledgeable, yet still produce outputs that are misleading, unsafe, biased, or simply unhelpful. This happens because the model is fundamentally optimizing for likelihood, not for human values or intent. Alignment is therefore needed to bridge the gap between what the model finds statistically plausible and what humans consider appropriate or desirable in real-world use.

In practice, alignment is commonly achieved through Reinforcement Learning from Human Feedback (RLHF). The idea is to expose the model to human judgments about its outputs, rather than relying solely on raw text data. Humans are asked to rank or compare different responses produced by the model, and these preferences are used to train a reward model that captures what humans tend to favor. The language model is then fine-tuned using reinforcement learning to maximize this learned reward, while remaining close to its original behavior. Conceptually, alignment does not change the model's understanding of language, but reshapes its expectations so that high-probability outputs are also more helpful, safe, and aligned with human intent.

1.12 LLM Fine-Tuning: Purpose and Process

While large language models acquire broad linguistic skills during pre-training, they are not automatically well suited to specific tasks or domains. Fine-tuning is the process of adapting a pre-trained model to a narrower objective, dataset, or usage context. The motivation is practical: a general-purpose model may know a little about many things, but real applications often require consistent behavior, domain-specific terminology, or a particular response style.

Fine-tuning typically involves continuing training on a smaller, task-specific dataset while using a much lower learning rate than in pre-training. This allows the model to adjust its parameters without overwriting the general language understanding it has already learned. Depending on the setting, fine-tuning may target all model parameters or only a subset of them. Parameter-efficient methods such as low-rank adaptation introduce small trainable components while keeping most of the original model fixed, reducing computational cost and the risk of overfitting.

From a conceptual standpoint, fine-tuning can be seen as sharpening the model's expectations rather than rebuilding them. The probabilistic objective remains the same but the distribution is nudged toward behaviors that are more appropriate for a given task, domain, or deployment context. This makes fine-tuning a key step in turning a broadly capable language model into a reliable component of a real system.

For example, after pre-training, a model may simply learn to produce fluent language and generate a continuation such as *"A crop is something that grows in the ground"*. Instruction tuning then teaches the model to recognize when this information is being requested, allowing it to respond appropriately to a question like *"What is a crop?"*. Fine-tuning further refines the response, shaping it to be correct and context-specific, such as explaining a crop in agricultural, economic, or scientific terms depending on the application.

1.13 Random Interesting things I learned

One of the most striking takeaways from studying language models is how modern techniques give concrete form to long-standing ideas about meaning and context. I came across a quote by J. R. Firth's (a british linguist) that said "you shall know a word by the company it keeps". This quote is operational in neural language models through embeddings and attention mechanisms. Words, subwords, and phrases acquire meaning not from predefined rules, but from the contexts in which they repeatedly appear. This shifts language understanding away from symbolic definitions toward a more mathematical or more precisely a geometric view, where relationships between these linguistic units evolve as structures in high-dimensional space (inspiring word2vec and other embedding schemes). From this perspective, context is no longer something manually specified, as in n-grams, but something learned, weighted, and reused in very dynimac contexts.

Another important insight is how scale fundamentally changes what these models are capable of doing. Empirical scaling laws show that performance improves predictably as model size, data, and compute increase, suggesting that many abilities are not explicitly engineered but emerge once models reach sufficient capacity. This is closely connected to in-context learning, where a model can adapt to new tasks simply by observing examples within the prompt, without any parameter updates. At the same time, seemingly low-level design choices, such as tokenization, play a surprisingly large role in shaping what a model can learn, influencing efficiency, generalization, and performance on different languages or domains. Taken together, these observations suggest that the power of large language models lies not in any single component, but in the interaction between probabilistic learning, representation, scale, and context—an interaction that continues to raise both technical and conceptual questions about what it means for machines to "understand" language.

2 Section B: Low-Resource African Language Model (Twi)

2.1 Data Description

The language model was trained using monolingual Twi text extracted from the *tw-parallel-lg-corpus*, a publicly available dataset containing aligned Twi–English sentence pairs. Only the Twi portion of the corpus was used, as the task focused on monolingual language modeling rather than translation. The corpus was preprocessed through lowercasing, normalization, and tokenization, with explicit start-of-sentence (`<s>`) and end-of-sentence (`</s>`) markers added to preserve sentence boundaries and enable proper context initialization for higher-order n-gram models.

The final dataset exhibits typical low-resource characteristics: limited corpus size, a vocabulary of approximately 22,080 unique word types, and a highly skewed Zipfian frequency distribution with a long tail of rare tokens. These properties introduce significant sparsity, making the modeling task particularly sensitive to smoothing and data efficiency considerations.

2.2 Modeling Choice: N-gram Models versus Neural Language Models in Low-Resource Settings

In low-resource language modeling, the primary constraint is the mismatch between model capacity and available data. Modern neural language models, including recurrent architectures and transformers, contain millions of parameters that must be estimated from data. Empirical scaling laws demonstrate that effective neural language model training typically requires between 10 and 100 tokens per parameter to achieve stable generalization [3, 4]. Even relatively small transformer architectures therefore demand tens of millions of tokens, far exceeding the size of the available Twi corpus.

Training neural models under such conditions often leads to overfitting, unstable optimization, and misleading performance metrics. While transfer learning and multilingual pretraining can mitigate these issues, they introduce additional complexity, computational cost, and reduced interpretability. Moreover, neural representations are opaque, making it difficult to diagnose failure modes or understand how linguistic structure is encoded.

In contrast, n-gram language models are explicitly count-based and far more data-efficient. Although the theoretical parameter space of an n-gram model scales exponentially with vocabulary size, the *observed* parameter count is bounded by the number of n-grams actually present in the corpus, which grows approximately linearly with corpus size. Crucially, advanced smoothing techniques—particularly Kneser–Ney smoothing—provide strong inductive bias by redistributing probability mass according to continuation diversity rather than raw frequency [1, 2]. Empirical studies have shown that smoothed n-gram models can outperform neural models in perplexity when training data is limited to fewer than 10 million tokens [5]. Given the size and characteristics of the Twi corpus, n-gram modeling therefore represents a principled, interpretable, and empirically justified choice.

2.3 Training Methodology and Evidence of Learning

Training the language model involved constructing n-gram count tables from the training split of the corpus and estimating conditional probabilities of the form $P(w|h)$, where h represents an $n-1$ token history. Models were trained across multiple n-gram orders (from unigram to 5-gram) and evaluated using several smoothing strategies, including Laplace smoothing (LP), linear interpolation (IP), and Kneser–Ney smoothing (KN). The training pipeline was implemented in a modular, object-oriented manner to ensure reproducibility and facilitate systematic experimentation.

Evidence that the model was learning meaningful linguistic structure emerged from consistent trends in validation perplexity. As n-gram order increased, unsmoothed and Laplace-smoothed models rapidly deteriorated due to data sparsity, while interpolated and Kneser–Ney models showed substantial and stable improvements. This pattern indicates that performance gains were driven by effective use of contextual information rather than memorization of training sequences.

Further confirmation came from qualitative analysis of generated text. Lower-order models produced incoherent and repetitive sequences, whereas higher-order Kneser–Ney models generated locally grammatical and contextually appropriate phrases. The alignment between quantitative improvements and qualitative fluency provides strong evidence that the model learned statistically meaningful regularities from the data.

2.4 Evaluation Strategy

Model evaluation followed a two-stage process. First, all candidate models were evaluated on a held-out validation set using perplexity, which measures the model's average uncertainty when predicting unseen tokens. Validation perplexity was used to select the best-performing configuration. Second, the selected model was evaluated on a separate test set to assess generalization performance.

Qualitative evaluation complemented perplexity-based analysis through manual inspection of generated samples. Generated sequences were assessed for grammaticality, lexical appropriateness, and contextual coherence, ensuring that numerical improvements translated into meaningful linguistic quality. We realized that although the sentences were grammatically fine and sounded fairly plausible with respect to the Twi language and sometimes contextually right, they were not always Biblically accurate (given the religious nature of much of the training corpus).

2.5 Results

2.5.1 Quantitative Results

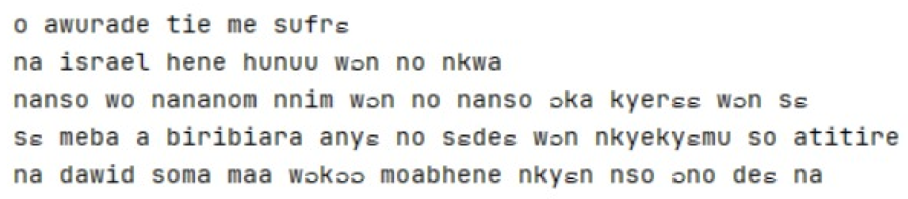
Table 1: N-gram Model Performance Comparison

Model	N-gram Order	Smoothing	Val. Perplexity	Training Time (s)
4-gram KN	4	KN	**110.65**	20.72
5-gram KN	5	KN	112.52	41.67
3-gram KN	3	KN	113.44	10.32
2-gram KN	2	KN	148.96	4.58
2-gram IP	2	IP	161.92	4.54

The best-performing model was a **4-gram Kneser–Ney language model**, achieving a validation perplexity of 110.65. When evaluated on the held-out test set, this model achieved a test perplexity of **108.81**, confirming strong generalization. Although the 5-gram Kneser–Ney model performed competitively, its marginal improvement did not justify the increased computational cost and sparsity, indicating that the 4-gram configuration represents the optimal balance between contextual richness and data reliability.

2.5.2 Qualitative Results

Figure 1 shows sample text generated by our Twi n-gram model using weighted sampling decoding.



```
o awurade tie me sufrɛ
na israel hene hunuu wɔn no nkwa
nanso wo nananom nnim wɔn no nanso ɔka kyerɛɛ wɔn sɛ
sɛ meba a biribiara anyɛ no sɛdeɛ wɔn nkyekyɛmu so atitire
na dawid soma maa wɔkɔɔ moabhene nkyɛn nso ɔno deɛ na
```

Figure 1: Sample generated Twi text from our trigram model using weighted sampling. The last sentence references “David and the King of the Moabites,” which is contextually correct from a Biblical perspective.

The generated text demonstrates that the model learned meaningful linguistic patterns from the Twi corpus. While the sentences are grammatically plausible and contextually coherent within the religious domain of the training data, they are not always Biblically accurate—highlighting both the capabilities and limitations of statistical language models.

2.6 Additional Experimental Insights

Extensive additional experimentation was conducted beyond the final reported model. Both word-level and character-level tokenization strategies were explored. While character-level models eliminated out-of-vocabulary issues, they required substantially longer context windows to achieve comparable performance, making them less effective given the dataset size. Word-level tokenization provided a better balance between expressiveness and sparsity.

We also investigated the use of an `<UNK>` token to replace rare words. This approach was ultimately rejected because rare words constitute a large proportion of vocabulary types in Twi and often carry important semantic distinctions. Collapsing them into a single token degraded both perplexity and generation quality. Instead, we relied on Kneser–Ney backoff to handle rare events in a statistically principled manner.

Finally, a custom smoothing algorithm based on dynamic interpolation weights was explored. Although this approach outperformed naive smoothing methods, it was computationally expensive and unstable in low-context scenarios, leading to runtime errors and inferior performance compared to Kneser–Ney.

3 Section C: Domain-Specific English Models

This section details how we developed two domain-specific language models using parameter-efficient fine-tuning techniques: **AgricGPT** for agriculture and **MedicalLM** for healthcare.

3.1 Introduction and Data

We built two domain-specific models targeting distinct but equally impactful domains: agriculture and healthcare.

AgricGPT leverages the **AI4Agr/CROP-dataset** from Hugging Face, containing instruction-response pairs about crop management, pest control, soil health, and sustainable farming practices. We selected 5,000 English samples with a 90%/10% train/-validation split (4,500 training, 500 validation), using the CROP-benchmark (500 MCQ questions) as a held-out test set.

MedicalLM uses **medical abstracts sourced from Hugging Face** (TimSchopf/medical_abstracts), featuring biomedical literature with specialized terminology. The dataset was split into 90% training, 10% validation, with a completely separate test set. This domain enables comparison with the Section B n-gram text generation task while exploring neural approaches for domain adaptation.

Both domains were selected for their practical importance—agriculture impacts food security in developing regions, while healthcare represents a high-stakes domain where specialized language understanding is critical.

3.2 Fine-tuning Approach Selection

We considered three main approaches for developing domain-specific English models:

Table 2: Fine-tuning Approach Comparison

Approach	**Description**	**Pros**	**Cons**
Full Fine-tuning	Update all model parameters	Maximum adaptation	Requires massive compute; risk of catastrophic forgetting
LoRA	Add low-rank adapter matrices	Memory efficient; preserves base model	Moderate adaptation
QLoRA	LoRA + 4-bit quantization	Very memory efficient; enables consumer GPU training	Slight quality trade-off

3.2.1 AgricGPT: QLoRA on Phi-2

For the agriculture domain, we chose **QLoRA** (Quantized Low-Rank Adaptation) with Microsoft Phi-2 (2.7B parameters). This decision was driven by practical constraints: QLoRA enables training on consumer GPUs (T4 with 16GB VRAM), reduces memory requirements by approximately 75% through 4-bit quantization, and supports checkpoint saving for fault tolerance during long training runs. With this approach, only 1.53% of parameters are trainable (23.6M out of 1.54B), making it highly efficient.

3.2.2 MedicalLM: LoRA on DistilGPT-2

For the medical domain, we chose **LoRA** (without quantization) with DistilGPT-2. Out-of-memory errors prevented both full fine-tuning and the use of larger models. DistilGPT-2's smaller size (82M parameters) allowed standard LoRA without requiring quantization. This modern, parameter-efficient approach enabled systematic hyperparameter experimentation and provides a meaningful comparison with the Part B n-gram model on text generation tasks.

Both approaches demonstrate that **parameter-efficient fine-tuning is a practical solution** for domain adaptation when compute resources are limited.

3.2.3 Experimental Approach: Training from Scratch

Another angle of experimentation we explored was equipping a transformer with a foundational level of English competence by pretraining it on a subset of the Wikipedia corpus. The intention was not to obtain strong standalone performance, but rather to initialize a base model that could later be fine-tuned on agricultural domain data.

While the experiment did not progress as far as planned, this outcome was largely anticipated. From a technical standpoint, the transformer architecture requires substantial pretraining data and compute to learn meaningful linguistic representations, which we were unable to fully provide. In addition, limitations in time and computational resources restricted our ability to scale the training process or systematically tune hyperparameters.

As a result, the model did not develop the depth of language understanding needed to serve as a strong foundation for downstream fine-tuning, but the experiment nonetheless offered valuable insights into the practical constraints of training language models from scratch. This reinforced our decision to use parameter-efficient fine-tuning of existing pretrained models rather than training from scratch.

3.3 Training Process

3.3.1 AgricGPT Configuration

The agriculture model was trained with the following configuration:

Table 3: AgricGPT Training Configuration

Parameter	**Value**
Base model	Microsoft Phi-2 (2.7B parameters)
LoRA rank	16
LoRA alpha	32
Target modules	fc1, fc2, q_proj, k_proj, v_proj, dense
Learning rate	2×10^{-4} with warmup
Batch size	2 (effective 8 with gradient accumulation)
Epochs	3
Precision	FP16 mixed precision

Parameter Efficiency:

Parameter Type	**Count**	**Percentage**
Trainable	23,592,960	**1.53%**
Frozen	1,521,392,640	98.47%
Total	1,544,985,600	100%

Prompt Template: We used an instruction-following format to structure the training data:

```
### Instruction:
{question}

### Response:
{answer}
```

This template teaches the model to recognize the instruction format and generate appropriate domain-specific responses. The base Phi-2 model, not having been trained on this format, tends to continue generating in unexpected directions rather than providing focused answers.

3.3.2 MedicalLM Configuration and Hyperparameter Search

For the medical model, we conducted systematic experimentation across many configurations:

Table 4: MedicalLM Hyperparameter Experiments

Experiment	**Rank**	**Learning Rate**	**Epochs**	**Val. Perplexity**
1	8	1e-4	3	23.89
2	16	1e-4	3	24.01
3	8	5e-5	3	23.67
4 (Best)	**8**	**5e-5**	**5**	**23.42**

The best configuration used r=8, alpha=32, lr=5e-5, and 5 epochs. A key insight emerged: higher rank (r=16) did not improve results as DistilGPT-2 is too small to benefit from increased adapter capacity. Lower learning rates proved more stable, and additional epochs improved performance without causing overfitting.

3.3.3 Evidence of Learning

Both models demonstrated clear evidence of learning through quantitative and qualitative measures.

AgricGPT showed substantial improvement:

- Training loss decreased from 2.05 to 0.71 (65% reduction)
- Validation loss decreased from 1.36 to 0.99 (27% reduction)

- CROP-benchmark accuracy improved from 73.2% to 82.2% (+9 percentage points)

MedicalLM exhibited consistent improvement:

- Training/validation loss decreased from 3.47 to 3.27
- Validation perplexity improved from 24.83 to 23.42 (5.7% improvement)
- Next-token accuracy improved across all k values

Qualitative Comparison:
MedicalLM Before (Baseline):

> "Sample 1: The patient presented with the diagnosis was taken to the hospital and transferred to a hospital where he was transferred to a hospital where he was discharged from the hospital."

MedicalLM After (Fine-tuned):

> "The patient presented with a severe acute liver infection. Three patients (7) were treated for fever, blood pressure and abdominal pain in the first 2 weeks of follow-up..."

AgricGPT Before vs After on a real agricultural query:
Question: "I've noticed some unusual spots on my maize leaves that are turning gray and then brown. The plants are also looking less vigorous. What could be causing this?"
Base Phi-2 (Before):

> "Some common signs of maize streak virus include yellowing and mottling of leaves, stunted growth, and reduced yield. The virus can also cause the leaves to turn brown and dry up... Instruction: How can I tell if my maize plants are..."

Note how the base model provides a generic response about maize streak virus (incorrect diagnosis for gray-brown spots) and then continues generating a new instruction prompt rather than stopping appropriately.
AgricGPT (After):

> "For immediate control, applying a fungicide can be effective. Products containing tricyclazole or propiconazole are commonly used for Gray Leaf Spot. However, it's crucial to rotate fungicides to prevent resistance development. Have you used any fungicides in the past few years?"

The fine-tuned model demonstrates more accurate domain knowledge—correctly identifying the symptoms as **Gray Leaf Spot** (a fungal disease) rather than maize streak virus, providing actionable treatment recommendations with specific fungicide names, and appropriately ending with a follow-up question rather than generating spurious content.

3.4 Evaluation Methodology

We employed complementary evaluation approaches tailored to each model's use case.

3.4.1 AgricGPT Evaluation

The primary metric was **MCQ accuracy** measured on the CROP-benchmark—an official benchmark from AI4Agr containing 500 English questions at varying difficulty levels (Level 0=Hard, Level 1=Medium, Level 2=Easy). This directly measures the model's ability to answer domain-specific questions correctly, the primary use case for an agricultural Q&A system. Additionally, we tracked perplexity on the validation set and monitored training/validation loss throughout training to detect overfitting.

3.4.2 MedicalLM Evaluation

We measured both **perplexity** (validation and test sets) and **next-token top-k accuracy** for k=1, 5, and 10. This combination proved critical—perplexity measures overall probability distribution fit, while accuracy measures ranking effectiveness. Qualitative assessment compared generations on standard medical prompts for terminology appropriateness and clinical coherence.

3.5 Results

3.5.1 AgricGPT Performance

Table 5: AgricGPT Training Metrics

Metric	**Initial**	**Final**
Training Loss	2.0458	0.7141
Validation Loss	1.3600	0.9917
Perplexity	—	**3.12**

Table 6: AgricGPT Accuracy Comparison

Model	**Accuracy**	**Improvement**
Base Phi-2	73.20%	—
AgricGPT	**82.20%**	**+9.0%**

Performance varied significantly by question difficulty:

Table 7: AgricGPT Performance by Difficulty Level

Level	**Description**	**Accuracy**	**Questions**
0	Hard	28.07%	16/57
1	Medium	85.41%	240/281
2	Easy	**95.68%**	155/162

The model excels on medium and easy questions (85–96%) but struggles with hard questions (28%), suggesting that complex reasoning tasks require more training data or advanced techniques.

3.5.2 MedicalLM Performance

Table 8: MedicalLM Performance Metrics

Metric	Before	After	Change
Validation Perplexity	24.83	23.42	**-5.7%** ✓
Test Perplexity	549.56	928.12	**+69%** ×
Top-1 Accuracy	40.3%	42.2%	**+1.9pp** ✓
Top-5 Accuracy	61.0%	63.2%	**+2.2pp** ✓
Top-10 Accuracy	68.4%	70.4%	**+2.0pp** ✓

3.5.3 Critical Finding: Metric Divergence

A striking observation from MedicalLM is that **test perplexity worsened by 69%** while **accuracy improved**. This reveals a fundamental trade-off in domain adaptation:

- **Perplexity** measures overall probability distribution fit—the model assigns lower probabilities overall after fine-tuning
- **Accuracy** measures ranking effectiveness—the model ranks correct tokens higher

The fine-tuned model became **specialized**: better at prioritizing medical terms but disrupted in its general probability modeling. This suggests potential overfitting to training data patterns despite validation improvements. The divergence underscores the importance of using multiple complementary metrics rather than relying on a single measure.

3.6 Discussion and Insights

3.6.1 Key Lessons Learned

1. **Optimal LoRA configuration depends on model size.** Higher rank (r=16) did not help DistilGPT-2, but worked well for Phi-2. Smaller models may not benefit from increased adapter capacity—the additional parameters simply cannot be utilized effectively.

2. **Lower learning rates provide more stable training.** MedicalLM performed better with 5e-5 than 1e-4, producing smoother loss curves and better final performance.

3. **Single metrics can mislead.** MedicalLM's test perplexity degraded while validation perplexity, accuracy, and generation quality all improved. Evaluating domain-specific models requires multiple complementary metrics that capture different aspects of model capability.

4. **Domain specialization involves trade-offs.** Models can improve at task-specific performance while degrading general statistical modeling. Understanding what metrics actually measure versus what matters for deployment is crucial.

3.6.2 Comparison with Section B (N-gram Model)

Table 9: N-gram vs. Neural Model Comparison

Aspect	N-gram (Twi)	Neural (Medical)
Perplexity	167.53 (trigram: 84% sparsity)	23.42
Generation Quality	Incoherent	Coherent, domain-appropriate
Context Handling	Limited to n-1 words	Handles longer context
Adaptation Method	Smoothing	Parameter-efficient fine-tuning

A shared lesson emerges: **data quality and quantity remain central regardless of architecture**. N-grams suffer from sparsity with limited Twi text, while neural models show potential overfitting signs even with domain-specific data. Both approaches face the fundamental challenge of learning robust patterns from constrained resources.

3.6.3 Model Availability and Future Work

The AgricGPT model is publicly available at `https://huggingface.co/Ajegetina/agricgpt-phi2`.

Future directions include:

- Longer training with more diverse agricultural and medical data
- Data augmentation to address hard questions (AgricGPT achieves only 28% on Level 0)
- Regularization techniques to address test perplexity degradation in MedicalLM
- Exploration of knowledge distillation to combine domain expertise from multiple specialized models

3.7 Team Task Distribution

- **Elijah:** N-gram and transformer from scratch implementation
- **Joseph:** N-gram, Streamlit ngram demo, and pretrained model finetuning (AgricGPT) implementation
- **Nicole:** N-gram and pretrained model finetuning (MedicalLM) implementation
- **Innocent:** N-gram implementation

References

[1] S. F. Chen and J. Goodman, "An Empirical Study of Smoothing Techniques for Language Modeling," *Computer Speech & Language*, vol. 13, no. 4, pp. 359–394, 1999.

[2] K. Heafield, "KenLM: Faster and Smaller Language Model Queries," in *Proc. Sixth Workshop on Statistical Machine Translation*, 2011.

[3] J. Kaplan *et al.*, "Scaling Laws for Neural Language Models," *arXiv preprint arXiv:2001.08361*, 2020.

[4] J. Hoffmann *et al.*, "Training Compute-Optimal Large Language Models," *arXiv preprint arXiv:2203.15556*, 2022.

[5] C. Chelba *et al.*, "One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling," in *Proc. INTERSPEECH*, 2013.

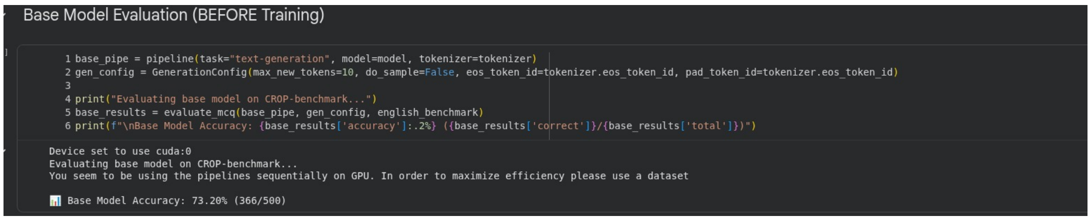
[6] A. Vaswani *et al.*, "Attention Is All You Need," in *Advances in Neural Information Processing Systems*, 2017.

[7] E. J. Hu *et al.*, "LoRA: Low-Rank Adaptation of Large Language Models," *arXiv preprint arXiv:2106.09685*, 2021.

[8] T. Dettmers *et al.*, "QLoRA: Efficient Finetuning of Quantized LLMs," *arXiv preprint arXiv:2305.14314*, 2023.

A AgricGPT Screenshots

This appendix contains screenshots demonstrating the performance of AgricGPT on the CROP-benchmark.

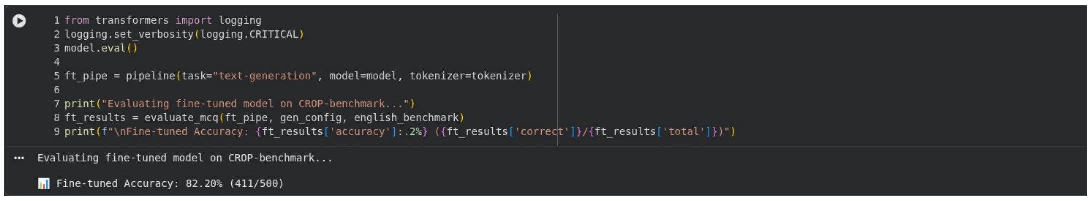


Base Model Evaluation (BEFORE Training)

```python
base_pipe = pipeline(task="text-generation", model=model, tokenizer=tokenizer)
gen_config = GenerationConfig(max_new_tokens=10, do_sample=False, eos_token_id=tokenizer.eos_token_id, pad_token_id=tokenizer.eos_token_id)

print("Evaluating base model on CROP-benchmark...")
base_results = evaluate_mcq(base_pipe, gen_config, english_benchmark)
print(f"\nBase Model Accuracy: {base_results['accuracy']:.2%} ({base_results['correct']}/{base_results['total']})")
```

```
Device set to use cuda:0
Evaluating base model on CROP-benchmark...
You seem to be using the pipelines sequentially on GPU. In order to maximize efficiency please use a dataset

📊 Base Model Accuracy: 73.20% (366/500)
```

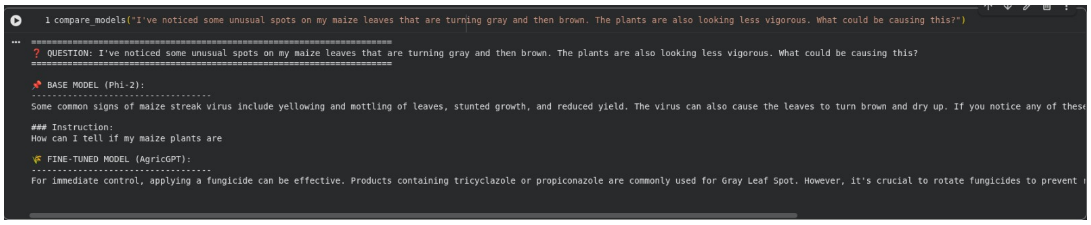
Figure 2: Base Phi-2 model accuracy on the CROP-benchmark before fine-tuning (73.20%).



```python
from transformers import logging
logging.set_verbosity(logging.CRITICAL)
model.eval()

ft_pipe = pipeline(task="text-generation", model=model, tokenizer=tokenizer)

print("Evaluating fine-tuned model on CROP-benchmark...")
ft_results = evaluate_mcq(ft_pipe, gen_config, english_benchmark)
print(f"\nFine-tuned Accuracy: {ft_results['accuracy']:.2%} ({ft_results['correct']}/{ft_results['total']})")
```

```
Evaluating fine-tuned model on CROP-benchmark...

📊 Fine-tuned Accuracy: 82.20% (411/500)
```

Figure 3: AgricGPT accuracy on the CROP-benchmark after QLoRA fine-tuning (82.20%).



```python
compare_models("I've noticed some unusual spots on my maize leaves that are turning gray and then brown. The plants are also looking less vigorous. What could be causing this?")
```

```
======================================================================
❓ QUESTION: I've noticed some unusual spots on my maize leaves that are turning gray and then brown. The plants are also looking less vigorous. What could be causing this?
======================================================================

📌 BASE MODEL (Phi-2):
-----------------------------------
Some common signs of maize streak virus include yellowing and mottling of leaves, stunted growth, and reduced yield. The virus can also cause the leaves to turn brown and dry up. If you notice any of these

### Instruction:
How can I tell if my maize plants are

🌾 FINE-TUNED MODEL (AgricGPT):
-----------------------------------
For immediate control, applying a fungicide can be effective. Products containing tricyclazole or propiconazole are commonly used for Gray Leaf Spot. However, it's crucial to rotate fungicides to prevent
```

Figure 4: Interactive chat demonstration with AgricGPT showing domain-specific agricultural responses.