

# Chapter 7: Large Language Models

Covered Material	Value
Chapters	Chapter 4 (Naive Bayes), Chapter 10 (Large Language Models), Chapter 11 (Masked Language Models), Chapter 12 (Model Alignment, Prompting, and In-Context Learning)
Book	Speech and Language Processing (Jurafsky & Martin)
Related Week	Week 7
Related Slides / Files	<a href="#">week7.md</a> , <a href="#">9.md</a> , <a href="#">10.md</a> , <a href="#">11.md</a> , <a href="#">12.md</a>

## Index

- Index (this section)
- Brief Content Summary
- Abstract
- Keywords
- 1. Introduction
  - 1.1 From Language Models to Large Language Models
  - 1.2 Chapter Overview
- 2. Background
  - 2.1 Key References
  - 2.2 Historical Context
- 3. Text Classification Foundations
  - 3.1 The Naive Bayes Classifier
  - 3.2 Bayesian Inference for Text
  - 3.3 Training Naive Bayes: Maximum Likelihood Estimation
  - 3.4 Smoothing Techniques
  - 3.5 Naive Bayes as a Language Model
- 4. Evaluation Metrics for Classification
  - 4.1 The Confusion Matrix
  - 4.2 Precision, Recall, and F-Measure
  - 4.3 Multi-Class Evaluation: Micro- and Macro-Averaging
  - 4.4 Cross-Validation
  - 4.5 Statistical Significance Testing
- 5. Large Language Models with Transformers
  - 5.1 Conditional Generation and Word Prediction
  - 5.2 Decoding Strategies

- 5.3 Sampling Methods for Generation
- 6. Pretraining Large Language Models
  - 6.1 Self-Supervised Training
  - 6.2 Training Corpora and Data Considerations
  - 6.3 Finetuning Paradigms
- 7. Masked Language Models
  - 7.1 Bidirectional Transformer Encoders
  - 7.2 Masked Language Modeling Objective
  - 7.3 Next Sentence Prediction
  - 7.4 BERT Architecture and Variants
- 8. Contextual Embeddings
  - 8.1 From Static to Contextual Representations
  - 8.2 Word Sense and Polysemy
  - 8.3 Word Sense Disambiguation with Contextual Embeddings
  - 8.4 Anisotropy and Similarity Computation
- 9. Finetuning for Downstream Tasks
  - 9.1 Sequence Classification
  - 9.2 Sequence-Pair Classification and Natural Language Inference
  - 9.3 Sequence Labeling and Named Entity Recognition
- 10. Scaling and Efficiency
  - 10.1 Scaling Laws
  - 10.2 The KV Cache
  - 10.3 Parameter-Efficient Finetuning: LoRA
- 11. Model Alignment and Instruction Tuning
  - 11.1 The Alignment Problem
  - 11.2 Instruction Tuning (SFT)
  - 11.3 Reinforcement Learning from Human Feedback (RLHF)
- 12. Prompting and In-Context Learning
  - 12.1 Prompt Engineering
  - 12.2 Zero-Shot and Few-Shot Prompting
  - 12.3 Chain-of-Thought Prompting
  - 12.4 In-Context Learning Mechanisms
  - 12.5 Automatic Prompt Optimization
- 13. Potential Harms and Ethical Considerations
- 14. Conclusion
- 15. References

---

## Brief Content Summary

---

This chapter provides a comprehensive treatment of Large Language Models (LLMs), integrating foundational text classification with modern transformer-based architectures. The exposition begins with Naive Bayes classification, establishing the probabilistic framework for text understanding through Bayes' theorem, the bag-of-words assumption, and maximum likelihood estimation with Laplace smoothing. Evaluation methodology is formalized through precision, recall,  $F_1$ -measure, and statistical significance testing via the paired bootstrap. The chapter then transitions to transformer-based LLMs, covering both causal (autoregressive) and masked (bidirectional) language models. Key architectures including GPT and BERT are examined, with emphasis on their distinct pretraining objectives and downstream applications. Contextual embeddings are characterized as dynamic representations enabling word sense disambiguation. The chapter culminates with model alignment techniques—instruction tuning and RLHF—alongside prompting paradigms including zero-shot, few-shot, and chain-of-thought approaches for eliciting desired model behaviors.

---

## Abstract

---

Large Language Models (LLMs) represent a paradigm shift in natural language processing, demonstrating that knowledge about language structure and world facts can be acquired through self-supervised pretraining on massive text corpora. This chapter traces the theoretical foundations from classical probabilistic text classification to modern transformer-based architectures. We formalize the Naive Bayes classifier as a generative model employing the conditional independence assumption, deriving maximum likelihood estimators with Laplace smoothing for robust probability estimation. Evaluation frameworks are established through the confusion matrix, precision-recall trade-offs, and the  $F_\beta$ -measure, with statistical validity ensured via bootstrap significance testing. The transition to neural approaches is realized through transformer language models, distinguishing between causal models (GPT family) optimized for autoregressive generation and masked language models (BERT family) designed for bidirectional context encoding. We characterize contextual embeddings as position-dependent vector representations that capture word sense in context, enabling applications from sentiment analysis to named entity recognition. Practical deployment considerations including scaling laws, KV caching, and parameter-efficient finetuning (LoRA) are examined. The chapter concludes with model alignment methodologies—supervised finetuning on instruction datasets and reinforcement learning from human feedback—alongside prompting strategies that enable in-context learning without parameter updates. Potential harms including hallucination, toxicity, and bias are discussed within an ethical framework for responsible LLM development.

---

## Keywords

---

Large Language Models; Transformer; BERT; GPT; Naive Bayes; Text Classification; Pretraining; Finetuning; Contextual Embeddings; Masked Language Modeling; Causal Language Modeling; Instruction Tuning; RLHF; Prompting; In-Context Learning; Chain-of-Thought; Perplexity; Attention Mechanism; Word Sense Disambiguation

---

## 1. Introduction

---

This section establishes the motivation for studying Large Language Models and outlines the chapter's scope.

### 1.1 From Language Models to Large Language Models

The development of Large Language Models represents one of the most significant advances in artificial intelligence. The central insight underlying these models is the **distributional hypothesis**: that aspects of word meaning can be learned from the patterns of co-occurrence in text. Children acquire vocabulary at rates of 7–10 words per day, primarily through reading, suggesting that remarkable amounts of knowledge can be extracted from text alone.

LLMs formalize this intuition through **pretraining**—learning representations of language and world knowledge from vast text corpora—followed by application to downstream tasks. The resulting pretrained language models exhibit remarkable performance across natural language tasks because of the knowledge acquired during pretraining. They have proven especially transformative for generative tasks including summarization, machine translation, question answering, and conversational agents.

The power of LLMs derives from their ability to cast diverse NLP tasks as **word prediction**. Sentiment analysis becomes predicting whether "positive" or "negative" is more likely given a review. Question answering becomes generating the most probable continuation after a question. This unification enables a single model to address multiple tasks through appropriate prompting.

## 1.2 Chapter Overview

This chapter synthesizes four complementary perspectives on language understanding:

1. **Probabilistic Classification** (Section 3): The Naive Bayes framework establishes foundational concepts including Bayesian inference, conditional independence, and smoothing.
2. **Evaluation Methodology** (Section 4): Rigorous metrics including precision, recall,  $F_1$ , and statistical significance testing enable meaningful model comparison.
3. **Transformer Architectures** (Sections 5–7): Both causal (GPT-style) and masked (BERT-style) language models are examined, along with their pretraining objectives and contextual representations.
4. **Deployment and Alignment** (Sections 10–12): Practical considerations including scaling, efficiency, instruction tuning, and prompting complete the treatment.

---

## 2. Background

This section situates Large Language Models within the broader NLP research landscape.

### 2.1 Key References

- Jurafsky, D. & Martin, J.H. (2024). *Speech and Language Processing*, Chapters 4, 10, 11, 12.
- Vaswani, A., et al. (2017). Attention Is All You Need. *NeurIPS*.
- Devlin, J., et al. (2019). BERT: Pre-training of Deep Bidirectional Transformers. *NAACL-HLT*.
- Brown, T., et al. (2020). Language Models are Few-Shot Learners. *NeurIPS*.
- Ouyang, L., et al. (2022). Training Language Models to Follow Instructions with Human Feedback. *NeurIPS*.

### 2.2 Historical Context

The trajectory from n-gram language models to modern LLMs spans several decades. Early statistical language models, developed by Jelinek and colleagues at IBM in the 1970s, used n-gram probabilities for speech recognition. Neural language models emerged with Bengio et al. (2003), who combined discriminative word prediction with learned embeddings. The RNN language model of Mikolov et al. (2010) achieved performance surpassing 5-gram models. The transformer architecture (Vaswani et al., 2017) enabled efficient parallelization, leading to the scaling of models like GPT-2, GPT-3, and BERT. The current era is characterized by models with hundreds of billions of parameters trained on terabytes of text.

---

## 3. Text Classification Foundations

This section develops the probabilistic framework for text classification using Naive Bayes.

### 3.1 The Naive Bayes Classifier

Text classification assigns a document  $d$  to a class  $c$  from a fixed set  $C = \{c_1, c_2, \dots, c_J\}$ . Examples include:

- **Sentiment Analysis:** Classifying reviews as positive, negative, or neutral.
- **Spam Detection:** Distinguishing spam from legitimate email.
- **Language Identification:** Determining the language of a text.
- **Authorship Attribution:** Identifying the author of an anonymous document.

The Naive Bayes classifier is a **generative model** that applies Bayes' theorem to compute the posterior probability of each class given a document:

$$\hat{c} = \arg \max_{c \in C} P(c|d)$$

By Bayes' rule:

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)}$$

Since  $P(d)$  is constant across classes, the classification decision becomes:

$$\hat{c} = \arg \max_{c \in C} P(d|c)P(c) \quad (3.1)$$

Here  $P(c)$  is the **prior probability** of class  $c$ , reflecting how frequent each class is, and  $P(d|c)$  is the **likelihood** of observing document  $d$  given class  $c$ .

### 3.2 Bayesian Inference for Text

To make the likelihood  $P(d|c)$  tractable, Naive Bayes introduces two simplifying assumptions:

**Bag-of-Words Assumption:** The position of words in a document is irrelevant; only word frequencies matter. A document is represented as an unordered collection (bag) of its words.

**Conditional Independence Assumption:** Given the class, the probability of observing a word is independent of the other words in the document:

$$P(d|c) = P(w_1, w_2, \dots, w_n|c) = \prod_{i=1}^n P(w_i|c) \quad (3.2)$$

where  $w_1, w_2, \dots, w_n$  are the words in document  $d$ .

Combining equations (3.1) and (3.2), the Naive Bayes classifier becomes:

$$\hat{c}_{NB} = \arg \max_{c \in C} P(c) \prod_{i \in \text{positions}} P(w_i|c) \quad (3.3)$$

To avoid numerical underflow from multiplying small probabilities, computation proceeds in log-space:

$$\hat{c}_{NB} = \arg \max_{c \in C} \left[ \log P(c) + \sum_{i \in \text{positions}} \log P(w_i|c) \right] \quad (3.4)$$

### 3.3 Training Naive Bayes: Maximum Likelihood Estimation

The parameters  $P(c)$  and  $P(w_i|c)$  are estimated from a labeled training corpus using **maximum likelihood estimation (MLE)**.

**Prior Probability:** The probability of class  $c$  is estimated as the fraction of training documents belonging to  $c$ :

$$\hat{P}(c) = \frac{N_c}{N_{\text{doc}}} \quad (3.5)$$

where  $N_c$  is the number of documents in class  $c$  and  $N_{\text{doc}}$  is the total number of training documents.

**Likelihood:** The probability of word  $w_i$  given class  $c$  is estimated as the relative frequency of  $w_i$  among all word tokens in documents of class  $c$ :

$$\hat{P}(w_i|c) = \frac{\text{count}(w_i, c)}{\sum_{w \in V} \text{count}(w, c)} \quad (3.6)$$

where  $\text{count}(w_i, c)$  is the number of times word  $w_i$  appears in training documents of class  $c$ , and the denominator sums over all words in vocabulary  $V$ .

### 3.4 Smoothing Techniques

A critical issue with MLE is the **zero-frequency problem**: if a word  $w_k$  never appears in training documents of class  $c$ , then  $\hat{P}(w_k|c) = 0$ . This zeros out the entire product in equation (3.3), regardless of other evidence.

**Laplace (Add-One) Smoothing** addresses this by adding a pseudo-count of 1 to every word:

$$\hat{P}(w_i|c) = \frac{\text{count}(w_i, c) + 1}{\sum_{w \in V} (\text{count}(w, c) + 1)} = \frac{\text{count}(w_i, c) + 1}{\sum_{w \in V} \text{count}(w, c) + |V|} \quad (3.7)$$

where  $|V|$  is the vocabulary size. This ensures every word has non-zero probability while normalizing the distribution to sum to 1.

**Add- $\alpha$  Smoothing** generalizes this with a fractional pseudo-count  $\alpha < 1$ :

$$\hat{P}(w_i|c) = \frac{\text{count}(w_i, c) + \alpha}{\sum_{w \in V} \text{count}(w, c) + \alpha|V|} \quad (3.8)$$

### 3.5 Naive Bayes as a Language Model

When Naive Bayes uses all words in the vocabulary as features, it instantiates a class-specific unigram language model. Each class defines a probability distribution over words, and the likelihood  $P(d|c)$  is the probability that the class-specific language model assigns to the document:

$$P(s|c) = \prod_{i \in \text{positions}} P(w_i|c) \quad (3.9)$$

This connection illuminates how Naive Bayes captures class-specific word usage patterns. A document is classified according to which class's "language" it most resembles.

## 4. Evaluation Metrics for Classification

This section formalizes evaluation methodology for text classifiers.

### 4.1 The Confusion Matrix

For binary classification, the **confusion matrix** tabulates predictions against gold-standard labels:

	Gold Positive	Gold Negative
System Positive	True Positive (TP)	False Positive (FP)
System Negative	False Negative (FN)	True Negative (TN)

**Accuracy** measures the fraction of correct predictions:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

However, accuracy is misleading for **imbalanced classes**. A classifier that predicts "not spam" for all emails achieves 99% accuracy if only 1% of emails are spam, yet fails completely at its intended task.

## 4.2 Precision, Recall, and F-Measure

**Precision** measures the fraction of positive predictions that are correct:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (4.1)$$

**Recall** measures the fraction of actual positives that are correctly identified:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (4.2)$$

Precision and recall trade off against each other. The **F-measure** combines them via the harmonic mean:

$$F_{\beta} = \frac{(\beta^2 + 1) \cdot P \cdot R}{\beta^2 P + R} \quad (4.3)$$

The parameter  $\beta$  weights the relative importance of recall versus precision. When  $\beta = 1$ , precision and recall are equally weighted, yielding the **F<sub>1</sub>-measure**:

$$F_1 = \frac{2PR}{P + R} \quad (4.4)$$

The harmonic mean is used because it penalizes extreme values more than the arithmetic mean, emphasizing that both precision and recall must be reasonable.

## 4.3 Multi-Class Evaluation: Micro- and Macro-Averaging

For multi-class problems, precision and recall are computed per class, then aggregated:

**Macro-averaging** computes the metric for each class independently and averages:

$$\text{Macro-Precision} = \frac{1}{J} \sum_{j=1}^J \text{Precision}_j$$

**Micro-averaging** pools predictions across all classes into a single confusion matrix:

$$\text{Micro-Precision} = \frac{\sum_{j=1}^J \text{TP}_j}{\sum_{j=1}^J (\text{TP}_j + \text{FP}_j)}$$

Micro-averaging is dominated by frequent classes; macro-averaging weights all classes equally, better reflecting performance on rare classes.

## 4.4 Cross-Validation

**k-Fold Cross-Validation** addresses the limitation of fixed train/test splits by partitioning data into  $k$  disjoint **folds**. The model is trained on  $k - 1$  folds and evaluated on the held-out fold, rotating through all  $k$  possibilities. Performance is averaged across folds.

**10-fold cross-validation** is standard, training 10 models on 90% of the data each. This provides more robust performance estimates while using all data for both training and testing (across different folds).

## 4.5 Statistical Significance Testing

To determine whether system A genuinely outperforms system B, we use **statistical hypothesis testing**. The **effect size**  $\delta(x)$  measures the performance difference on test set  $x$ :

$$\delta(x) = M(A, x) - M(B, x) \quad (4.5)$$

The **null hypothesis**  $H_0$  posits that  $\delta(x) \leq 0$  (A is not better than B). The **p-value** is the probability of observing a difference as large as  $\delta(x)$  assuming  $H_0$  is true:

$$\text{p-value} = P(\delta(X) \geq \delta(x) | H_0 \text{ is true}) \quad (4.6)$$

A result is **statistically significant** if the p-value falls below a threshold (typically 0.01 or 0.05), allowing rejection of  $H_0$ .

The **Paired Bootstrap Test** creates  $b$  virtual test sets by sampling with replacement from the original test set  $x$ . For each virtual set  $x^{(i)}$ , we compute  $\delta(x^{(i)})$ . The p-value is estimated as:

$$\text{p-value}(x) = \frac{1}{b} \sum_{i=1}^b 1(\delta(x^{(i)}) \geq 2\delta(x)) \quad (4.7)$$

where  $1(\cdot)$  is the indicator function. If fewer than 1% of bootstrap samples show A "accidentally" beating B by  $\delta(x)$  or more, we conclude A is significantly better.

## 5. Large Language Models with Transformers

This section introduces transformer-based Large Language Models and their application to NLP tasks.

### 5.1 Conditional Generation and Word Prediction

Large Language Models built from transformers excel at **conditional generation**: generating text conditioned on an input prompt. The key insight is that many NLP tasks can be cast as **word prediction**:

**Sentiment Analysis** as word prediction:

$$P(\text{positive} \mid \text{"The sentiment of 'I like this movie' is:"}) \quad \text{vs.} \quad P(\text{negative} \mid \text{"The sentiment of 'I like this movie' is:"})$$

**Question Answering** as word prediction:

$$P(w \mid \text{"Q: Who wrote 'The Origin of Species'? A:"})$$

The model generates "Charles" as the most probable next token, then "Darwin" given the extended context.

**Text Summarization** uses special tokens like `<tl;dr>` (too long; didn't read) to signal that a summary should follow the input text. The model generates the summary token-by-token, conditioned on the full input.

The power of transformer LLMs for these tasks derives from their **long context windows** (thousands of tokens), enabling attention over the full input during generation.

### 5.2 Decoding Strategies

Given a context, the language model computes a probability distribution over the vocabulary for the next token. **Decoding** is the process of selecting which token to generate.

**Greedy Decoding** selects the highest-probability token at each step:

$$w_t = \arg \max_{w \in V} P(w | w_{<t}) \quad (5.1)$$

While simple, greedy decoding produces generic, repetitive text because high-probability tokens are, by definition, predictable. The decoding is deterministic: identical contexts always yield identical outputs.

**Beam Search** maintains  $k$  candidate sequences (beams), extending each by considering multiple high-probability continuations. It selects the sequence with highest cumulative log-probability:

$$\text{score}(w_1, \dots, w_t) = \sum_{i=1}^t \log P(w_i | w_{<i})$$

Beam search works well for constrained tasks like machine translation but still tends toward generic outputs for open-ended generation.

## 5.3 Sampling Methods for Generation

**Sampling** methods introduce randomness to generate more diverse, creative text.

**Random Sampling** draws tokens according to their model probabilities:

$$w_i \sim P(w_i | w_{<i})$$

However, the long tail of low-probability words leads to occasional incoherent outputs.

**Top- $k$  Sampling** restricts sampling to the  $k$  most probable tokens:

1. Compute  $P(w_t | w_{<t})$  for all vocabulary words.
2. Retain only the top  $k$  words by probability.
3. Renormalize to form a valid distribution.
4. Sample from this truncated distribution.

When  $k = 1$ , top- $k$  sampling reduces to greedy decoding.

**Top- $p$  (Nucleus) Sampling** dynamically adjusts the candidate set based on cumulative probability rather than fixed count. The **nucleus**  $V^{(p)}$  is the smallest set such that:

$$\sum_{w \in V^{(p)}} P(w | w_{<t}) \geq p \quad (5.2)$$

This adapts to the shape of the distribution: when probability is concentrated on few words, the nucleus is small; when probability is diffuse, more candidates are included.

**Temperature Sampling** reshapes the probability distribution before sampling. The logits  $\mathbf{u}$  are divided by temperature  $\tau$  before softmax:

$$\mathbf{y} = \text{softmax}(\mathbf{u}/\tau) \quad (5.3)$$

- **Low temperature** ( $\tau < 1$ ): Sharpens the distribution, making high-probability tokens more dominant (more deterministic).
- **High temperature** ( $\tau > 1$ ): Flattens the distribution, giving rare tokens more chance (more random).
- As  $\tau \rightarrow 0$ , sampling approaches greedy decoding.

---

## 6. Pretraining Large Language Models

This section details the training methodology for Large Language Models.

### 6.1 Self-Supervised Training

LLMs are trained using **self-supervision**: the training signal comes from the text itself, without human-provided labels. At each position  $t$ , the model predicts the next token  $w_{t+1}$  given the preceding context  $w_1, \dots, w_t$ .

The training objective is to minimize **cross-entropy loss**:

$$L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\log \hat{\mathbf{y}}_t[w_{t+1}] \quad (6.1)$$

where  $\hat{\mathbf{y}}_t$  is the model's probability distribution and  $\mathbf{y}_t$  is the one-hot vector for the true next word.

The loss for a sequence is averaged over all positions:

$$L = \frac{1}{T} \sum_{t=1}^T L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) \quad (6.2)$$

**Teacher Forcing**: During training, the model always receives the correct history  $w_{1:t}$  to predict  $w_{t+1}$ , rather than conditioning on its own (potentially incorrect) predictions.

## 6.2 Training Corpora and Data Considerations

LLMs are pretrained on massive datasets, typically scraped from the web:

- **Common Crawl**: Billions of web pages, filtered and deduplicated.
- **The Pile**: 825 GB of English text from diverse sources (web, books, academic papers, code).
- **Wikipedia**: High-quality encyclopedic text in many languages.

**Quality Filtering** uses classifiers to score documents, favoring high-quality sources (Wikipedia, books) and removing boilerplate, duplicates, and personal information.

**Safety Filtering** applies toxicity classifiers to reduce harmful content, though this remains imperfect—current toxicity classifiers can mistakenly flag non-toxic text from minority dialects.

**Ethical Considerations** include:

- **Copyright**: Much training text is copyrighted; fair use applicability is debated.
- **Data Consent**: Increasing numbers of websites opt out of LLM training via robots.txt.
- **Privacy**: Web data contains personal information despite filtering attempts.

## 6.3 Finetuning Paradigms

**Finetuning** continues training a pretrained model on new data. Several paradigms exist:

1. **Continued Pretraining**: Retrain all parameters on domain-specific data using the same objective.
2. **Parameter-Efficient Finetuning (PEFT)**: Freeze most parameters and train only a subset (e.g., LoRA adapters).
3. **Task-Specific Finetuning**: Add a classification head and train on labeled data for a specific task (common with masked LMs like BERT).
4. **Supervised Finetuning (SFT)**: Train on instruction-response pairs to improve instruction-following (discussed in Section 11).

---

# 7. Masked Language Models

This section introduces bidirectional transformer encoders and the masked language modeling objective.

## 7.1 Bidirectional Transformer Encoders

Causal (left-to-right) transformers can only attend to preceding tokens, limiting their ability to use future context. **Bidirectional encoders** remove this restriction, allowing attention over the entire input sequence.

The implementation is straightforward: remove the causal mask from attention computation. Recall that causal attention masks the  $\mathbf{QK}^T$  matrix to prevent attending to future tokens:

$$\mathbf{A} = \text{softmax} \left( \text{mask} \left( \frac{\mathbf{QK}^T}{\sqrt{d_k}} \right) \right) \mathbf{V} \quad (7.1)$$

For bidirectional attention, the mask is simply removed:

$$\mathbf{A} = \text{softmax} \left( \frac{\mathbf{QK}^T}{\sqrt{d_k}} \right) \mathbf{V} \quad (7.2)$$

This enables each token to attend to all tokens in the sequence, capturing both left and right context.

## 7.2 Masked Language Modeling Objective

With bidirectional attention, predicting the next word becomes trivial (the model can simply look ahead). Instead, **Masked Language Modeling (MLM)** trains the model to predict masked tokens within the sequence—a **cloze task**.

In BERT:

- 15% of input tokens are selected for prediction.
- Of these: 80% are replaced with `[MASK]`, 10% with random tokens, 10% unchanged.

The model must predict the original tokens for all masked positions:

$$L_{MLM} = \frac{1}{|M|} \sum_{i \in M} -\log P(x_i | \mathbf{h}^i) \quad (7.3)$$

where  $M$  is the set of masked positions and  $\mathbf{h}^i$  is the output representation at position  $i$ .

The language modeling head computes probabilities via:

$$\mathbf{u}_i = \mathbf{h}_i^L \mathbf{E}^T \quad (7.4)$$

$$\mathbf{y}_i = \text{softmax}(\mathbf{u}_i) \quad (7.5)$$

where  $\mathbf{E}$  is the embedding matrix and  $\mathbf{h}_i^L$  is the final-layer representation.

## 7.3 Next Sentence Prediction

Some applications require understanding relationships between sentence pairs (paraphrase detection, entailment). **Next Sentence Prediction (NSP)** addresses this by training on pairs of sentences:

- 50% are actual consecutive sentences (positive pairs).
- 50% have the second sentence randomly sampled (negative pairs).

Special tokens structure the input:

- `[CLS]` prepended to the input.
- `[SEP]` between and after sentences.

The output vector for `[CLS]` is passed through an NSP head:

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{h}_{CLS}^L \mathbf{W}_{NSP})$$

to predict whether the pair is consecutive. Later models like RoBERTa dropped NSP, finding it unnecessary.

## 7.4 BERT Architecture and Variants

**BERT-base** (Devlin et al., 2019):

- 30,000 WordPiece vocabulary
- 12 transformer layers
- Hidden size  $d = 768$
- 12 attention heads
- ~110M parameters

**RoBERTa** (Liu et al., 2019):

- Removes NSP objective
- Trains longer on more data
- Dynamic masking (different masks each epoch)

**XLNet-RoBERTa** (multilingual):

- 250,000 SentencePiece vocabulary
- 24 transformer layers
- Hidden size 1024
- Trained on 100 languages
- ~550M parameters

Masked language models are typically much smaller than causal LLMs (hundreds of millions vs. hundreds of billions of parameters) because they are used for encoding rather than generation.

---

## 8. Contextual Embeddings

This section characterizes contextual embeddings as dynamic, position-dependent word representations.

### 8.1 From Static to Contextual Representations

Static embedding methods (word2vec, GloVe) assign a single vector to each word type, regardless of context. The word "bank" receives the same representation whether referring to a financial institution or a river bank.

**Contextual embeddings** assign different vectors to each word **instance** based on its sentential context. Given input tokens  $x_1, \dots, x_n$ , the output vector  $\mathbf{h}_i^T$  from the final transformer layer represents the meaning of token  $x_i$  in context.

This enables the representation to capture:

- **Word sense:** Different senses of polysemous words receive different embeddings.
- **Semantic role:** The same word as subject vs. object may differ.
- **Discourse context:** Representations reflect broader textual context.

### 8.2 Word Sense and Polysemy

Words are **polysemous** (Greek: "many signs")—the same word form can express multiple distinct meanings. A **word sense** is a discrete representation of one aspect of meaning:

- **bank**<sup>1</sup>: financial institution

- **bank**<sup>2</sup>: sloping mound (river bank)
- **mouse**<sup>1</sup>: small rodent
- **mouse**<sup>2</sup>: computer input device

Context disambiguates senses:

- "A **mouse** controlling a computer system" → **mouse**<sup>2</sup>
- "A quiet animal like a **mouse**" → **mouse**<sup>1</sup>

Visualization of BERT embeddings for the word "die" across many contexts reveals distinct clusters corresponding to:

- English verb meaning "to cease living"
- English noun meaning "singular of dice"
- German definite article "the" (feminine)

Contextual embeddings thus provide a continuous, high-dimensional model of meaning that naturally captures sense distinctions without explicit sense inventories.

### 8.3 Word Sense Disambiguation with Contextual Embeddings

**Word Sense Disambiguation (WSD)** assigns the correct sense from a fixed inventory (e.g., WordNet) to each word in context. Contextual embeddings enable a simple, effective approach.

**Training:** For each sense  $s$  in a labeled corpus, compute the **contextual sense embedding** by averaging the contextual representations of all instances of that sense:

$$\mathbf{v}_s = \frac{1}{n} \sum_{i=1}^n \mathbf{v}_i \quad \forall \mathbf{v}_i \in \text{tokens}(s) \quad (8.1)$$

**Inference:** Given a target word  $t$  in context, compute its contextual embedding  $\mathbf{t}$  and select the sense with highest cosine similarity:

$$\text{sense}(t) = \arg \max_{s \in \text{senses}(t)} \text{cosine}(\mathbf{t}, \mathbf{v}_s) \quad (8.2)$$

This 1-nearest-neighbor approach achieves state-of-the-art WSD performance, demonstrating that contextual embeddings effectively encode sense information.

### 8.4 Anisotropy and Similarity Computation

A challenge with contextual embeddings is **anisotropy**: vectors for all words tend to point in similar directions, resulting in very high cosine similarities even for unrelated words. An **isotropic** embedding space would have vectors uniformly distributed, with expected cosine similarity near zero for random pairs.

**Cause:** A few **rogue dimensions** have very large magnitudes and high variance, dominating the cosine computation.

**Solution: Standardization** (z-scoring) normalizes each dimension:

$$\boldsymbol{\mu} = \frac{1}{|C|} \sum_{\mathbf{x} \in C} \mathbf{x} \quad (8.3)$$

$$\sigma = \sqrt{\frac{1}{|C|} \sum_{\mathbf{x} \in C} (\mathbf{x} - \boldsymbol{\mu})^2} \quad (8.4)$$

$$\mathbf{z} = \frac{\mathbf{x} - \boldsymbol{\mu}}{\sigma} \quad (8.5)$$

where  $C$  is a corpus of embeddings. This transformation reduces the influence of rogue dimensions, yielding more meaningful similarity scores.

## 9. Finetuning for Downstream Tasks

This section details how pretrained masked language models are adapted to specific NLP applications.

### 9.1 Sequence Classification

**Sequence classification** assigns a single label to an entire text (sentiment analysis, topic classification, spam detection).

The pretrained `[CLS]` token representation  $\mathbf{h}_{CLS}^L$  serves as the sequence-level representation. A **classifier head** with learned weights  $\mathbf{W}_C \in \mathbb{R}^{d \times k}$  (for  $k$  classes) produces predictions:

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{h}_{CLS}^L \mathbf{W}_C) \quad (9.1)$$

Training uses cross-entropy loss on labeled data. The classifier weights  $\mathbf{W}_C$  are learned, and the pretrained model parameters may be frozen or lightly updated.

### 9.2 Sequence-Pair Classification and Natural Language Inference

Some tasks classify pairs of sentences:

- **Paraphrase detection**: Are two sentences paraphrases?
- **Natural Language Inference (NLI)**: Does sentence A entail, contradict, or neither with sentence B?

Input format: `[CLS] sentence_A [SEP] sentence_B [SEP]`

The `[CLS]` representation captures the relationship between sentences. For NLI with three classes (entails, contradicts, neutral):

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{h}_{CLS}^L \mathbf{W}_{NLI}) \quad (9.2)$$

where  $\mathbf{W}_{NLI} \in \mathbb{R}^{d \times 3}$ .

**Example** (MultiNLI dataset):

- **Entails**: Premise: "I'm confused." → Hypothesis: "Not all of it is very clear to me."
- **Contradicts**: Premise: "Tourist offices can be very helpful." → Hypothesis: "Tourist offices are never of any help."

### 9.3 Sequence Labeling and Named Entity Recognition

**Sequence labeling** assigns a label to each token. **Named Entity Recognition (NER)** identifies spans denoting persons, locations, organizations, etc.

**BIO Tagging** encodes span boundaries:

- **B-X**: Beginning of entity type X
- **I-X**: Inside entity type X
- **O**: Outside any entity

**Example**: "Jane Villanueva of United Airlines said..."

Token	Tag
Jane	B-PER
Villanueva	I-PER

Token	Tag
of	O
United	B-ORG
Airlines	I-ORG
said	O

For each token position  $i$ , the output representation  $\mathbf{h}_i^L$  is passed through a classifier:

$$\mathbf{y}_i = \text{softmax}(\mathbf{h}_i^L \mathbf{W}_K) \quad (9.3)$$

$$t_i = \arg \max_k (\mathbf{y}_i) \quad (9.4)$$

where  $\mathbf{W}_K \in \mathbb{R}^{d \times (2n+1)}$  for  $n$  entity types (the  $2n + 1$  accounts for B and I tags per type plus O).

**Subword Alignment:** Since models use subword tokenization, words may be split. During training, each subword receives the word's gold tag. During inference, the first subword's prediction is typically used for the full word.

**Evaluation** uses entity-level precision, recall, and  $F_1$ , counting an entity correct only if both boundaries and type match.

## 10. Scaling and Efficiency

This section addresses practical considerations for training and deploying Large Language Models.

### 10.1 Scaling Laws

LLM performance follows **scaling laws**: loss decreases as power-law functions of model size, dataset size, and compute:

$$L(N) \propto \left( \frac{N_c}{N} \right)^{\alpha_N} \quad (10.1)$$

$$L(D) \propto \left( \frac{D_c}{D} \right)^{\alpha_D} \quad (10.2)$$

$$L(C) \propto \left( \frac{C_c}{C} \right)^{\alpha_C} \quad (10.3)$$

where  $N$  is number of parameters,  $D$  is dataset size (tokens), and  $C$  is compute budget.

The number of parameters in a transformer scales as:

$$N \approx 12 n_{\text{layer}} d^2 \quad (10.4)$$

where  $n_{\text{layer}}$  is the number of layers and  $d$  is the model dimension. GPT-3, with 96 layers and  $d = 12288$ , has approximately 175 billion parameters.

Scaling laws guide resource allocation: early training curves can predict final performance, enabling efficient hyperparameter search.

### 10.2 The KV Cache

During training, attention is computed efficiently in parallel via matrix multiplication. At inference, tokens are generated one at a time, but recomputing key and value vectors for all previous tokens at each step is wasteful.

The **KV cache** stores previously computed key (**K**) and value (**V**) vectors. When generating token  $i$ :

1. Compute query, key, value for the new token.
2. Retrieve keys and values for tokens  $1, \dots, i - 1$  from the cache.

3. Compute attention using the new query against all cached keys/values.
4. Store the new key/value pair in the cache.

This trades memory for computation, dramatically accelerating autoregressive generation.

### 10.3 Parameter-Efficient Finetuning: LoRA

Finetuning all parameters of a 100B+ model is prohibitively expensive. **Low-Rank Adaptation (LoRA)** provides an efficient alternative.

**Intuition:** Weight updates during finetuning can be approximated by low-rank matrices.

**Method:** For a weight matrix  $\mathbf{W} \in \mathbb{R}^{N \times d}$ , instead of learning update  $\Delta \mathbf{W}$ , learn two low-rank matrices  $\mathbf{A} \in \mathbb{R}^{N \times r}$  and  $\mathbf{B} \in \mathbb{R}^{r \times d}$  where  $r \ll \min(N, d)$ :

$$\mathbf{h} = \mathbf{x}\mathbf{W} + \mathbf{x}\mathbf{A}\mathbf{B} \quad (10.5)$$

The original  $\mathbf{W}$  is frozen; only  $\mathbf{A}$  and  $\mathbf{B}$  are trained.

**Advantages:**

- Dramatically reduced memory and compute (training only  $r \times (N + d)$  parameters instead of  $N \times d$ ).
- LoRA modules can be swapped in/out for different tasks.
- No inference slowdown:  $\mathbf{A}\mathbf{B}$  can be precomputed and added to  $\mathbf{W}$ .

LoRA is typically applied to attention weights ( $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V, \mathbf{W}^O$ ), with  $r$  values of 4–64 common in practice.

---

## 11. Model Alignment and Instruction Tuning

This section addresses the challenge of making LLMs helpful and safe through alignment techniques.

### 11.1 The Alignment Problem

Pretrained LLMs are trained to predict the next word—an objective misaligned with human goals of helpfulness and safety. This manifests in two failure modes:

**Insufficient Helpfulness:** Models may ignore instructions, producing continuations rather than responses:

- **Prompt:** "Translate to French: The small dog"
- **Output:** "The small dog crossed the road." (continuation instead of translation)

**Potential Harmfulness:** Models may generate toxic, false, or dangerous content:

- **Hallucination:** Confidently stating false information.
- **Toxicity:** Generating hate speech, abuse, or harmful advice.
- **Bias:** Perpetuating stereotypes about demographic groups.

**Model alignment** addresses these issues through techniques that adjust LLMs to align with human values of helpfulness and non-harm.

### 11.2 Instruction Tuning (SFT)

**Instruction tuning** (or **supervised finetuning**, SFT) trains models to follow instructions by finetuning on a corpus of (instruction, response) pairs.

The training objective remains next-word prediction, but the data consists of instruction-response examples:

$$L_{SFT} = - \sum_t \log P(w_t | w_{<t}, \text{instruction})$$

#### Instruction Dataset Creation:

1. **Human-written:** Annotators write instruction-response pairs directly.
2. **Repurposed NLP datasets:** Existing labeled datasets (QA, translation, summarization) are converted to instruction format via templates:
  - Template: "Translate the following to French: {input}"
  - Response: "{French translation}"
3. **Crowdworker guidelines:** Detailed annotation instructions become prompts for generating training examples.
4. **Model-generated:** LLMs generate instruction-response pairs, which are human-validated.

Large instruction datasets include:

- **FLAN:** 15 million examples from 1836 tasks
- **Super-Natural Instructions:** 12 million examples from 1600 tasks
- **Aya:** 503 million instructions in 114 languages

**Meta-learning Effect:** Instruction tuning improves performance not just on trained tasks but on novel tasks—the model learns to follow instructions generally.

## 11.3 Reinforcement Learning from Human Feedback (RLHF)

RLHF aligns models with human preferences through a three-stage process:

### Step 1: Supervised Finetuning (SFT)

Train on instruction-response pairs as described above, producing an initial instruction-following model.

### Step 2: Reward Model Training

Human annotators rank model outputs for the same prompt. A **reward model**  $R$  is trained to predict these rankings:

- Given prompt and two responses  $(y_1, y_2)$ , if humans prefer  $y_1$ :

$$L_{RM} = -\log \sigma(R(x, y_1) - R(x, y_2)) \quad (11.1)$$

The reward model learns to assign higher scores to preferred outputs.

### Step 3: Policy Optimization

The LLM is finetuned to maximize reward model scores while staying close to the SFT model (to prevent reward hacking):

$$L_{RL} = -\mathbb{E}_{y \sim \pi_\theta} [R(x, y)] + \beta \cdot \text{KL}(\pi_\theta || \pi_{SFT}) \quad (11.2)$$

**Proximal Policy Optimization (PPO)** is commonly used to optimize this objective.

The KL penalty prevents the model from generating outputs that game the reward model without being genuinely helpful.

**Direct Preference Optimization (DPO)** is an alternative that directly optimizes preferences without training a separate reward model, simplifying the pipeline.

---

## 12. Prompting and In-Context Learning

---

This section covers techniques for eliciting desired behaviors from LLMs through prompting.

## 12.1 Prompt Engineering

A **prompt** is text provided to an LLM to guide its generation. **Prompt engineering** is the process of designing effective prompts.

Effective prompts are:

- **Unambiguous:** Any reasonable continuation accomplishes the task.
- **Constrained:** The format limits possible outputs.
- **Specific:** Roles, output formats, and constraints are explicit.

**Example:** A sentiment analysis prompt:

```
Human: Do you think that "{input}" has negative or positive sentiment?  
Choices:  
(P) Positive  
(N) Negative  
Assistant: I believe the best answer is: (
```

The trailing open parenthesis strongly constrains output to (P) or (N).

## 12.2 Zero-Shot and Few-Shot Prompting

**Zero-shot prompting** provides only instructions, no examples:

```
Translate to French: The cat sat on the mat.
```

**Few-shot prompting** includes **demonstrations**—labeled examples that clarify the task:

```
Translate to French:  
English: The dog ran. French: Le chien a couru.  
English: She is happy. French: Elle est heureuse.  
English: The cat sat on the mat. French:
```

Few-shot prompting often improves performance, though surprisingly:

- Primary benefit is demonstrating format, not providing task knowledge.
- Even incorrect labels can help (by showing the format).
- Too many examples may cause overfitting to those specific examples.

**Demonstration Selection:** Similar demonstrations (by embedding similarity to the current input) tend to work best. Systems like DSPy can automatically optimize demonstration selection.

## 12.3 Chain-of-Thought Prompting

**Chain-of-thought (CoT) prompting** improves reasoning by including intermediate steps in demonstrations:

**Standard prompting:**

```
Q: Roger has 5 tennis balls. He buys 2 more cans of 3. How many does he have?  
A: 11
```

**Chain-of-thought prompting:**

Q: Roger has 5 tennis balls. He buys 2 more cans of 3. How many does he have?  
A: Roger started with 5 balls. 2 cans of 3 balls each is 6 balls.  $5 + 6 = 11$ .  
The answer is 11.

When demonstrations include reasoning steps, the model is more likely to generate similar reasoning for new problems, leading to correct answers on tasks where direct prediction fails.

CoT is particularly effective for:

- Mathematical word problems
- Multi-step reasoning
- Temporal sequencing
- Logical inference

## 12.4 In-Context Learning Mechanisms

In-context learning (ICL) refers to learning that occurs during prompt processing, without gradient updates. Two forms exist:

1. **Demonstration-based:** Few-shot examples teach new tasks.
2. **Context accumulation:** Predictions improve as context grows.

**Induction Heads** are a proposed mechanism for ICL. These attention circuits implement pattern completion:

$$AB \dots A \rightarrow B$$

When processing token A, the induction head:

1. **Prefix matching:** Searches context for prior instance of A.
2. **Copying:** Increases probability of the token B that followed the earlier A.

A generalized fuzzy version ( $A^*B^* \dots A \rightarrow B^*$  where  $A^* \approx A$ ) may underlie in-context learning: the model learns patterns from demonstrations and applies similar patterns to new inputs.

**Evidence:** Ablating induction heads (zeroing their output) significantly degrades in-context learning performance.

## 12.5 Automatic Prompt Optimization

Manual prompt engineering is tedious. **Automatic prompt optimization** searches for better prompts algorithmically.

**Components:**

1. **Start state:** Initial human or machine-generated prompt.
2. **Scoring metric:** Task performance (accuracy, ROUGE, BLEU).
3. **Expansion method:** Generate prompt variations.

**Beam Search Optimization:**

1. Maintain top- $k$  prompts by score.
2. Expand each by generating variants (paraphrases, modifications).
3. Evaluate variants on a sample of training data.
4. Keep the best-scoring prompts.
5. Repeat until convergence.

**Critique-Based Optimization:**

1. Run current prompt on training examples.
2. Identify examples where it fails.
3. Use an LLM to critique the prompt based on failures.
4. Generate improved prompts addressing the critique.

This gradient-like approach directs search toward improvements rather than random exploration.

---

## 13. Potential Harms and Ethical Considerations

---

Large Language Models pose significant risks that must be addressed through careful design and deployment.

**Hallucination:** LLMs generate plausible-sounding but false information. Training objectives favor fluent, coherent text, not factual accuracy. This is especially dangerous for medical, legal, or safety-critical applications. **Retrieval-augmented generation** is one mitigation approach.

**Toxic Language:** Models trained on internet text reproduce and sometimes amplify toxic content—hate speech, abuse, stereotypes. Even non-toxic prompts can elicit harmful outputs. Safety filtering of training data helps but is imperfect.

**Bias and Stereotypes:** Models encode and perpetuate societal biases present in training data. Sentiment classifiers assign more negative sentiment to African American names. Toxicity classifiers disproportionately flag text in African American Vernacular English.

**Privacy Leakage:** LLMs can memorize and reproduce training data, including personal information (names, addresses, phone numbers). Adversarial attacks can extract this data.

**Misinformation and Misuse:** LLMs can generate convincing misinformation, phishing content, or extremist propaganda at scale.

**Mitigation Strategies:**

- **Model cards** documenting training data, known biases, and intended use.
- **Red teaming** to identify failure modes before deployment.
- **Safety training** via instruction tuning and RLHF.
- **Content filtering** on inputs and outputs.
- **Transparency** about model capabilities and limitations.

---

## 14. Conclusion

---

This chapter has traced the development of language understanding from classical probabilistic classification to modern Large Language Models.

**Key Theoretical Contributions:**

1. **Probabilistic Framework:** Naive Bayes establishes foundational concepts of Bayesian inference, conditional independence, and smoothing for text classification.
2. **Evaluation Rigor:** Precision, recall,  $F_1$ , and statistical significance testing enable meaningful model comparison.
3. **Transformer Architectures:** Both causal (GPT) and masked (BERT) language models demonstrate that self-supervised pretraining on massive corpora produces powerful representations.
4. **Contextual Embeddings:** Dynamic, position-dependent word representations capture sense, role, and discourse context.
5. **Model Alignment:** Instruction tuning and RLHF address the mismatch between pretraining objectives and human values.

6. **In-Context Learning:** Prompting enables task adaptation without parameter updates, with chain-of-thought improving complex reasoning.

**Practical Implications:**

- A single LLM can address diverse tasks through appropriate prompting.
- Scaling laws guide resource allocation for model development.
- Parameter-efficient methods (LoRA) enable adaptation with limited resources.
- Alignment techniques are essential for safe, helpful deployment.

**Open Challenges:**

- Reliable factual grounding to prevent hallucination.
- Fair, unbiased performance across demographic groups.
- Interpretability of model reasoning.
- Sustainable compute requirements.

The field continues to evolve rapidly, with each advance revealing new capabilities and new challenges for responsible development.

---

## 15. References

- Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, 3, 1137–1155.
- Brown, T., et al. (2020). Language models are few-shot learners. *NeurIPS*, 33.
- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *NAACL-HLT*.
- Gehman, S., Gururangan, S., Sap, M., Choi, Y., & Smith, N.A. (2020). RealToxicityPrompts: Evaluating neural toxic degeneration in language models. *Findings of EMNLP*.
- Holtzman, A., Buys, J., Du, L., Forbes, M., & Choi, Y. (2020). The curious case of neural text degeneration. *ICLR*.
- Hu, E.J., et al. (2022). LoRA: Low-rank adaptation of large language models. *ICLR*.
- Kaplan, J., et al. (2020). Scaling laws for neural language models. *arXiv preprint*.
- Liu, Y., et al. (2019). RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint*.
- Mikolov, T., et al. (2010). Recurrent neural network based language model. *INTERSPEECH*.
- Olsson, C., et al. (2022). In-context learning and induction heads. *arXiv preprint*.
- Ouyang, L., et al. (2022). Training language models to follow instructions with human feedback. *NeurIPS*, 35.
- Raffel, C., et al. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 21(140), 1–67.
- Vaswani, A., et al. (2017). Attention is all you need. *NeurIPS*.
- Wei, J., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. *NeurIPS*, 35.

---

## 16. Glossary

Term	Definition
Attention	Mechanism that computes weighted combinations of input representations based on learned relevance scores.
Bag-of-Words	Document representation that ignores word order, treating text as an unordered collection of words.
BERT	Bidirectional Encoder Representations from Transformers; masked language model for contextualized embeddings.
Causal Language Model	Language model that predicts tokens left-to-right, attending only to previous context.
Chain-of-Thought	Prompting technique that includes reasoning steps in demonstrations to improve multi-step reasoning.
Conditional Independence	Assumption that features are independent given the class label.
Contextual Embedding	Vector representation of a word instance that depends on its surrounding context.
Cross-Entropy Loss	Loss function measuring difference between predicted and true probability distributions.
F-measure	Harmonic mean of precision and recall; $F_1$ weights them equally.
Few-Shot Prompting	Providing labeled examples in the prompt to demonstrate the task.
Finetuning	Continuing to train a pretrained model on task-specific or domain-specific data.
GPT	Generative Pre-trained Transformer; causal language model for text generation.
Hallucination	Generation of plausible-sounding but factually incorrect content.
In-Context Learning	Learning that occurs during prompt processing without gradient updates.
Instruction Tuning	Finetuning on instruction-response pairs to improve instruction following.
KV Cache	Stored key and value vectors from previous tokens to accelerate autoregressive generation.
Laplace Smoothing	Adding pseudo-counts to avoid zero probabilities for unseen events.
LoRA	Low-Rank Adaptation; parameter-efficient finetuning via low-rank weight updates.
Masked Language Model	Model trained to predict masked tokens from bidirectional context.
Naive Bayes	Generative classifier applying Bayes' rule with conditional independence assumption.
Perplexity	Inverse probability normalized by sequence length; lower is better.
Precision	Fraction of positive predictions that are correct.
Pretraining	Initial training on large unlabeled corpora to learn general representations.
Prompt	Text input that guides LLM generation toward a desired output.
Recall	Fraction of actual positives correctly identified.
RLHF	Reinforcement Learning from Human Feedback; alignment technique using learned reward models.
Scaling Laws	Power-law relationships between model/data size and performance.
Self-Supervision	Training where labels come from the data itself (e.g., next-word prediction).
Teacher Forcing	Training technique where the model receives correct history rather than its own predictions.
Tokenization	Splitting text into subword units for model processing.
Transformer	Neural architecture based on self-attention, enabling parallel processing of sequences.
Word Sense	Discrete representation of one meaning of a polysemous word.
Zero-Shot Prompting	Prompting with instructions only, no labeled examples.