Predicting the Next Word

From Shannon to ChatGPT

Test Compilation

Contents

Chapter 1

Introduction: The Problem of Prediction

In this chapter, we advance next-word prediction by:

- Defining the fundamental question: What word comes next?
- Connecting language modeling to information theory
- Establishing perplexity as our evaluation metric
- Understanding why prediction difficulty varies with context

1.1 What is Next-Word Prediction?

Consider the sentence fragment:

"The cat sat on the _____"

What word comes next? Most English speakers would guess "mat," "floor," "couch," or similar nouns describing surfaces where a cat might reasonably sit. This simple task—predicting the next word given preceding context—lies at the heart of language modeling and, more broadly, modern natural language processing. The question appears deceptively simple, yet answering it well requires capturing the full complexity of human language: syntactic rules that determine which word categories can follow "the," semantic knowledge that cats sit on physical surfaces rather than abstract concepts, and world knowledge that mats are common resting places for household pets. Every time a language model produces a response, translates a sentence, completes a search query, or generates code, it is fundamentally answering this question of what word or token should come next. The applications of accurate next-word prediction extend far beyond autocomplete: machine translation systems generate target language text one word at a time, speech recognition systems score hypothesized transcriptions by their language model probability, and modern AI assistants synthesize coherent multi-paragraph responses through iterated next-word prediction. Understanding this core task is thus essential for understanding the entire field of natural language processing.

A **language model** assigns probabilities to sequences of words, quantifying how likely each possible continuation is given the words that have come before. Given some context, the model tells us which continuations are likely and which are not, effectively encoding our expectations about language in a mathematical form that computers can manipulate. The "cat sat on the" context strongly favors surfaces over actions, and common surfaces over rare ones, because that is the pattern we observe in natural English text—the statistical regularities of language become encoded in the model's parameters. This probability assignment must satisfy the axioms of probability theory: all probabilities must be non-negative, and they must sum to exactly one over all possible next words, ensuring we have a valid distribution. The quality of a language model is determined by how well its probability assignments match the actual patterns of language: a good model assigns high probability to sequences that native speakers would find natural and low probability to sequences that are awkward

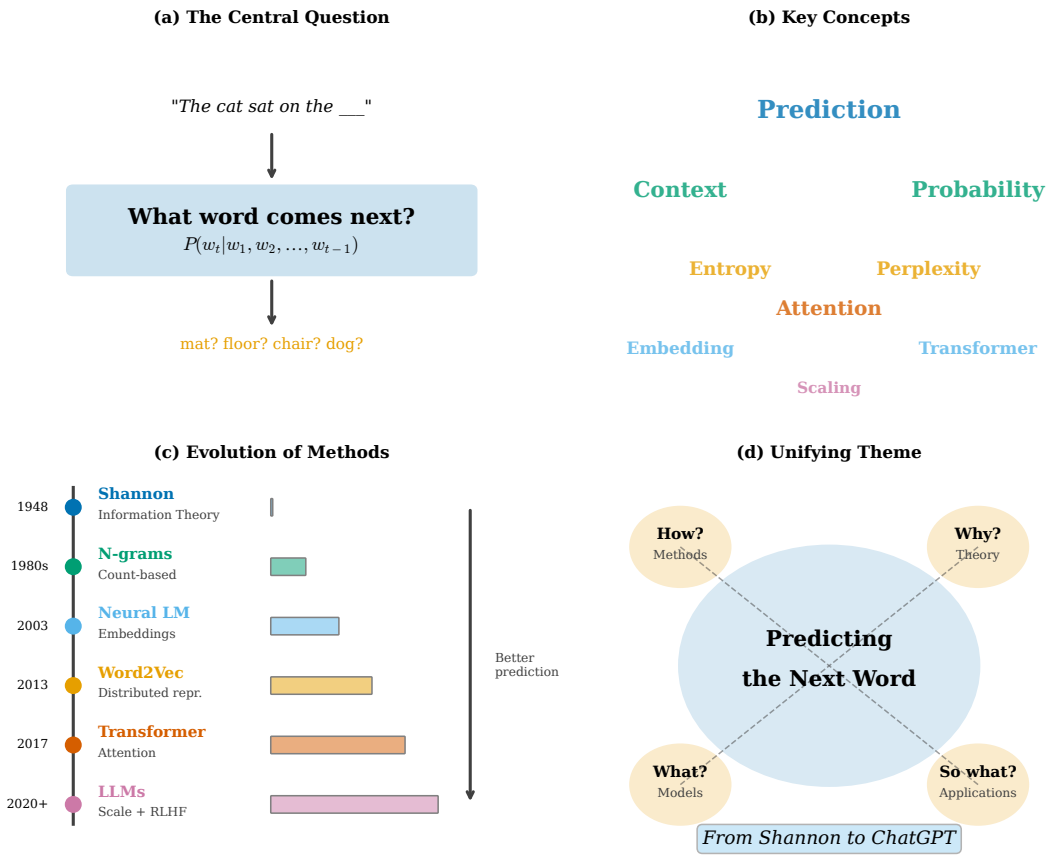

Figure 1.1: The central question of this book: predicting the next word. Panel (a) shows the core prediction problem with candidate words. Panel (b) displays key concepts that form the foundation of language modeling. Panel (c) traces the evolution from Shannon's information theory (1948) through n-grams, neural language models, Word2Vec, Transformers, to modern LLMs. Panel (d) presents the unifying theme connecting theory, methods, models, and applications.

or ungrammatical. Training a language model means adjusting its parameters so that its probability estimates become increasingly accurate on observed text.

Formally, we seek to model the probability distribution over the vocabulary $\mathcal{V}$ given a sequence of preceding words (the *context*). This mathematical formulation captures the essence of the language modeling problem: given everything that has been written or spoken so far, what word is most likely to come next? The vocabulary $\mathcal{V}$ represents the set of all possible words or tokens that the model can predict, typically ranging from tens of thousands of words for word-level models to hundreds of thousands of subword units for modern tokenizers. The context comprises all preceding words in the sequence, though as we shall see, different architectures handle context in fundamentally different ways—some considering only a fixed window of recent words, others attempting to capture the entire preceding history. The conditional probability distribution must satisfy the axioms of probability: every probability must be non-negative, and the probabilities over all possible next words must sum to exactly one, ensuring we have a valid distribution from which we could, in principle, sample the next word. This constraint shapes the design of neural language models, which typically use a softmax function to transform unconstrained outputs into valid probability distributions:

$$P(w_t \,|\, w_1, w_2, \ldots, w_{t-1}) \tag{1.1}$$

where $w_t \in \mathcal{V}$ is the target word and $w_1, \ldots, w_{t-1}$ is the context.

This probability distribution captures the structure of language at multiple levels, each contributing different constraints on what words may follow. At the syntactic level, grammar imposes strict ordering requirements: after the article "the," English syntax demands nouns, adjectives, or other noun phrase constituents, effectively

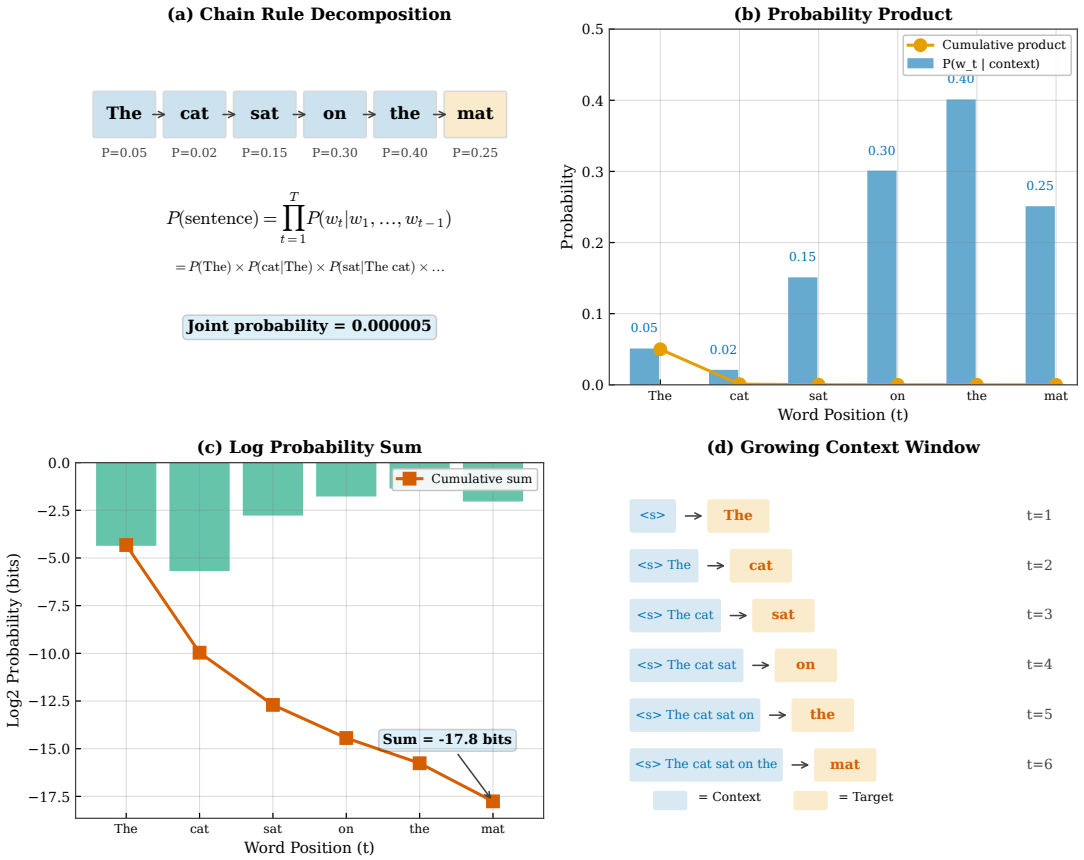

Figure 1.2: The chain rule decomposition of joint probability. Panel (a) shows how a sentence probability factors into conditional probabilities. Panel (b) illustrates the growing context at each position. Panel (c) demonstrates the multiplication of probabilities. Panel (d) shows numerically how small probabilities compound.

ruling out verbs, prepositions used alone, or conjunctions. These syntactic constraints arise from the hierarchical structure of language and dramatically reduce the space of plausible continuations at each position. Beyond syntax, semantic coherence further narrows predictions based on meaning. Given the phrase "cat sat on," we expect words denoting surfaces or locations—"mat," "floor," "windowsill"—rather than actions or abstract concepts, because the semantics of "sitting on" requires a physical support surface. World knowledge provides even stronger constraints in certain contexts: the phrase "The capital of France is" strongly predicts "Paris" not because of syntactic or semantic rules, but because of factual knowledge about geography that any competent English speaker possesses. Finally, discourse context tracks entities and relationships across sentences, so pronouns like "it" or "she" must resolve to previously mentioned referents, constraining predictions based on what the text has already established.

Definition 1.1 (Language Model)**.** *A* language model *is a probability distribution* $P(w_1, w_2, \ldots, w_T)$ *over sequences of words from a vocabulary* $\mathcal{V}$*. By the chain rule of probability:*

$$P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t \,|\, w_1, \ldots, w_{t-1}) \tag{1.2}$$

This decomposition is exact and universal—it applies to any probability distribution over sequences. The challenge lies in estimating each conditional probability $P(w_t \,|\, w_1, \ldots, w_{t-1})$ from finite data. Figure 1.2 illustrates this decomposition visually.

1.1.1 The Chain Rule in Practice

The chain rule tells us that to compute $P(\text{“The cat sat on the mat”})$, we multiply:

$$\begin{aligned} &P(\text{The}) \times P(\text{cat} \mid \text{The}) \times P(\text{sat} \mid \text{The cat}) \\ &\times P(\text{on} \mid \text{The cat sat}) \times P(\text{the} \mid \text{The cat sat on}) \\ &\times P(\text{mat} \mid \text{The cat sat on the}) \end{aligned} \tag{1.3}$$

Each factor in this product requires estimating a conditional distribution over the entire vocabulary, conditioned on all preceding words in the sequence. For sentences of length T with vocabulary size $|\mathcal{V}| = V$, the naive approach would require storing V^T parameters to represent every possible sequence of that length—clearly intractable even for modest vocabulary sizes and sentence lengths. Consider a vocabulary of 50,000 words and sentences of just 10 words: the naive approach would require $50,000^{10}$ parameters, a number far exceeding the estimated 10^{80} atoms in the observable universe. This exponential explosion in the number of possible sequences is sometimes called the “curse of dimensionality,” and it represents the central computational challenge of language modeling. All language modeling techniques, from the simplest n-gram models that consider only local context to the most sophisticated Transformer architectures that attend over thousands of tokens, address this fundamental challenge through parameter sharing and structural assumptions that dramatically reduce the effective number of parameters while still capturing the essential patterns in language. The art of language modeling lies in choosing the right inductive biases—assumptions about which patterns in the data actually matter for prediction—that allow robust generalization from the finite training data to the infinite space of possible sentences that a model may encounter in deployment.

1.2 A Brief History: From Shannon to ChatGPT

The quest to predict the next word has a rich history spanning over seven decades [Jurafsky and Martin, 2024]. Figure 1.3 presents a visual timeline of key developments.

1.2.1 The Information-Theoretic Foundation (1948–1960s)

Claude Shannon’s seminal 1948 paper “A Mathematical Theory of Communication” [Shannon, 1948] introduced the concept of *entropy* as a mathematical measure of uncertainty, providing the theoretical foundation that would underpin all subsequent work in language modeling and communication systems alike. Shannon asked a deceptively simple question: How much information does each character in English text convey? This question connected the structure of language to the mathematics of information transmission, revealing that the redundancy and predictability inherent in natural language could be precisely quantified in bits—the same units used to measure digital information. Shannon’s insight was that uncertainty and information are two sides of the same coin: when we can predict the next symbol with high confidence, that symbol carries little information because it tells us nothing we did not already expect; conversely, when a symbol surprises us by deviating from our expectations, it carries a great deal of information precisely because it could not have been anticipated. This fundamental inverse relationship between predictability and information content remains central to how we understand and evaluate language models today: a model that achieves low entropy has learned to predict well, which means it has captured the statistical structure of the language.

In a famous 1951 experiment [Shannon, 1951], Shannon had human subjects guess letters one at a time in English text, exploiting the predictive capabilities of native speakers to estimate the inherent uncertainty of the language. When subjects guessed incorrectly, they were told the correct letter and continued from that point, building up context that improved subsequent predictions. The experimental methodology was elegant in its simplicity: by counting the number of guesses required at each position and analyzing the distribution of correct responses, Shannon could bound the entropy from both above and below, producing tight estimates of the information content of English. By analyzing how many guesses subjects required on average and applying information-theoretic bounds, Shannon estimated that English has approximately 1.0–1.5 bits of entropy per character—meaning each character conveys roughly one bit of information on average, far below the theoretical

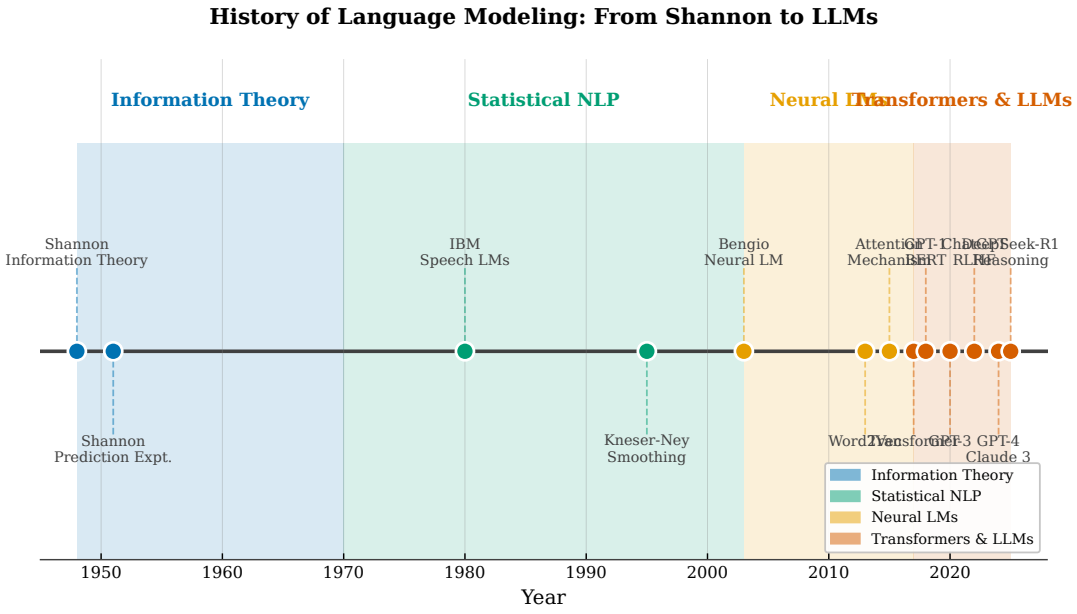

Figure 1.3: History of language modeling from 1948 to 2025. Panel (a) shows the timeline of key milestones. Panel (b) illustrates exponential growth in model parameters. Panel (c) displays corresponding improvements in perplexity. Panel (d) marks paradigm shifts in methodology.

maximum of about 4.7 bits that would be required for a uniformly random selection from the 26-letter alphabet. This finding revealed that English is highly redundant: roughly 75 percent of the information capacity of written text is predictable from context, leaving only about one bit per character of genuine unpredictability. This redundancy is why we can understand text with missing letters, why compression algorithms work so well on natural language, and why language models can achieve such impressive prediction accuracy.

Example 1.1 (Shannon's Guessing Game)**.** *Consider guessing letters in the word "INFORMATION" as shown in Figure 1.4. After observing the prefix "INFORM," a guesser familiar with English word patterns would likely predict "A" as the next letter, recognizing the common morphological pattern leading toward "INFORMATION" or "INFORMAL." Once "INFORMA" has been revealed, the prediction becomes nearly certain: "T" is the only plausible continuation in English, as "INFORMAT" leads uniquely toward "INFORMATION." This example illustrates the fundamental relationship between predictability and information content. When a letter is highly predictable—as "T" is after "INFORMA"—it conveys little new information to the reader, since the reader could have anticipated it with high confidence. Conversely, surprising or unexpected letters carry more information precisely because they could not have been predicted. This inverse relationship between predictability and information content lies at the heart of Shannon's information theory: high predictability corresponds to low entropy, which in turn means less information is conveyed per symbol.*

Shannon's experiments established a fundamental bound that continues to serve as a benchmark for language model evaluation: any language model for English must achieve at least 1.0–1.5 bits per character to match human prediction ability, reflecting the inherent entropy of written English as estimated through human performance. This bound represents a theoretical floor—the uncertainty that remains even when a predictor has access to all available contextual information and perfect knowledge of English, arising from the genuine unpredictability of natural language where multiple continuations may be equally valid. Remarkably, modern large language models approach and sometimes surpass this bound [Brown et al., 2020], achieving bits-per-character rates that rival or exceed the best human performance measured in Shannon's experiments over seven decades ago. This achievement suggests that these models have learned to capture most of the statistical regularities that human readers exploit when predicting text, a remarkable feat considering that the models learn entirely from raw text without explicit linguistic instruction. The convergence of machine and human performance on this fundamental task represents one of the great accomplishments of artificial intelligence research, though

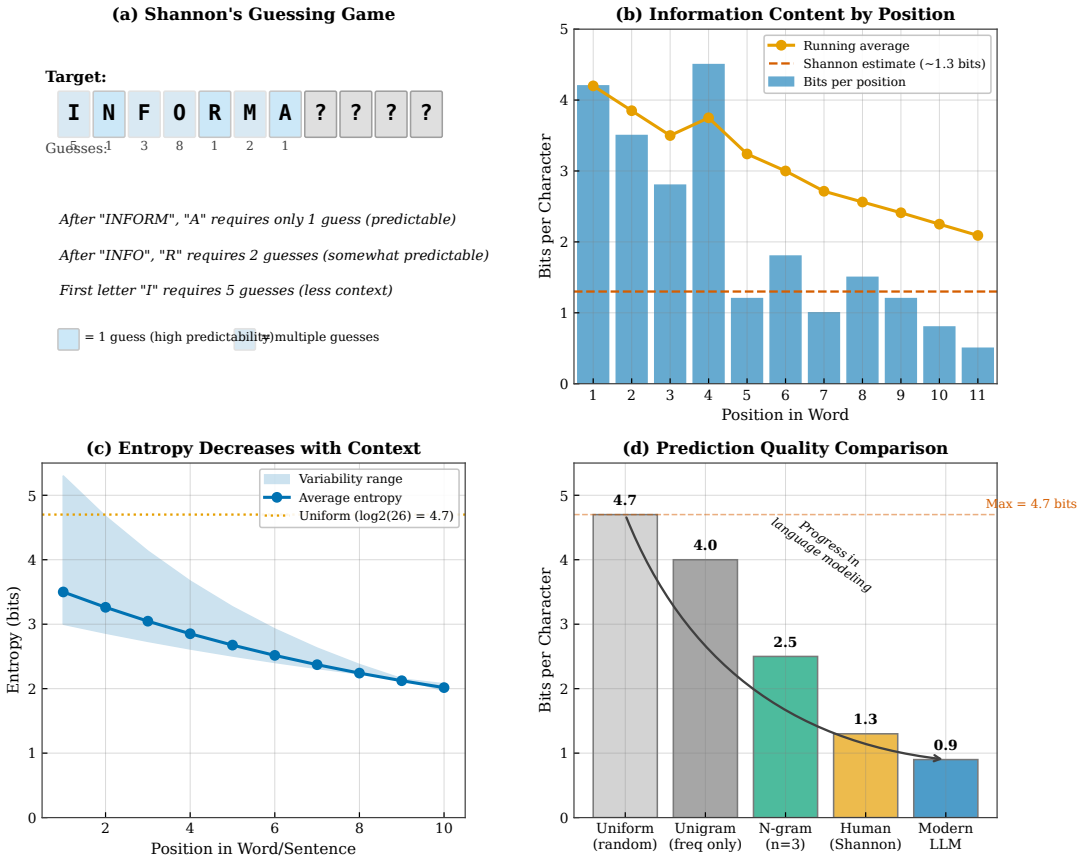

Figure 1.4: Shannon's guessing game experiment. Panel (a) shows the experimental setup where subjects guess letters sequentially. Panel (b) displays guess distributions for different contexts. Panel (c) illustrates how predictability varies with context. Panel (d) shows the relationship between guesses needed and entropy.

debates continue about whether this statistical competence reflects genuine language understanding or merely sophisticated pattern matching.

1.2.2 Statistical Methods (1970s–2000s)

The practical development of language models began with speech recognition research at IBM [Jelinek, 1990], where researchers discovered that adding a statistical language model to acoustic models dramatically improved transcription accuracy by guiding the recognizer toward word sequences that sounded similar but differed in linguistic plausibility. When acoustic evidence is ambiguous between "recognize speech" and "wreck a nice beach," the language model supplies the prior probability that makes the correct interpretation far more likely. This discovery launched decades of research into statistical language modeling, transforming it from a theoretical curiosity into an essential component of practical systems. **N-gram models**—which estimate $P(w_t \mid w_{t-n+1}, \ldots, w_{t-1})$ from simple word counts in a training corpus—became the dominant approach for decades due to their simplicity, computational efficiency, and surprisingly strong performance on many tasks despite their severe limitations. The key insight behind n-gram models is the Markov assumption: rather than conditioning on the entire preceding context which would be computationally intractable, we approximate by conditioning only on the previous $n-1$ words, making the estimation problem tractable while still capturing the local dependencies that account for much of language's predictability within short spans of text.

Several key developments during this era transformed language modeling from a theoretical curiosity into a practical technology. Good-Turing estimation, introduced by Good [1953], provided a principled method for handling rare events by using the frequency of frequencies to estimate probabilities for unseen n-grams. Building on this foundation, Kneser and Ney [1995] developed Kneser-Ney smoothing, which remains one of

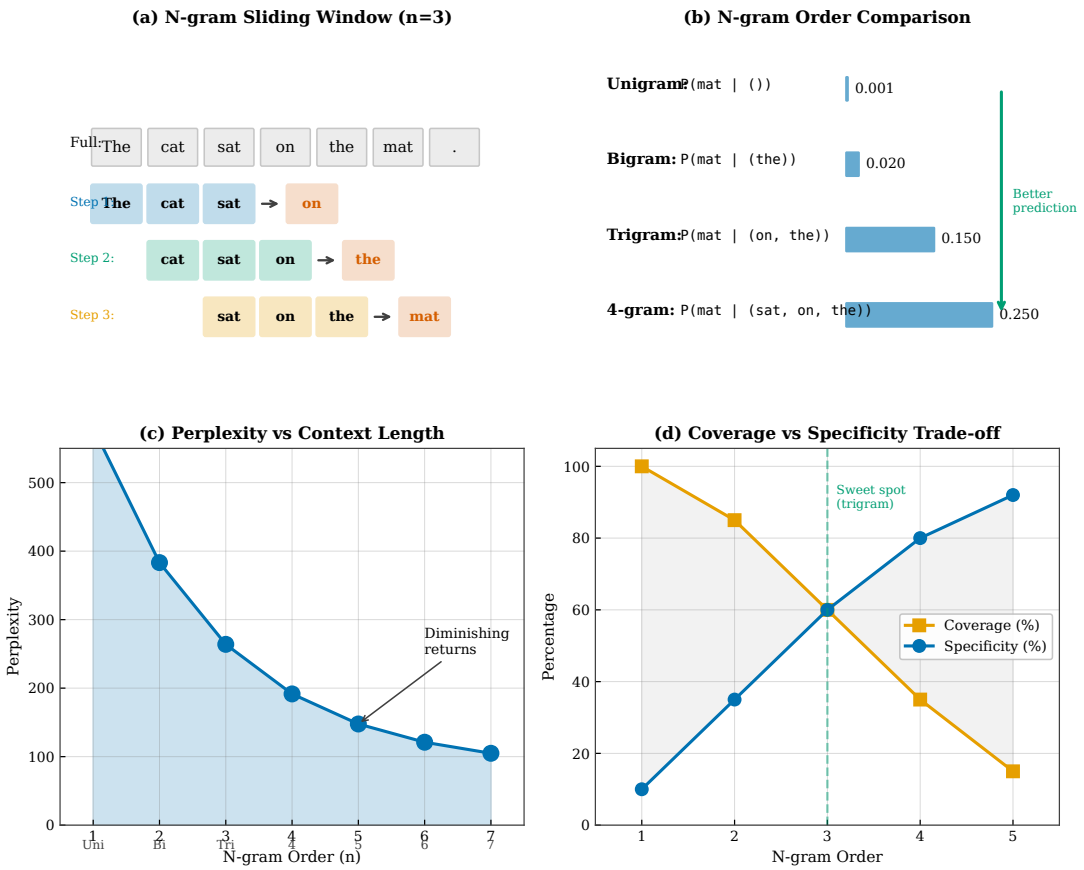

Figure 1.5: N-gram sliding window approach. Panel (a) shows unigram, bigram, and trigram windows. Panel (b) illustrates context limitation. Panel (c) displays the trade-off between context size and sparsity. Panel (d) demonstrates how n-grams miss long-range dependencies.

the most effective smoothing techniques for n-gram models by combining absolute discounting with a sophisticated backoff mechanism that considers how many distinct contexts a word appears in. The computational infrastructure of this era also advanced dramatically, enabling researchers to train language models on billions of words—a scale that would have been unimaginable in Shannon's time. These larger training corpora led to significant improvements in perplexity, demonstrating the fundamental importance of data scale for language modeling. Perhaps most importantly, this period saw the successful integration of language models with speech recognition systems at IBM and AT&T Bell Labs, and later with statistical machine translation systems. These applications demonstrated that language models had practical value beyond theoretical interest, setting the stage for the explosion of NLP applications that followed. Figure 1.5 illustrates how n-gram models use fixed-size context windows, trading off context length against data sparsity.

1.2.3 The Neural Revolution (2003–2017)

Bengio et al. [2003] introduced the first neural language model, representing words as dense vectors (*embeddings*) and using a feedforward network to predict the next word. This foundational work demonstrated that neural networks could learn meaningful representations of words—vectors in which semantically similar words cluster together—while simultaneously learning to predict text. The ideas from this paper sparked a revolution in natural language processing that unfolded over the following decade and a half, fundamentally changing how researchers approached the problem of modeling language. Mikolov et al. [2013] dramatically scaled these ideas with Word2Vec, demonstrating that word embeddings trained on massive corpora exhibited remarkable algebraic properties, such as the famous "king - man + woman = queen" analogy that captured semantic relationships through vector arithmetic. For sequence modeling, recurrent neural networks and particularly Long

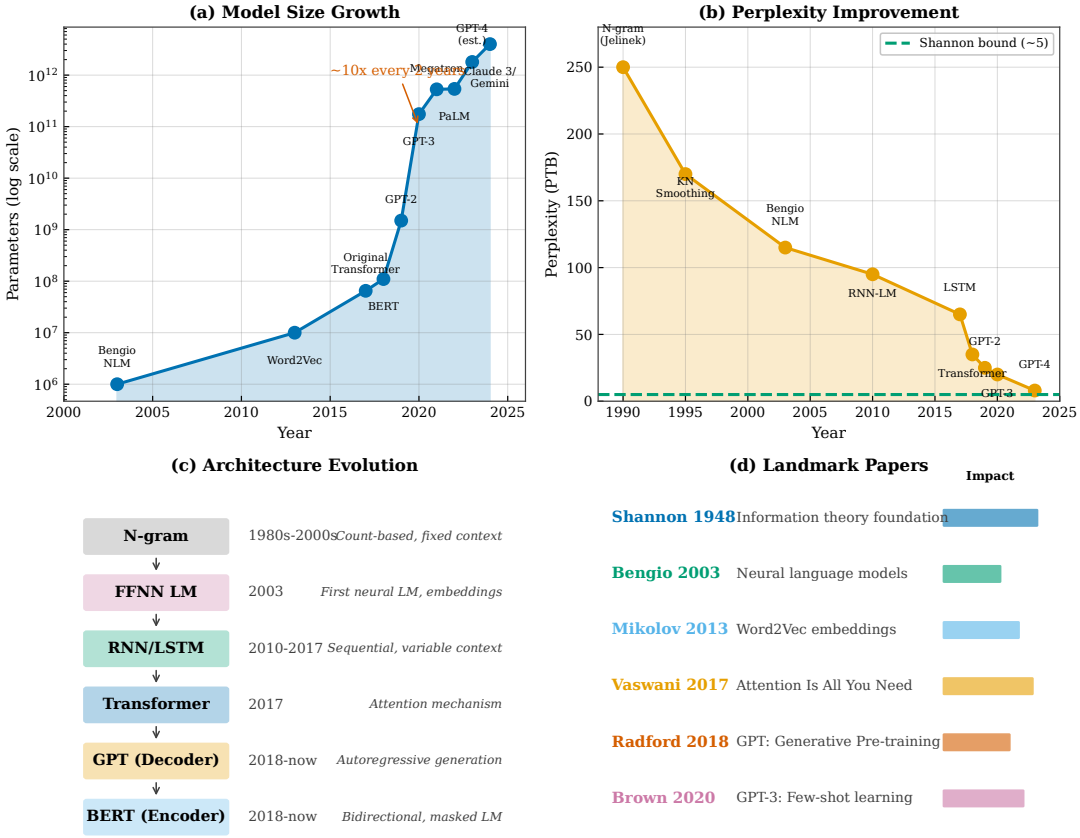

Figure 1.6: Evolution of language modeling architectures. Panel (a) shows the progression from count-based to neural approaches. Panel (b) displays architectural improvements over time. Panel (c) illustrates the growth in model capacity. Panel (d) demonstrates capability emergence across paradigms.

Short-Term Memory networks [Hochreiter and Schmidhuber, 1997] provided a way to process variable-length contexts by maintaining a hidden state that accumulates information over time, overcoming the fixed-context limitation of n-gram models and enabling models to capture dependencies across arbitrary distances. The final breakthrough came with the attention mechanism [Bahdanau et al., 2015], which allowed models to dynamically focus on relevant parts of the input, and its culmination in the Transformer architecture [Vaswani et al., 2017], which replaced recurrence entirely with self-attention and enabled unprecedented parallelization during training.

1.2.4 The Era of Large Language Models (2018–Present)

The Transformer architecture [Vaswani et al., 2017] enabled training on unprecedented scales by replacing sequential recurrent computation with parallel self-attention operations, allowing researchers to leverage modern GPU hardware to train models with billions of parameters on datasets containing hundreds of billions of tokens. This architectural innovation was transformative: while RNNs processed sequences one token at a time, requiring $O(n)$ sequential operations for a sequence of length n, Transformers could process all positions simultaneously, dramatically accelerating training and enabling the scaling that would define the following era. GPT-2 [Radford et al., 2019] demonstrated coherent multi-paragraph text generation that could maintain topic consistency and stylistic coherence over hundreds of words, surprising researchers with its ability to generate plausible news articles and stories; GPT-3 [Brown et al., 2020] showed emergent few-shot learning capabilities where the model could perform new tasks from just a handful of examples provided in the prompt, without any gradient updates to the model parameters. Subsequent work introduced instruction tuning [Ouyang et al., 2022], which fine-tuned models to follow explicit instructions, and preference optimization [Rafailov et al.,

2023], which aligned model outputs with human preferences through direct learning from human feedback. By 2024–2025, models like GPT-4, Claude, LLaMA [Touvron et al., 2023], and reasoning models like DeepSeek-R1 [DeepSeek-AI, 2025] exhibit sophisticated reasoning, multi-step problem solving, and natural dialogue abilities that would have seemed like science fiction just a decade earlier.

Figure 1.6 traces this evolution, showing how each paradigm shift brought qualitative improvements in prediction capability and enabled entirely new applications of language technology.

1.3 Information Theory Foundations

Understanding language modeling requires key concepts from information theory [Shannon, 1948], the mathematical framework that Shannon developed to study communication systems and that has proven equally applicable to the study of natural language. These concepts provide both the theoretical foundation for evaluating language models and deep intuition about the task's inherent difficulty. Information theory answers questions such as: How do we measure uncertainty? How do we quantify how well a model predicts? What is the theoretical limit of prediction accuracy? The answers to these questions—entropy, cross-entropy, and perplexity—form the vocabulary that researchers use to discuss language model quality, and understanding them is essential for anyone working in this field. The mathematical elegance of information theory lies in its ability to unify seemingly disparate phenomena under a common framework: the same concepts that describe optimal data compression apply equally to evaluating language models, because both tasks fundamentally involve predicting sequences of symbols from a probabilistic source. This section develops the core information-theoretic concepts that will recur throughout this book, providing the mathematical tools needed to precisely measure and compare language model performance.

1.3.1 Entropy

Definition 1.2 (Entropy)**.** *The* entropy *of a discrete random variable X with distribution $p(x)$ is:*

$$H(X) = -\sum_{x} p(x) \log_2 p(x) \tag{1.4}$$

measured in bits.

Entropy measures uncertainty or unpredictability in a probability distribution, quantifying how much "surprise" we should expect on average when sampling from that distribution. For a fair coin, $H(X) = 1$ bit, meaning we need exactly one yes/no question to resolve the uncertainty—will it be heads or tails? For a biased coin with $P(\text{heads}) = 0.9$, $H(X) \approx 0.47$ bits—less uncertain, so less entropy, because we already have a strong expectation that heads will appear. In the extreme case of a completely deterministic outcome, entropy equals zero: there is no uncertainty to resolve. The unit of entropy, the bit, corresponds to the uncertainty of a single fair coin flip—this provides an intuitive anchor for understanding larger entropy values. A vocabulary of 50,000 words under a uniform distribution would have entropy of approximately 15.6 bits, representing the information needed to specify one word among 50,000 equally likely alternatives. For language modeling, entropy captures how uncertain we are about the next word given the context, and the goal of training is to reduce this uncertainty by learning which words are more or less likely in different contexts. A well-trained language model dramatically reduces this entropy by concentrating probability mass on a small set of plausible continuations.

Theorem 1.1 (Maximum Entropy)**.** *For a discrete random variable X taking n possible values, entropy is maximized when X is uniformly distributed:*

$$H(X) \leq \log_2 n \tag{1.5}$$

with equality if and only if $p(x) = 1/n$ for all x.

Figure 1.7 visualizes entropy as a surface over probability distributions, showing how it peaks at the uniform distribution and decreases as the distribution becomes more concentrated.

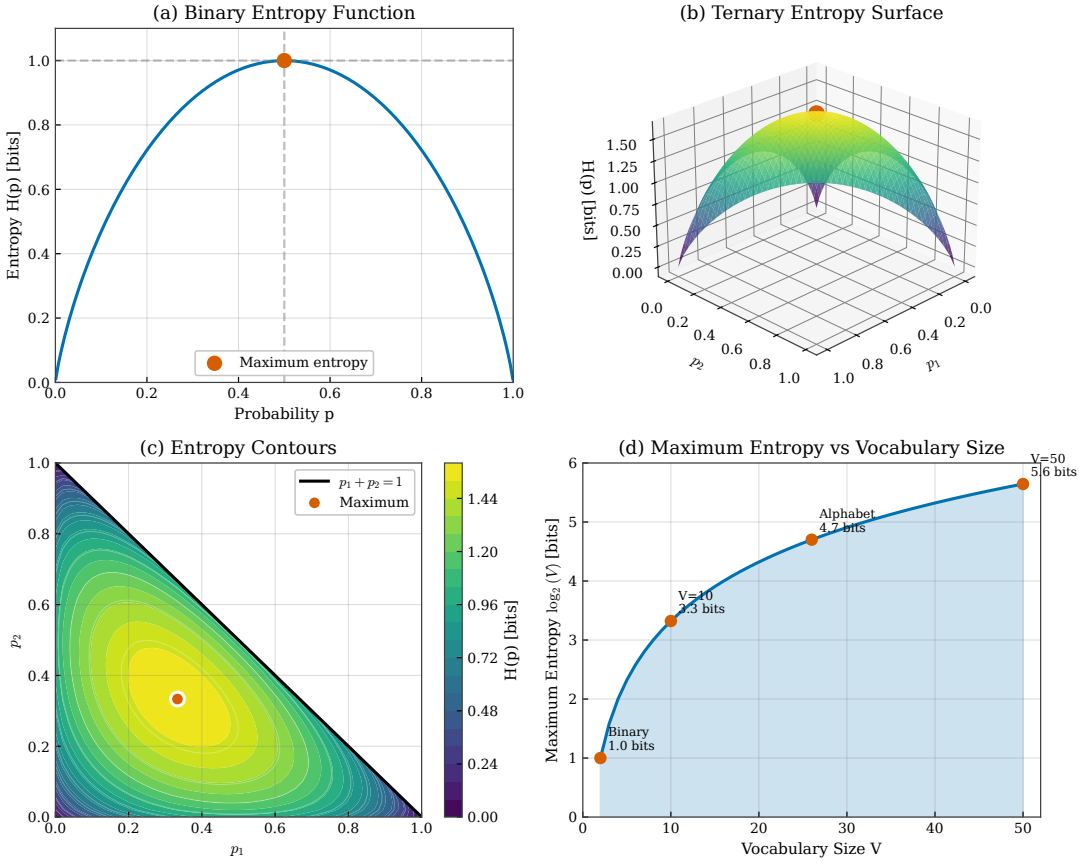

Figure 1.7: Entropy as a function of probability distribution. Panel (a) shows the binary entropy function $H(p) = -p\log p - (1-p)\log(1-p)$. Panel (b) displays a 3D entropy surface for three-outcome distributions. Panel (c) illustrates maximum entropy at uniform distribution. Panel (d) shows entropy contours on the probability simplex.

1.3.2 Cross-Entropy and Perplexity

When we have a model distribution q approximating the true distribution p:

Definition 1.3 (Cross-Entropy)**.**

$$H(p,q) = -\sum_x p(x)\log_2 q(x) \tag{1.6}$$

The cross-entropy measures how well q predicts samples from p. Figure 1.8 visualizes cross-entropy as a loss function, showing how it heavily penalizes confident wrong predictions. Cross-entropy equals the entropy $H(p)$ plus the KL divergence $D_{\mathrm{KL}}(p\,\|\,q)$:

$$H(p,q) = H(p) + D_{\mathrm{KL}}(p\,\|\,q) \tag{1.7}$$

so $H(p,q) \geq H(p)$ with equality when $q = p$.

For language modeling, we evaluate by computing the cross-entropy on a test corpus:

$$H(\text{corpus},\text{model}) = -\frac{1}{T}\sum_{t=1}^{T}\log_2 P(w_t\,|\,w_1,\ldots,w_{t-1}) \tag{1.8}$$

Definition 1.4 (Perplexity)**.**

$$PPL = 2^{H(corpus,model)} \tag{1.9}$$

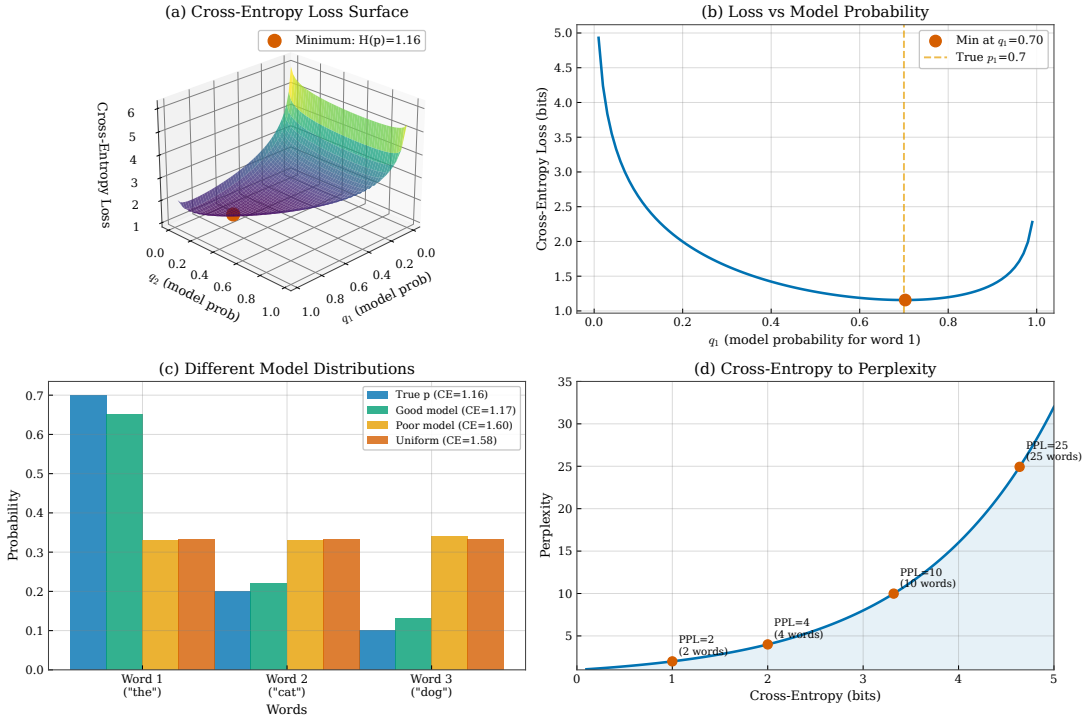

Figure 1.8: Cross-entropy as a loss function. Panel (a) shows the cross-entropy surface as a function of predicted probability. Panel (b) displays the gradient of cross-entropy loss. Panel (c) illustrates how cross-entropy penalizes confident wrong predictions. Panel (d) compares cross-entropy with other loss functions.

Perplexity can be interpreted as the effective vocabulary size over which the model spreads its probability mass: a perplexity of 100 means the model is "as confused" as if uniformly choosing among 100 equally likely words at each position, while a perplexity of 10 indicates concentration on roughly 10 plausible options. This interpretation makes perplexity intuitive: instead of thinking about bits and logarithms, we can think about how many words the model is effectively considering at each step. Lower perplexity indicates better prediction, with the theoretical minimum of 1 achieved only when the model assigns probability 1 to the correct word at every position—perfect prediction with zero uncertainty. State-of-the-art language models on benchmarks like WikiText-103 achieve perplexities below 10, meaning they effectively narrow down each prediction to about 10 plausible candidates on average, a remarkable feat given vocabularies containing tens of thousands of words. The progression from early n-gram models with perplexities around 100–200 to modern Transformers with perplexities below 10 represents a fundamental advance in our ability to capture the statistical structure of language, corresponding to an order-of-magnitude reduction in the effective uncertainty at each prediction step.

Example 1.2 (Perplexity Interpretation)**.** *Consider predicting the next word after "The capital of France is" as illustrated in Figure 1.10. A uniform model that assigns equal probability to all 50,000 words in its vocabulary would have a perplexity of exactly 50,000, reflecting maximum uncertainty—the model has no ability to distinguish likely continuations from unlikely ones. In contrast, a model that correctly recognizes this context and assigns probability 0.8 to "Paris" while distributing the remaining 0.2 probability across other words would achieve a perplexity of approximately 1.5, indicating that on average the model is nearly as certain as if it were choosing between just 1–2 equally likely words. A perfect model that assigns probability 1 to the correct word at every position would achieve perplexity 1, representing complete certainty. This example demonstrates why perplexity serves as such an intuitive metric: it directly measures how "confused" the model is, with lower values indicating greater predictive confidence. The logarithmic relationship between perplexity and probability means that improvements from 100 to 50 perplexity represent the same relative gain as improvements from 50 to 25.*

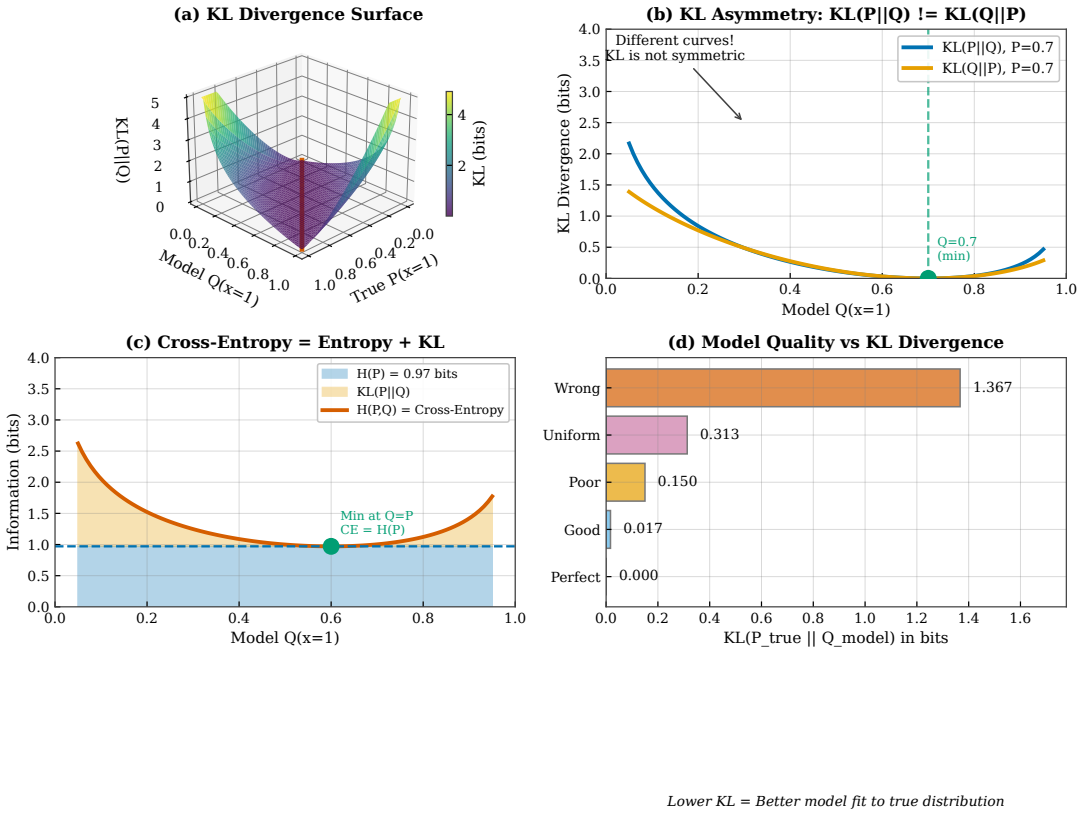

Figure 1.9: KL divergence visualization. Panel (a) shows asymmetry: $D_{\mathrm{KL}}(p \,\|\, q) \neq D_{\mathrm{KL}}(q \,\|\, p)$. Panel (b) displays a 3D surface of KL divergence. Panel (c) illustrates the relationship between cross-entropy and KL divergence. Panel (d) demonstrates mode-covering vs. mode-seeking behavior.

1.3.3 Bits Per Character

An alternative metric that normalizes for vocabulary differences is **bits per character (BPC)**:

$$\mathrm{BPC} = \frac{1}{C} \sum_{t=1}^{T} \log_2 \frac{1}{P(w_t \mid w_{<t})} \tag{1.10}$$

where C is the total number of characters in the corpus, effectively normalizing the information content by the length of the raw text rather than by the number of tokens. BPC is particularly useful for comparing models with different tokenization schemes: a character-level model predicting individual letters, a word-level model predicting whole words, and a subword model using byte-pair encoding would all have different perplexities that are not directly comparable, but their BPC values can be meaningfully compared because they measure the same underlying quantity—the information required to encode each character of the original text. This normalization is crucial in modern NLP research where tokenization strategies vary widely: GPT-2 and GPT-3 use byte-pair encoding with different vocabulary sizes, BERT uses WordPiece tokenization, and some models operate directly on characters or bytes. Without a common metric like BPC, comparing these models would require making arbitrary assumptions about how to convert between token-level and character-level measurements. BPC also connects directly to Shannon's entropy estimates from human experiments, enabling meaningful comparison between machine and human language prediction capabilities across seven decades of research.

Shannon's experiments suggest human-level English prediction corresponds to approximately 1.0–1.5 BPC. Figure 1.11 shows how modern language models have approached and, in some cases, surpassed this bound.

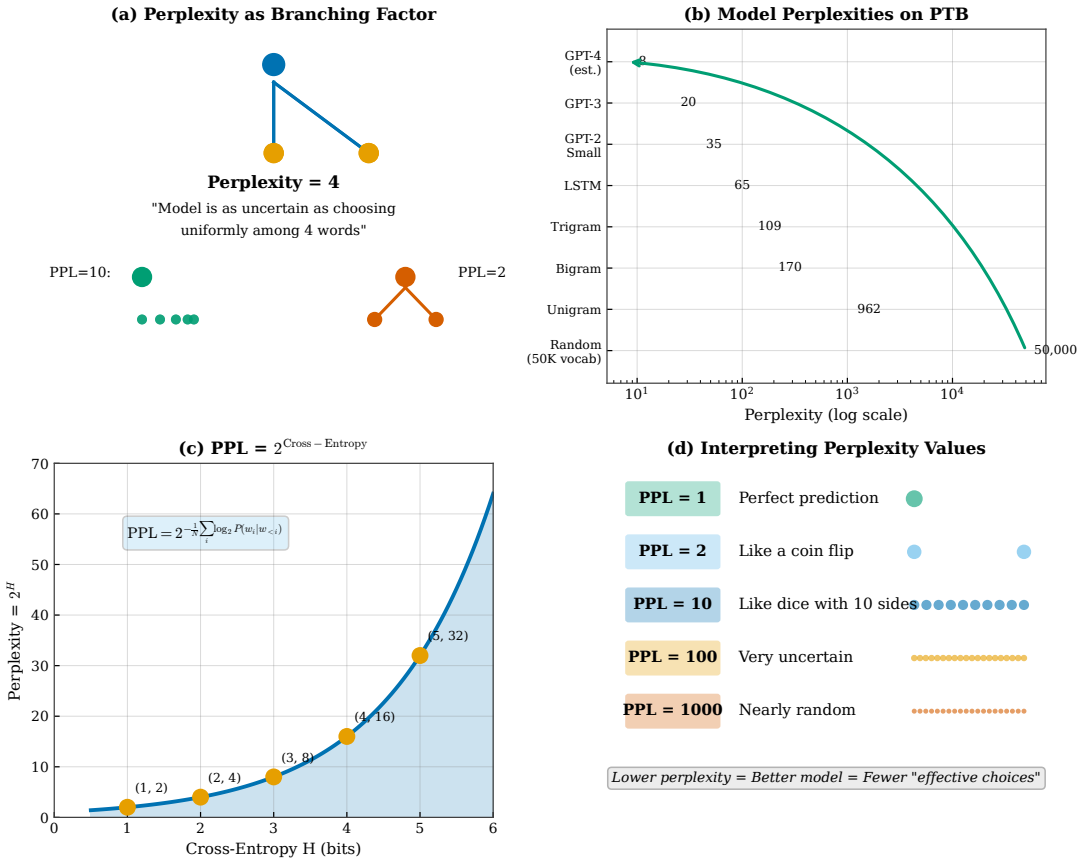

Figure 1.10: Interpreting perplexity. Panel (a) shows perplexity as effective vocabulary size. Panel (b) displays the relationship between perplexity and prediction confidence. Panel (c) compares perplexities of different models. Panel (d) illustrates how perplexity varies across text difficulty.

1.3.4 The Log Probability Space

Working with probabilities directly leads to numerical underflow for long sequences. We instead work in *log probability space*:

$$\log P(w_1, \ldots, w_T) = \sum_{t=1}^{T} \log P(w_t \mid w_1, \ldots, w_{t-1}) \tag{1.11}$$

This transformation, illustrated in Figure 1.12, converts products to sums, avoiding the exponential decay of probability values. The negative log probability $-\log P(w_t \mid w_{<t})$ is the *surprisal* of word w_t—the number of bits needed to encode it given the context.

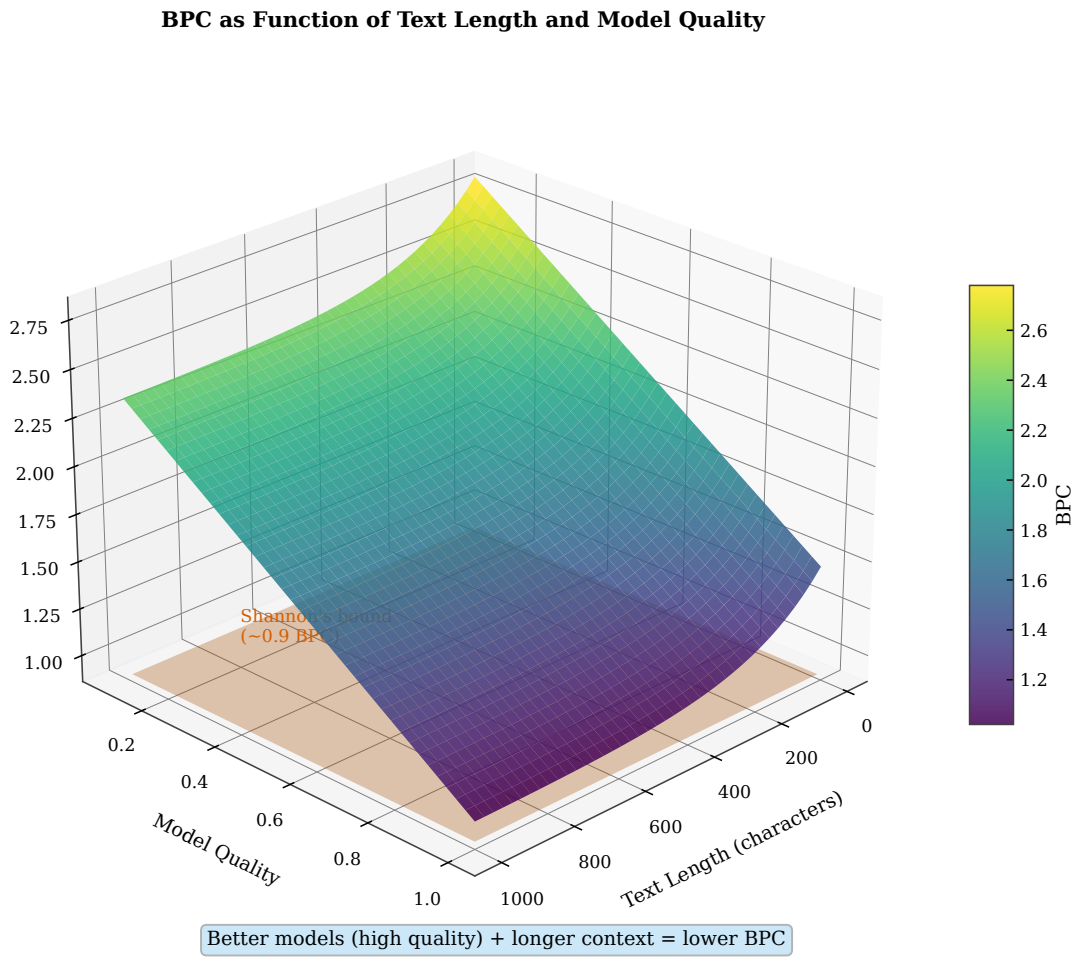

Figure 1.11: 3D surface showing bits per character (BPC) as a function of text length and model quality. Better models and longer contexts lead to lower BPC, approaching Shannon's theoretical bound of approximately 0.9 BPC for English.

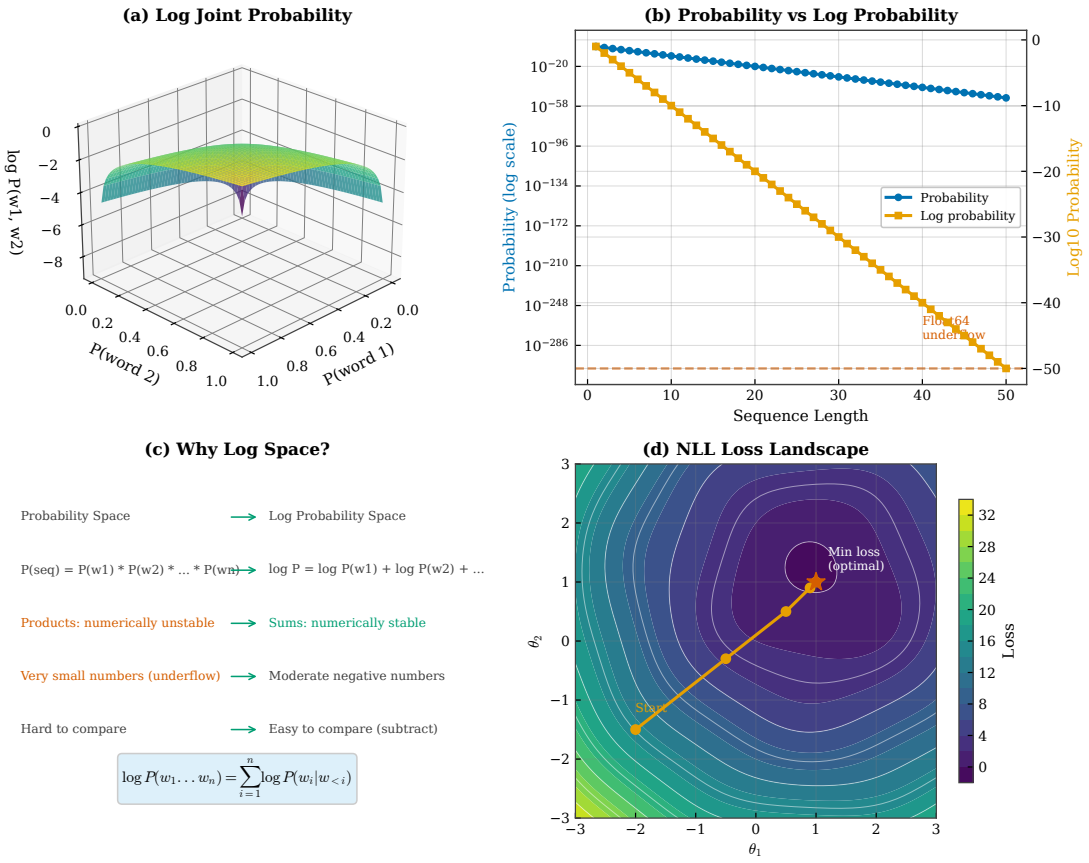

Figure 1.12: Working in log probability space. Panel (a) shows a 3D visualization of the log transformation. Panel (b) demonstrates why products become sums. Panel (c) illustrates numerical stability advantages. Panel (d) displays the log-sum-exp trick for computing normalizers.

1.4 The Probability Distribution Over Words

A language model must output a valid probability distribution over the vocabulary at each position, satisfying the fundamental axioms of probability theory: all values must be non-negative, and they must sum to exactly one across all possible next words. Understanding this constraint is essential for model design, as it determines how neural networks must transform their internal computations into valid probability outputs and influences architectural choices at every stage of the prediction pipeline. For a vocabulary of 50,000 words, the model must produce 50,000 non-negative numbers that sum to one—a high-dimensional constrained output that requires careful handling. This section explores the mathematical structure of probability distributions over discrete vocabularies, the softmax function that neural networks use to produce valid distributions, and the geometric interpretation of these constraints that helps build intuition for model behavior. The probability simplex provides the geometric setting for understanding these distributions, while the softmax function provides the computational mechanism for mapping unconstrained neural network outputs onto this constrained space. Together, these concepts illuminate the final stage of every neural language model: the projection from internal representations to probability distributions over words.

1.4.1 The Probability Simplex

Definition 1.5 (Probability Simplex)**.** *The* probability simplex Δ^{V-1} *is the set of all valid probability distributions over V outcomes:*

$$\Delta^{V-1} = \left\{ \mathbf{p} \in \mathbb{R}^V : p_i \geq 0, \sum_{i=1}^{V} p_i = 1 \right\} \tag{1.12}$$

Figure 1.13 visualizes the probability simplex for a three-word vocabulary. Every point on this triangular surface represents a valid probability distribution. The vertices represent deterministic predictions (probability 1 on one word), while the center represents the uniform distribution.

1.4.2 The Softmax Function

Neural language models produce *logits*—unconstrained real numbers for each vocabulary word that can be positive, negative, or zero—which must be converted to valid probabilities that are non-negative and sum to one. The term "logit" comes from the log-odds function in statistics, reflecting the interpretation of these values as unnormalized log-probabilities: higher logits correspond to words the model considers more likely, while lower logits correspond to less likely words, with the scale being completely arbitrary until normalization is applied. Raw logits reflect the model's internal scoring of each word's plausibility but do not satisfy probability axioms—they can be any real number, positive or negative, and they certainly do not sum to one. The **softmax function** accomplishes the essential transformation from unconstrained scores to valid probabilities, exponentiating each logit to ensure non-negativity and then normalizing by the sum to ensure the outputs sum to one. This two-step process—exponentiation followed by normalization—is computationally convenient because it allows the neural network to output any real numbers while guaranteeing valid probability distributions at the output, regardless of the input context or the learned model parameters:

$$\text{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_{j=1}^{V} \exp(z_j)} \tag{1.13}$$

The *temperature* parameter τ controls the sharpness of the distribution:

$$\text{softmax}(\mathbf{z}/\tau)_i = \frac{\exp(z_i/\tau)}{\sum_{j=1}^{V} \exp(z_j/\tau)} \tag{1.14}$$

As shown in Figure 1.14, the temperature parameter controls the entropy of the output distribution in a continuous manner. When temperature approaches zero, the softmax function converges to the argmax operation, placing all probability mass on the highest-scoring word and producing deterministic, greedy predictions that always select the single most likely token. At temperature equal to one, the standard softmax transformation

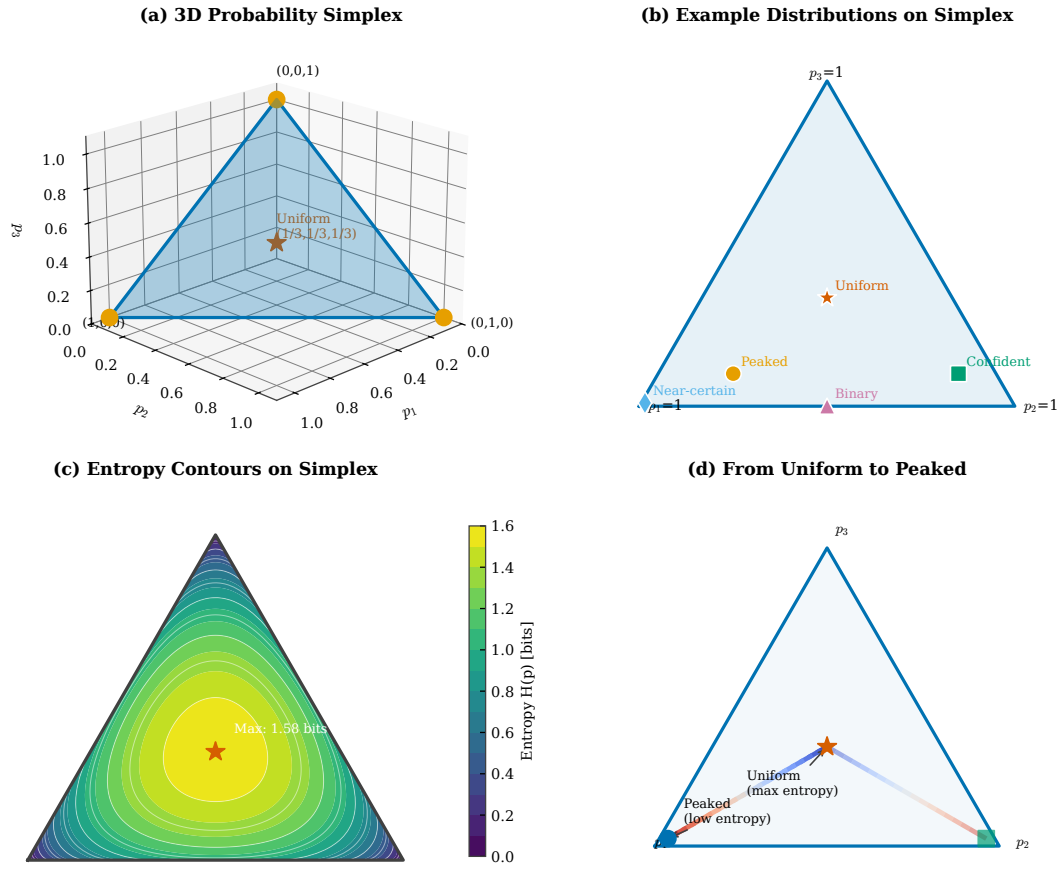

Figure 1.13: The probability simplex. Panel (a) shows the 3D simplex for three-word vocabulary. Panel (b) displays entropy contours on the simplex. Panel (c) illustrates how neural networks map to the simplex via softmax. Panel (d) demonstrates the effect of temperature on distribution sharpness.

applies, preserving the relative log-probability ratios from the logits exactly as the model learned them during training. As temperature increases beyond one, the distribution becomes progressively flatter, with the extreme case of temperature approaching infinity yielding a uniform distribution regardless of the input logits, where every word becomes equally likely and sampling becomes purely random. This temperature-controlled sharpening or smoothing proves essential for text generation, where practitioners balance the tension between coherent but repetitive low-temperature outputs and diverse but potentially incoherent high-temperature samples. The choice of temperature depends heavily on the application: code generation and formal writing benefit from low temperatures that favor precision, while creative writing and brainstorming benefit from higher temperatures that encourage novelty. Chapter 7 explores these decoding trade-offs in depth, examining how temperature interacts with other sampling strategies to produce high-quality generated text.

1.4.3 Conditional Probability Trees

Another perspective on language model outputs is the conditional probability tree, a powerful visualization that reveals the exponential structure of sequence generation. In this view, each node represents a partial sequence of words, and each edge represents the conditional probability of extending that sequence by one additional word. The root node represents the empty context (or a given prompt), and each path from root to leaf represents a complete generated sequence with total probability equal to the product of all edge weights along the path—this is simply the chain rule decomposition rendered as a tree. The branching factor at each node equals the vocabulary size, so even short sequences produce astronomically large trees: a vocabulary of 50,000 words and sequences of just 10 words yields $50,000^{10}$ possible paths, far more than could ever be enumerated explicitly.

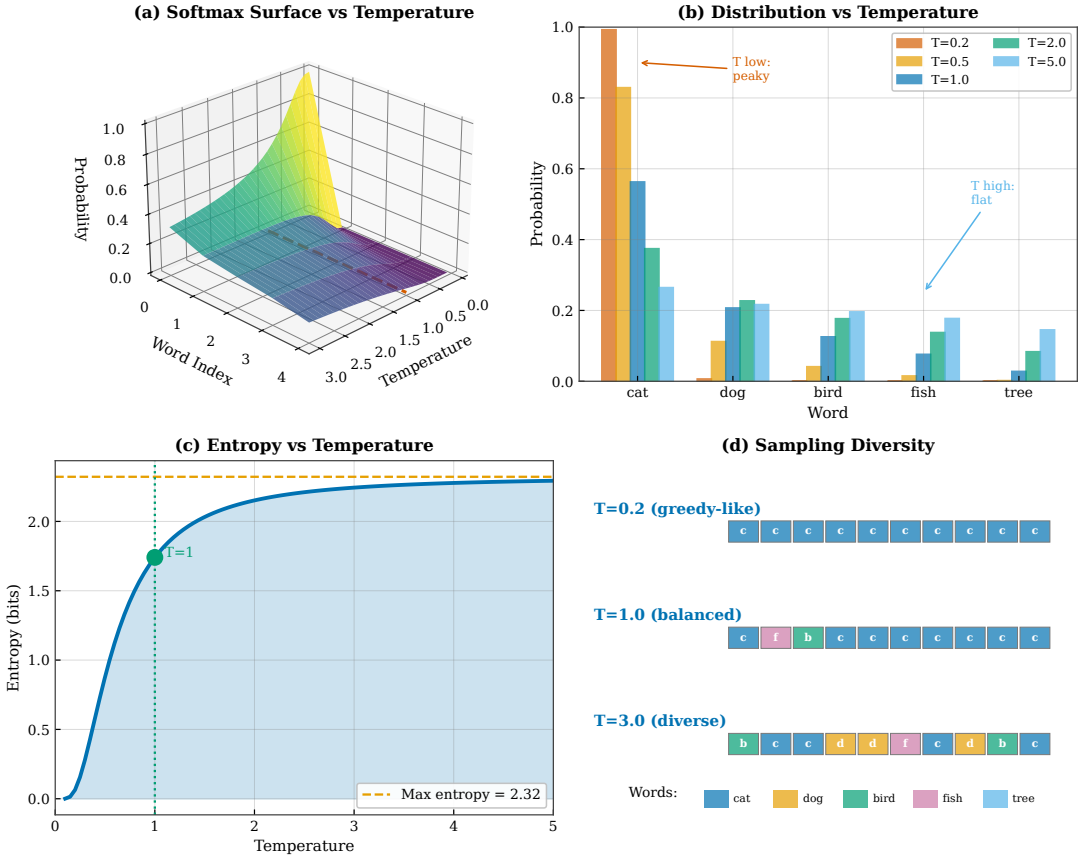

Figure 1.14: Softmax temperature effects. Panel (a) shows a 3D surface of softmax output as a function of logits and temperature. Panel (b) displays how temperature affects distribution entropy. Panel (c) illustrates low ($\tau < 1$) vs. high ($\tau > 1$) temperature behavior. Panel (d) demonstrates temperature's effect on sampling diversity.

This exponential explosion is both the challenge and the opportunity of language modeling: the challenge because exhaustive search is impossible, and the opportunity because a good language model concentrates probability mass on a tiny fraction of the possible sequences, making efficient decoding algorithms possible by pruning the low-probability branches.

Figure 1.15 shows how the conditional probability factorization creates a tree structure over all possible sequences. This view is essential for understanding decoding algorithms (Chapter 7).

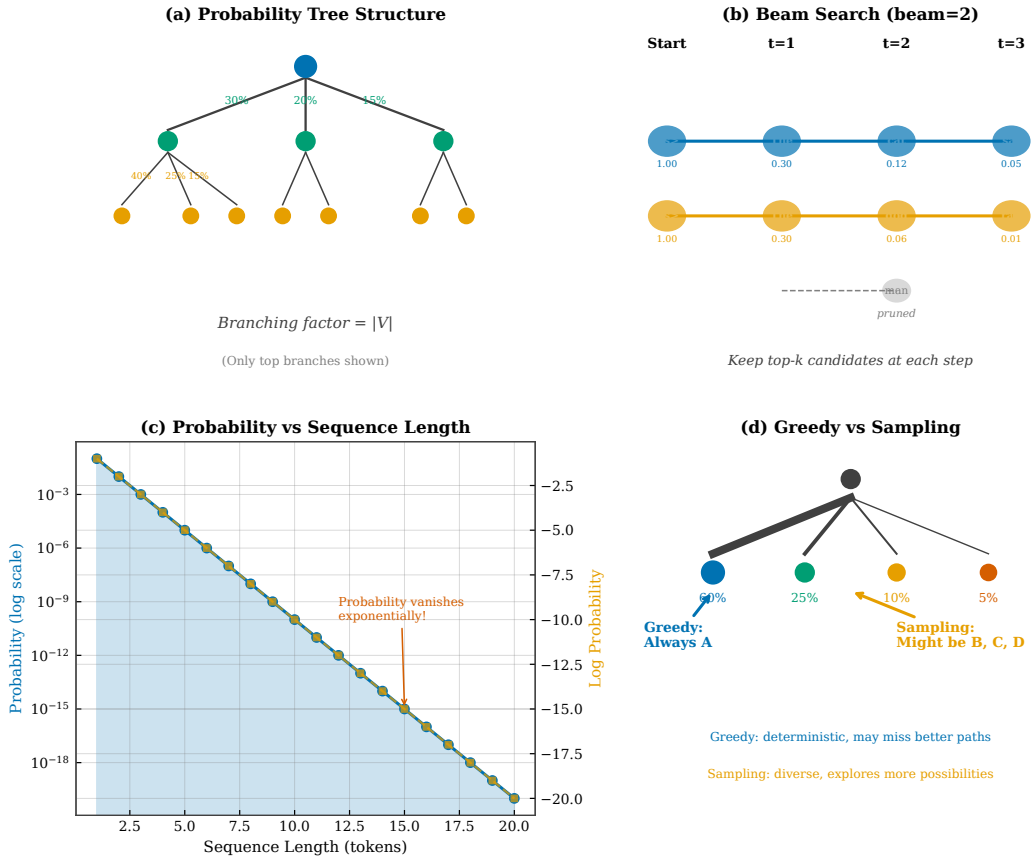

Figure 1.15: Conditional probability as a tree. Panel (a) shows the tree structure of sequence probabilities. Panel (b) illustrates how paths through the tree correspond to sentences. Panel (c) displays probability mass distribution across branches. Panel (d) demonstrates beam search traversal of the tree.

1.5 Linguistic Foundations

Language exhibits statistical regularities at multiple scales—from individual character sequences to discourse-level patterns spanning paragraphs—that language models must capture to make accurate predictions. Understanding these regularities helps us design better models by informing our choice of architecture, training data, and evaluation metrics, and it helps us interpret model behavior by revealing which linguistic phenomena a model has successfully learned versus which remain challenging. The statistical structure of language is remarkably consistent across different corpora and even across different languages, suggesting that some of these regularities reflect deep properties of human communication rather than arbitrary conventions of particular language communities. This section examines some of the fundamental statistical properties of natural language, from the heavy-tailed word frequency distribution described by Zipf's law to the hierarchical structure of linguistic knowledge from phonology through pragmatics. These linguistic foundations inform both the design of language models and the interpretation of their successes and failures: a model that fails to capture Zipf's law will waste capacity on rare words, while a model that ignores syntactic structure will make predictions that violate basic grammaticality constraints that any native speaker would recognize immediately.

1.5.1 Zipf's Law

Theorem 1.2 (Zipf's Law)**.** *The frequency of a word is approximately inversely proportional to its rank:*

$$f(r) \propto \frac{1}{r^{\alpha}} \tag{1.15}$$

where r is the rank and $\alpha \approx 1$ for natural language [Zipf, 1949].

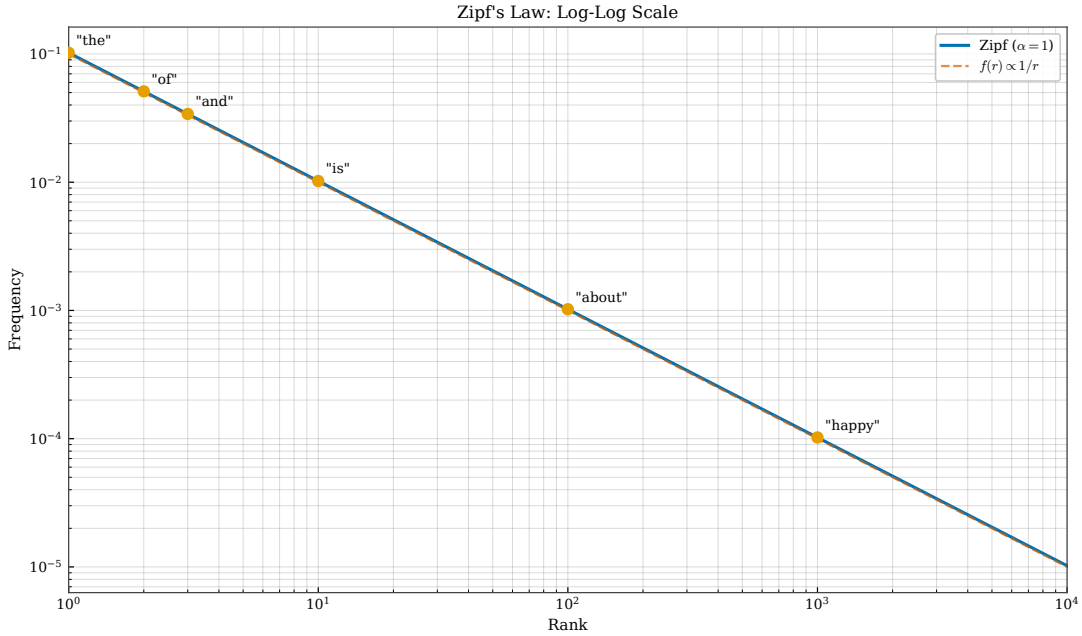

Figure 1.16: Zipf's law in natural language. Panel (a) shows the power-law relationship between rank and frequency on a log-log plot. Panel (b) displays the cumulative distribution. Panel (c) compares Zipf distributions across languages. Panel (d) illustrates implications for vocabulary coverage.

This power-law distribution, visualized in Figure 1.16, has profound implications for language modeling. The most striking consequence is the extreme concentration of probability mass: a small number of high-frequency words such as "the," "of," "and," and "to" account for a disproportionate fraction of all tokens in any corpus. In English text, the top 100 words typically cover more than 50 percent of all token occurrences, while the top 1,000 words cover approximately 80 percent. This concentration means that a language model encounters the same common words repeatedly during training, providing abundant data for learning their distributions. However, the flip side of this concentration is the "long tail" phenomenon: the vast majority of vocabulary items are rare, appearing only a handful of times even in billion-word corpora. A model trained on such data will inevitably encounter word combinations during evaluation that never appeared during training. This sparsity problem motivates the smoothing techniques discussed in Chapter 2, which redistribute probability mass from observed to unobserved events, as well as the subword tokenization methods of Chapter 3, which break rare words into more frequent constituent units.

1.5.2 The Vocabulary Problem

English contains hundreds of thousands of words in current use, plus proper names for people, places, and organizations, technical terminology from every field of human knowledge, and neologisms that emerge continuously as culture and technology evolve. The Oxford English Dictionary catalogs over 170,000 words in current use, and that figure excludes most technical jargon and proper nouns, which together easily push the effective vocabulary into the millions for any truly comprehensive language model. Any fixed vocabulary that a language model uses will inevitably encounter **out-of-vocabulary (OOV)** words during deployment—words that never appeared in the training data and thus have no learned representation. This vocabulary problem is fundamental to language modeling and has motivated significant research into solutions ranging from simple smoothing techniques to sophisticated subword tokenization methods. The challenge is particularly acute for languages with rich morphology, where a single root can generate dozens or hundreds of inflected forms, and for technical domains where specialized terminology proliferates. Even for English, new words enter the language constantly—"selfie," "cryptocurrency," and "COVID" were unknown just years before becoming ubiquitous—making the vocabulary problem an ongoing challenge rather than a one-time engineering decision.

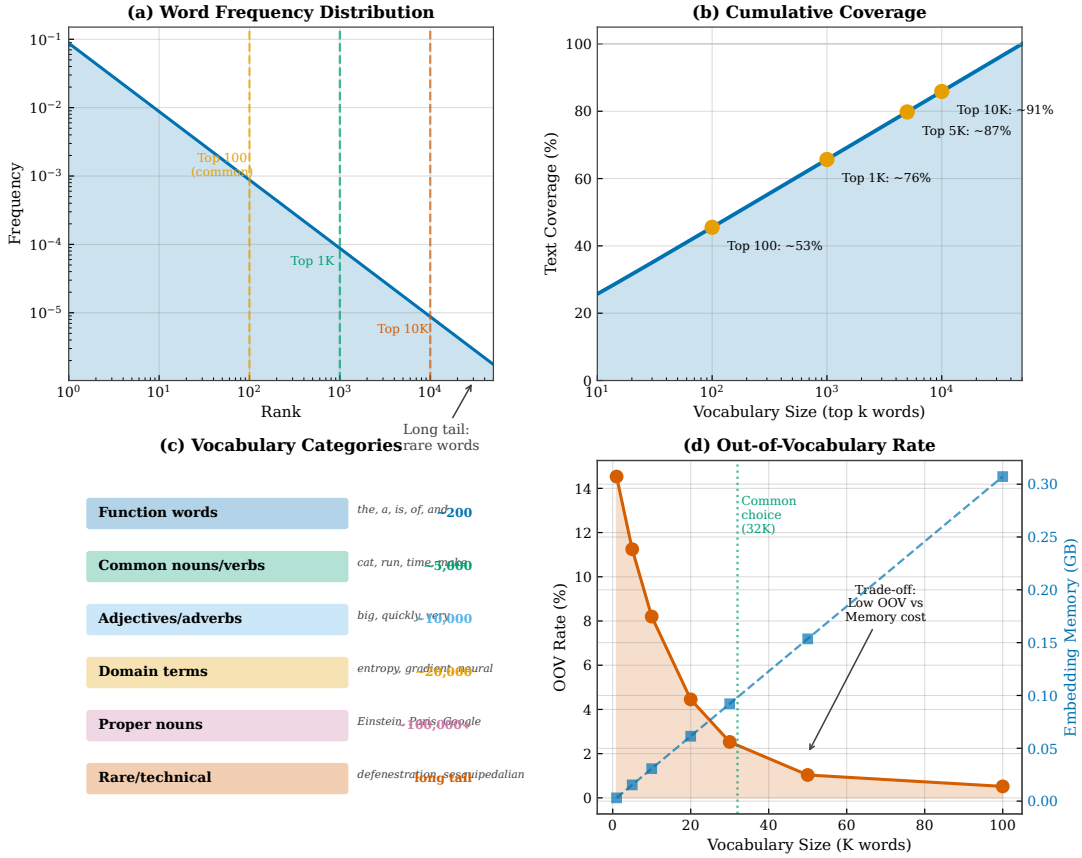

Figure 1.17: Vocabulary structure in language models. Panel (a) shows frequency-based vocabulary ordering. Panel (b) displays the relationship between vocabulary size and OOV rate. Panel (c) illustrates subword decomposition strategies. Panel (d) compares closed vs. open vocabulary approaches.

Figure 1.17 illustrates the vocabulary problem and the various solutions that have been developed to address it. The classical approach, detailed in Chapter 2, employs smoothing techniques that reserve a portion of probability mass for unseen words, typically by mapping all out-of-vocabulary items to a special unknown token and estimating its probability from held-out data. While simple and effective for moderate OOV rates, this approach loses all information about the specific unknown word encountered. A more sophisticated solution, which has become standard in modern systems and receives thorough treatment in Chapter 3, is subword tokenization. Methods such as Byte Pair Encoding and WordPiece decompose words into frequently occurring character sequences, so that even a never-before-seen word like "cryptocurrency" can be represented as a sequence of known subunits such as "crypto" and "currency." This approach effectively creates an open vocabulary: any string of characters can be tokenized, eliminating the OOV problem entirely while retaining morphological information that helps the model generalize to novel words. Neural language models have further embraced open-vocabulary approaches, sometimes operating directly at the character or byte level to achieve complete coverage of any possible input.

1.5.3 Language Structure and Prediction

Natural language has structure at multiple levels, each affecting prediction difficulty differently and requiring different types of knowledge to exploit effectively. These levels range from the subword phonological patterns that govern which letter sequences are pronounceable, through morphological rules for word formation, syntactic constraints on phrase and sentence structure, semantic requirements for meaningful interpretation, and pragmatic conventions for coherent discourse. A complete language model must capture patterns at all these levels, and the history of language modeling can be viewed as a progression toward architectures capable of

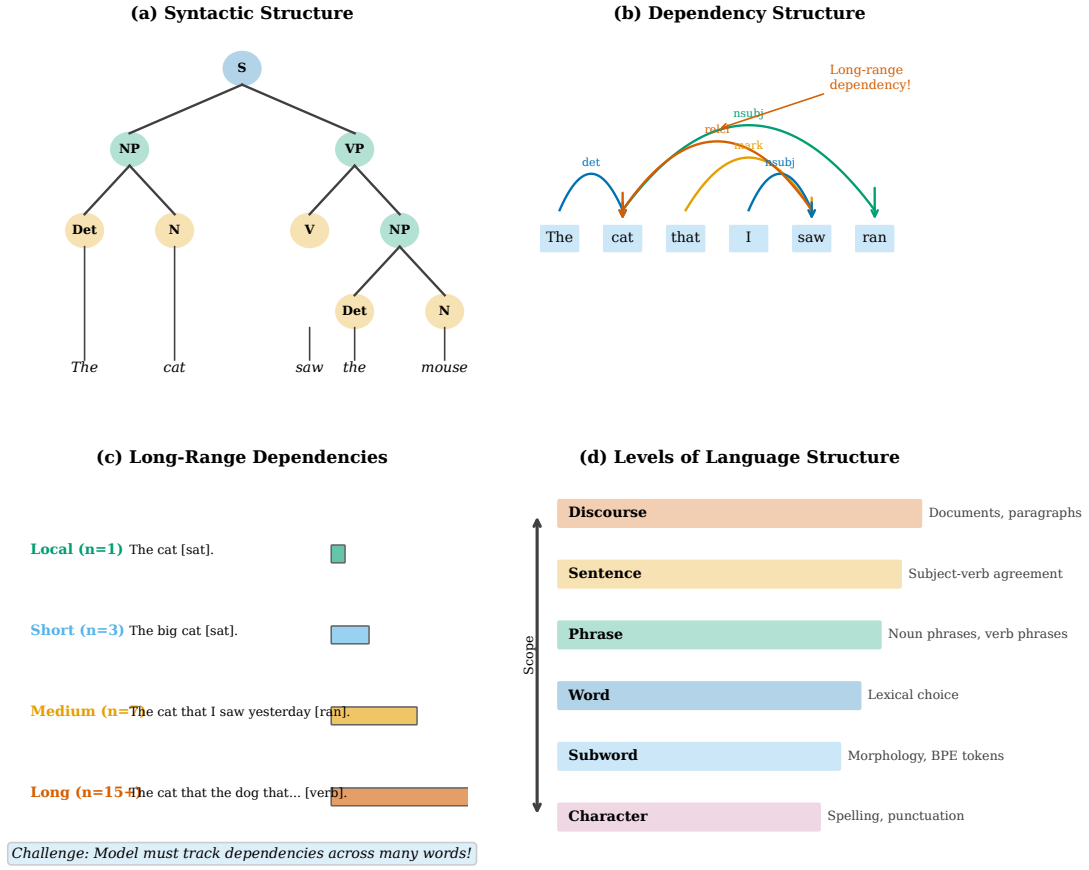

Figure 1.18: Language structure and prediction. Panel (a) shows phonological constraints on word sequences. Panel (b) displays morphological patterns. Panel (c) illustrates syntactic structure. Panel (d) demonstrates semantic coherence requirements.

learning increasingly sophisticated linguistic structure. Early n-gram models primarily captured local phonological and syntactic patterns within their limited context windows, while recurrent neural networks extended the reach to capture some longer-range syntactic dependencies and semantic coherence. Modern Transformer-based models have demonstrated remarkable ability to capture all of these linguistic levels to varying degrees, including pragmatic phenomena like tracking entities across long documents and maintaining consistent narrative voice throughout extended generations. The question of how well neural language models truly understand each of these linguistic levels—versus merely mimicking surface patterns without genuine comprehension—remains an active area of research and ongoing scholarly debate in the computational linguistics community.

These structural levels form a hierarchy of constraints that language models must learn to capture, with each level operating somewhat independently while also interacting with the others. At the phonological level, the sound patterns of a language constrain which letter sequences are plausible: English permits “str” at the beginning of words but not “tsr,” and these phonotactic constraints help predict spelling even for novel words that the model has never encountered during training. Morphological structure governs word formation through prefixes, suffixes, and root combinations, so knowing that “un-” typically attaches to adjectives helps predict that “unhappy” is more likely than “unsleep.” Syntactic structure imposes grammatical constraints on word order and phrase structure, determining that determiners precede nouns in English and that subjects typically precede verbs in declarative sentences. Semantic constraints require that word sequences make sense: “colorless green ideas sleep furiously” is syntactically valid but semantically anomalous, and language models must learn to penalize such combinations that violate real-world plausibility. Finally, pragmatic structure encompasses discourse-level phenomena including pronoun resolution, topic continuity, and speaker intent, requiring models to track entities and relationships across sentence boundaries. Figure 1.18 illustrates how each of these

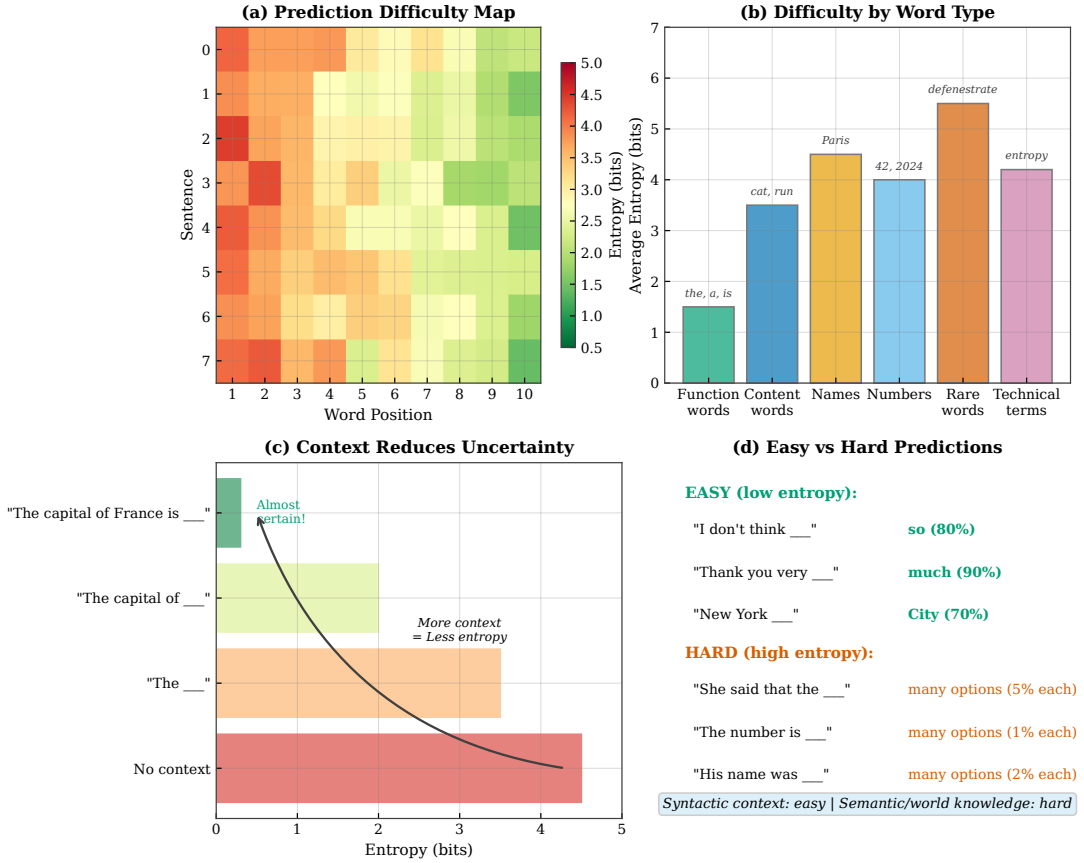

Figure 1.19: Variation in prediction difficulty. Panel (a) shows surprisal distribution across word positions. Panel (b) displays how context length affects difficulty. Panel (c) illustrates easy vs. hard prediction cases. Panel (d) demonstrates the relationship between difficulty and information content.

levels contributes different constraints on prediction.

1.6 Why Some Words Are Harder to Predict

Prediction difficulty varies dramatically across positions in text, ranging from nearly deterministic cases where only one word is plausible to highly ambiguous positions where dozens of words might reasonably follow. Understanding this variation is crucial for model evaluation and development, as it reveals both the strengths and limitations of different modeling approaches and highlights the linguistic phenomena that remain challenging even for state-of-the-art systems. Some predictions are trivially easy: after the phrase “the United States of,” the word “America” follows with near certainty, requiring only basic knowledge of common proper noun phrases. Other predictions approach randomness: the first word of a news article could be almost anything, from “Scientists” to “The” to “Breaking,” depending on topic and style. Between these extremes lies a rich spectrum of prediction difficulty that depends on syntactic constraints, semantic coherence, world knowledge, and discourse context. A key insight from Shannon’s information theory is that this variation in predictability directly corresponds to variation in information content: easy-to-predict words carry little information because they tell us nothing unexpected, while hard-to-predict words carry substantial information precisely because they surprise us. This relationship between predictability and information makes the study of prediction difficulty central to understanding both language models and language itself.

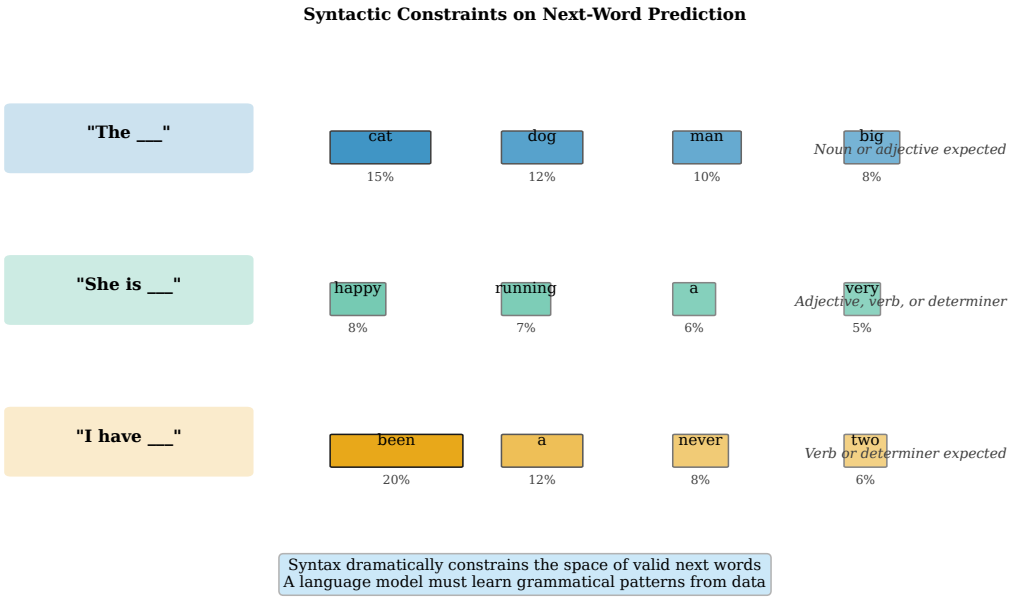

Figure 1.20: Syntactic constraints on next-word prediction. Grammar imposes strict ordering requirements that dramatically reduce the space of plausible continuations at each position.

1.6.1 Context Length and Information

Words at the beginning of a sentence are harder to predict than words later in the sentence, because less context is available to constrain the possibilities—the very first word of a document could be almost anything, while the tenth word is heavily constrained by the nine words preceding it. This relationship between context length and prediction difficulty follows directly from information theory: longer contexts reduce conditional entropy by providing more evidence to discriminate between possible continuations. Similarly, function words (“the,” “of,” “is”) are generally more predictable than content words (“quantum,” “fascinating,” “Constantinople”), because function words serve grammatical roles that are strongly determined by syntactic structure, while content words carry the specific semantic payload that varies widely depending on topic and domain. This distinction between predictable function words and unpredictable content words has important implications for evaluation: most of a language model’s perplexity comes from predicting content words, even though function words account for a larger fraction of tokens in running text. A model that perfectly predicts all function words but struggles with content words would still have high perplexity, because the content words carry the bulk of the information and thus dominate the cross-entropy calculation.

Figure 1.19 shows how surprisal varies across a sentence, revealing which positions carry more information.

1.6.2 Prediction Examples

To make these abstract notions of prediction difficulty concrete, we examine specific examples that span the full range from highly predictable to genuinely uncertain. Figure 1.20 provides illustrative cases organized by difficulty level, demonstrating how different types of linguistic knowledge contribute to prediction accuracy. The highly predictable cases typically involve fixed expressions, collocations, or contexts where syntax and semantics combine to leave only one plausible continuation—these are the positions where language models achieve their lowest perplexity and where even simple n-gram models perform reasonably well. The moderately predictable cases require more sophisticated reasoning about semantic coherence and world knowledge, often admitting several plausible continuations that a good model should assign comparable probabilities. The most difficult cases involve genuine ambiguity where multiple completions are equally valid, positions at the start of new topics where context provides little constraint, or situations requiring specialized domain knowledge that may not appear in general training corpora. Understanding these difficulty levels helps us interpret model

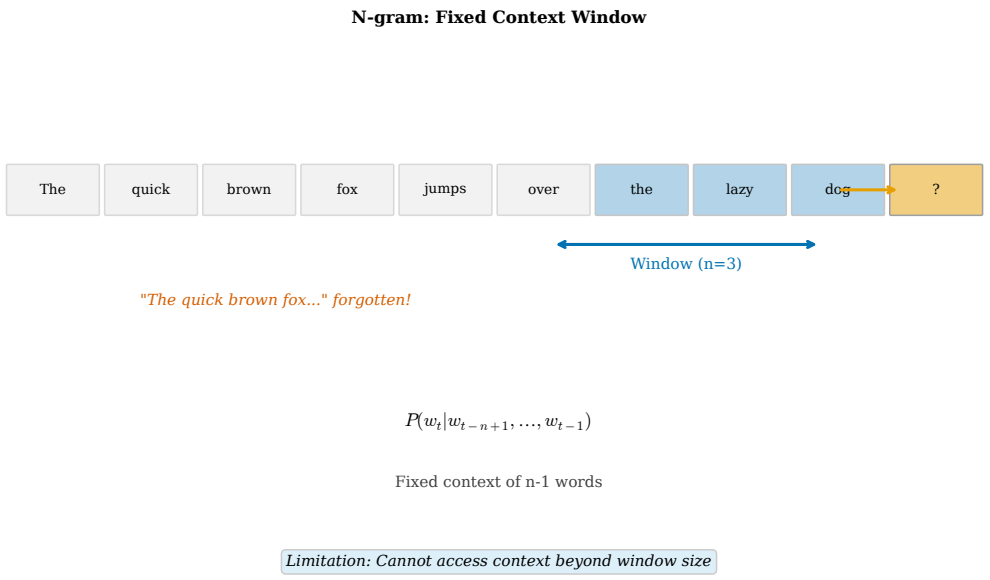

Figure 1.21: N-gram context window architecture. N-gram models use a fixed window of the previous $n-1$ words, making them fast and simple but unable to capture dependencies beyond that window.

performance: a model should not be penalized heavily for failing to predict the specific word chosen in a genuinely ambiguous context, but should be penalized for missing obvious predictions that human readers would find trivially predictable.

Example 1.3 (Easy Prediction)**.** *"She closed her _____" → "eyes" (high probability)*

Example 1.4 (Medium Prediction)**.** *"The meeting was scheduled for _____" → "Monday"/"Tuesday"/"next week"*

Example 1.5 (Hard Prediction)**.** *"The researcher's findings about _____" → Requires domain knowledge*

1.6.3 Context Window Comparison

The amount and type of context a model can access fundamentally determines its prediction capabilities, and different model architectures handle context in fundamentally different ways that reflect both computational constraints and inductive biases about which patterns matter for prediction. N-gram models use a fixed window of the previous $n-1$ words, making them fast and simple but unable to capture dependencies that span beyond that window—the choice of context in "The professor who taught the course that the students loved was" cannot help predict the final verb form because it lies outside any practical n-gram window. Recurrent neural networks theoretically access unbounded history through their hidden state, but in practice suffer from vanishing gradients that make distant context difficult to use effectively, despite architectural innovations like LSTM gates designed to ameliorate this problem. Transformer models use self-attention to directly access any position within a fixed context window, enabling effective use of long-range context up to the window limit but requiring computational resources that scale quadratically with sequence length. Understanding these architectural trade-offs in context handling is essential for choosing appropriate models for different applications and for appreciating the advances discussed in subsequent chapters:

Figure 1.21 compares how different architectures handle context, foreshadowing the detailed treatment in later chapters.

1.7 Training and Evaluation Paradigms

Before diving into specific model architectures, we establish the basic paradigm for training and evaluating language models that will recur throughout this book. Training a language model means adjusting its parameters to maximize the probability it assigns to observed text, while evaluation measures how well the trained model predicts held-out text that was not seen during training. The separation between training and evaluation data is crucial: we want to measure how well the model generalizes to new text, not merely how well it memorizes the training corpus, because real-world deployment will always involve text the model has never seen before. Good generalization is the hallmark of a successful language model, distinguishing genuine learning from mere memorization. This section introduces the mathematical framework of maximum likelihood estimation, the smoothing techniques that address data sparsity, and the metrics we use to quantify prediction quality. These foundations apply regardless of whether the model is a simple n-gram counter or a billion-parameter neural network. The mathematical objective is the same—maximize the probability of observed sequences—but the methods for achieving this objective vary dramatically across the different modeling approaches we will study in subsequent chapters.

1.7.1 Maximum Likelihood Estimation

The standard approach to training language models is **maximum likelihood estimation (MLE)**. Given a training corpus $\mathscr{D} = \{w_1, \ldots, w_T\}$, we maximize:

$$\mathscr{L}(\theta) = \sum_{t=1}^{T} \log p_\theta(w_t | w_1, \ldots, w_{t-1}) \tag{1.16}$$

Equivalently, we minimize cross-entropy loss (negative log-likelihood). Figure 1.22 visualizes the optimization landscape and the relationship between likelihood and our evaluation metric.

1.7.2 Smoothing Techniques

For count-based models, **smoothing** addresses the zero-probability problem for unseen n-grams:

The simplest approach, known as add-one or Laplace smoothing, adds a count of one to every possible n-gram before computing probabilities, ensuring that no combination receives zero probability. While this ensures no n-gram has zero probability, it tends to redistribute too much mass to unseen events, often degrading perplexity substantially on held-out test data. A refinement called add-k smoothing uses a fractional count less than one, typically tuned on held-out data to find the optimal balance between coverage of unseen events and fidelity to observed frequencies in the training corpus. Good-Turing smoothing takes a more principled approach by examining the "frequency of frequencies"—how many n-grams appear exactly once, exactly twice, and so on—and using this information to estimate the total probability mass that should be allocated to unseen events. The most sophisticated technique, Kneser-Ney smoothing, combines absolute discounting with a clever modification to the backoff distribution: instead of using the raw frequency of a word in lower-order contexts, it uses the number of distinct contexts in which the word appears, which better captures a word's versatility as a continuation rather than simply its overall frequency. Figure 1.23 compares these approaches, which we explore in depth in Chapter 2.

1.7.3 Evaluation Metrics

Figure **??** summarizes the key evaluation metrics used to assess language model quality across different dimensions and use cases. Perplexity serves as the primary intrinsic metric, directly measuring the model's ability to predict held-out text without reference to any downstream task or application-specific considerations. Lower perplexity indicates better predictions, with state-of-the-art models achieving perplexities below 10 on standard benchmarks such as WikiText-103. For comparing models with different tokenization schemes, bits per character provides a normalization that accounts for vocabulary differences: a character-level model and a subword

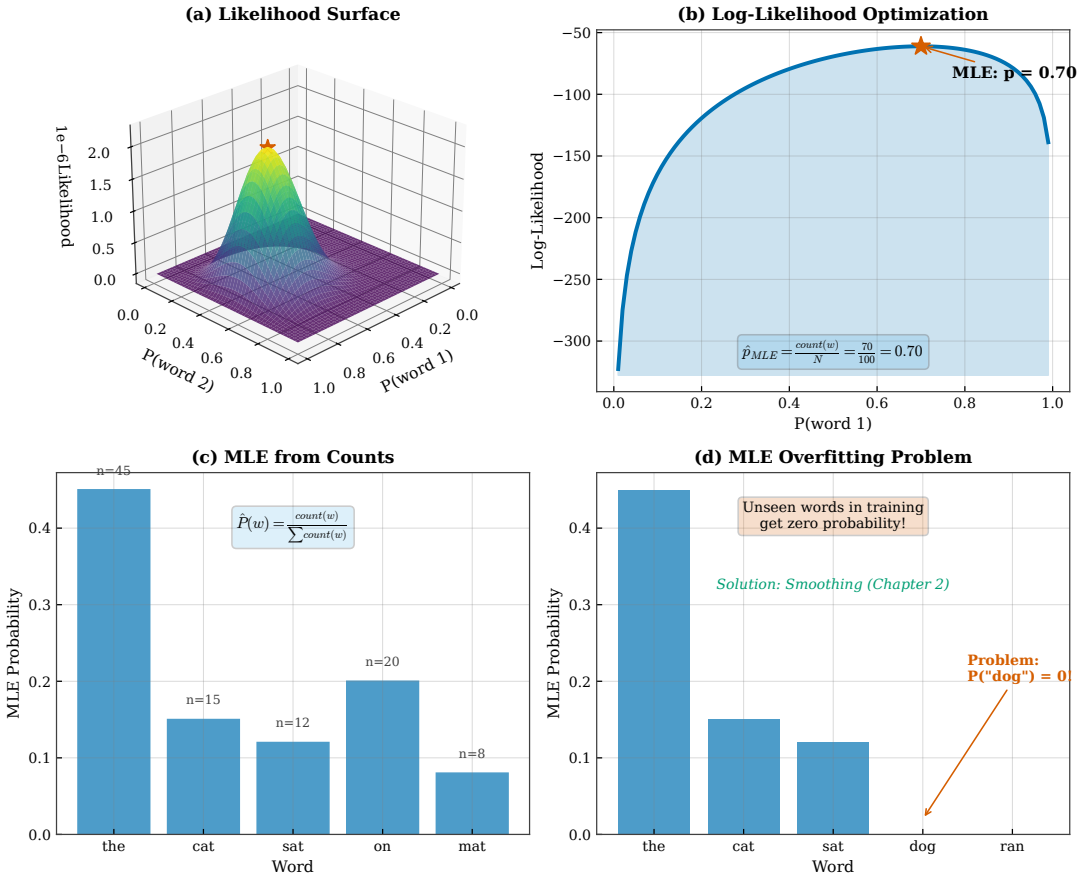

Figure 1.22: Maximum likelihood estimation for language models. Panel (a) shows a 3D likelihood surface over parameter space. Panel (b) illustrates gradient ascent optimization. Panel (c) displays the relationship between likelihood and perplexity. Panel (d) demonstrates overfitting on small datasets.

model can be directly compared through their BPC scores, even though their perplexities would be incomparable due to different prediction granularities. Beyond these intrinsic metrics, researchers increasingly evaluate language models through downstream task performance, measuring accuracy on question answering, BLEU scores on translation, or success rates on code generation benchmarks. These extrinsic evaluations capture whether the model's predictions translate into useful capabilities, though they introduce additional confounds from task-specific architectures and decoding strategies. The relationship between intrinsic perplexity and extrinsic task performance is generally positive but imperfect, motivating the use of multiple complementary evaluation approaches in practice.

1.7.4 Training Data Scale

Modern language models require massive training corpora:

Figure 1.25 shows the exponential growth in training data over the past three decades, a trajectory that has been essential to improving language model quality. In the 1990s, researchers worked with corpora containing millions of words, typically drawn from newswire text or carefully curated collections. By the 2000s, advances in storage and processing enabled training on billions of words from web crawls, substantially reducing perplexity and enabling more robust generalization. The 2020s have witnessed an explosion to trillions of tokens, with models like GPT-3 training on hundreds of billions of tokens and subsequent models pushing into the multi-trillion token regime. This exponential scaling has been accompanied by increasing sophistication in data curation: modern training corpora blend multiple sources including books, academic papers, code repositories, and filtered web text, with careful attention to deduplication, quality filtering, and domain balance. The scaling laws discussed in Chapter 10 formalize the relationship between data quantity and model performance,

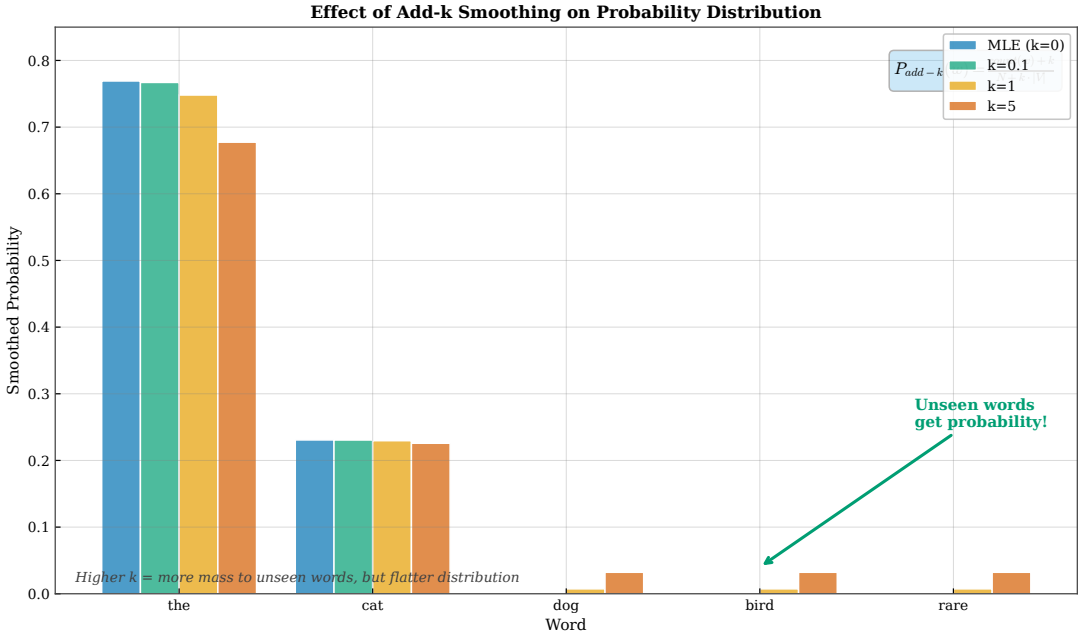

Figure 1.23: Effect of add-k smoothing on probability distributions. Different values of k redistribute probability mass from seen words to unseen words, addressing the zero-probability problem.

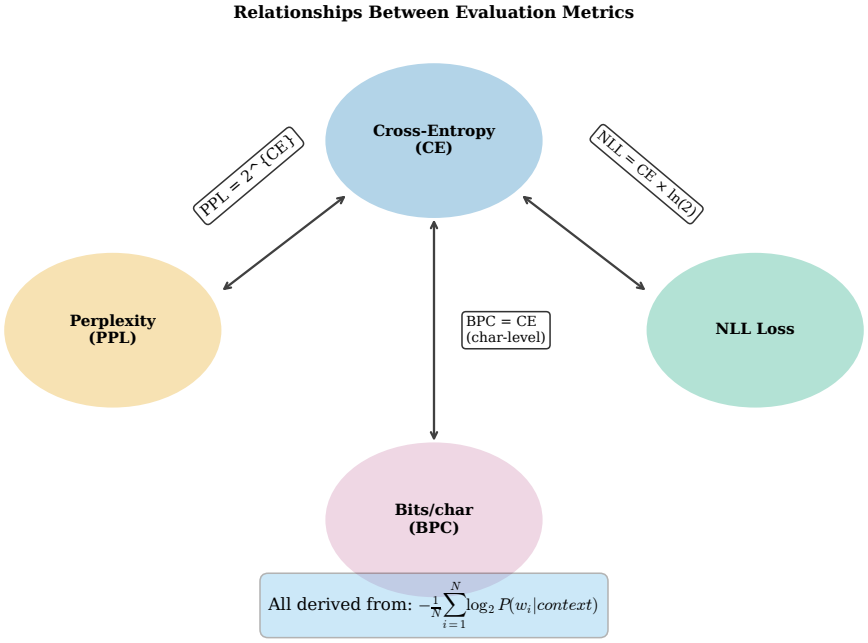

Figure 1.24: Relationships between evaluation metrics. Perplexity, cross-entropy, NLL loss, and bits-per-character are all mathematically related through the fundamental log-probability calculation.

revealing that larger models require proportionally more data to reach their optimal perplexity. This interplay between model capacity and data scale has become one of the central themes of modern large language model development.

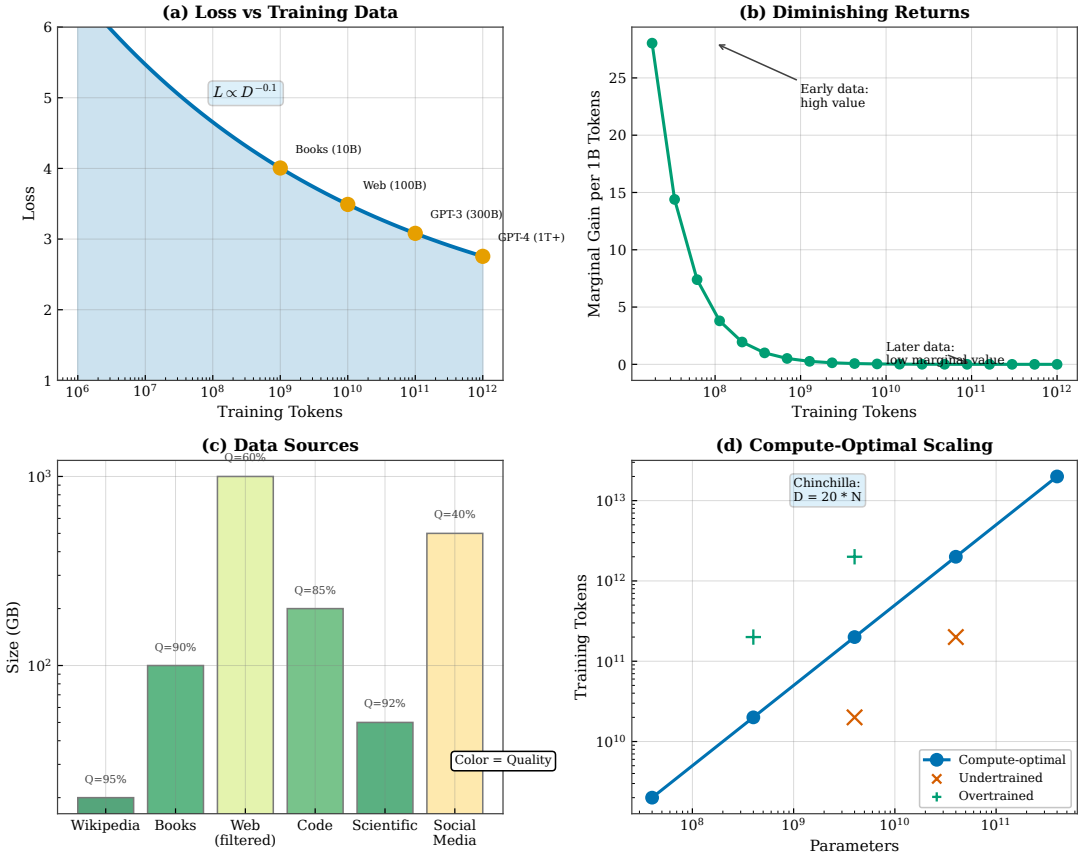

Figure 1.25: Training data scale in language modeling. Panel (a) shows the exponential growth in training data over time. Panel (b) displays the relationship between data size and model performance. Panel (c) illustrates data composition (books, web, code). Panel (d) demonstrates diminishing returns at scale.

1.8 Information Flow in Language Models

Understanding how information flows from input to prediction helps build intuition for the architectures we will study in subsequent chapters and reveals the common structure underlying seemingly different approaches to language modeling. Despite their apparent diversity, all language models share a fundamental processing pipeline that transforms raw text into probability distributions, and recognizing this shared structure makes it easier to understand each architecture's innovations and limitations. Every language model, regardless of its specific architecture, must solve the same fundamental problems: converting discrete text into a form that can be processed mathematically through tokenization and embedding, encoding contextual information into a representation that captures relevant patterns through some form of sequence processing, and transforming that representation back into a probability distribution over possible next words through a projection and normalization step. The differences between architectures lie in how they implement these stages, particularly the crucial encoding stage where context is compressed and transformed: n-grams use simple lookup tables, RNNs accumulate information through recurrent hidden states, and Transformers use attention mechanisms to selectively combine information across positions. By examining this general pipeline, we can appreciate both what different architectures share and where they diverge in their approaches to representation and computation, setting the stage for the detailed treatment of each approach in subsequent chapters.

Figure 1.26 illustrates the general pipeline that transforms raw text into probability predictions. The first stage, tokenization, converts the input text into a sequence of discrete tokens drawn from a fixed vocabulary, handling the boundary between continuous character streams and the discrete symbols that neural networks process. The second stage, embedding, maps each discrete token to a continuous vector in a high-dimensional space, enabling the model to learn rich representations of word meaning and capture similarities between related

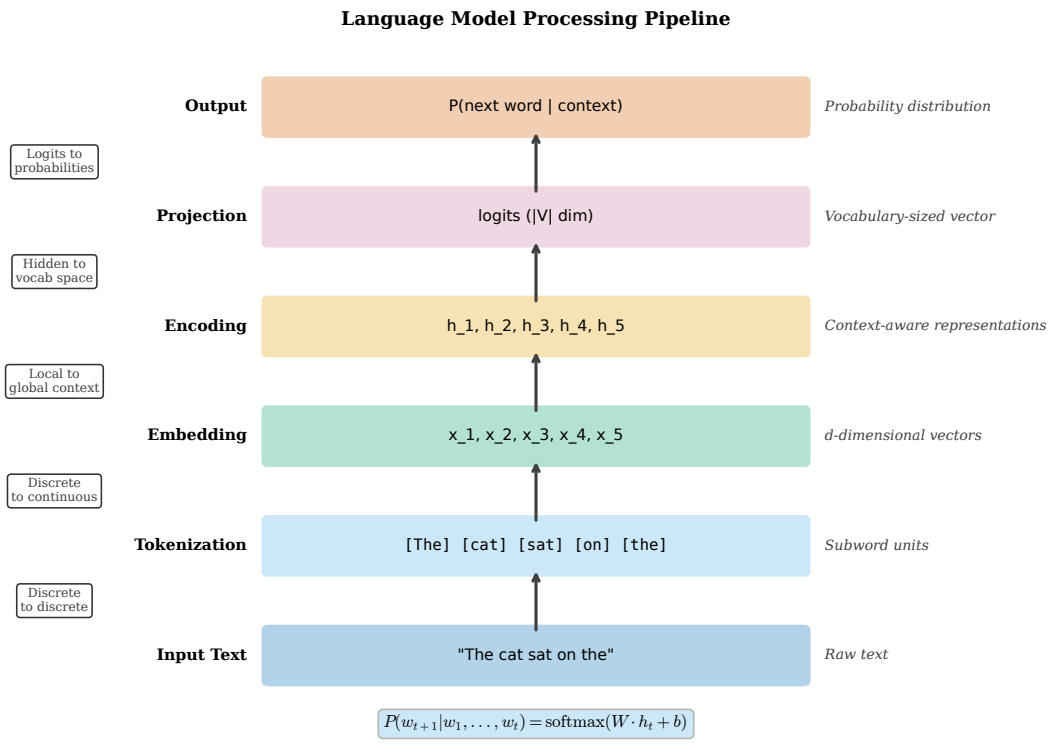

Figure 1.26: Language model processing pipeline. Every language model transforms raw text through tokenization, embedding, encoding, projection, and softmax normalization to produce a probability distribution over possible next words.

tokens. The third stage, encoding, constitutes the heart of the language model: it processes the sequence of embedding vectors and produces a contextual representation that summarizes the relevant information from the preceding context. This encoding stage is where the major architectural innovations occur—n-grams use lookup tables, RNNs use recurrent computation, and Transformers use self-attention. The final stage, projection, maps the contextual representation back to vocabulary space, producing a probability distribution over all possible next tokens through a linear transformation followed by softmax normalization. This pipeline, in various forms, appears in every modern language model, and the key differences between architectures lie primarily in the encoding stage and how context is represented and processed.

1.9 Comparing Language Model Approaches

Before diving into specific architectures in subsequent chapters, we preview the trade-offs between different approaches to give readers a roadmap of what lies ahead and to motivate the architectural choices we will examine in detail throughout this book. Each approach to language modeling makes different trade-offs along multiple dimensions including prediction accuracy as measured by perplexity, computational efficiency during both training and inference, memory requirements that determine what hardware can run the model, interpretability of predictions and internal representations, and ability to capture long-range dependencies that span many words or even sentences. No single architecture dominates across all these dimensions, which explains why multiple approaches remain relevant for different applications and deployment contexts. A simple n-gram model running on a smartphone can provide instant autocomplete suggestions with minimal battery drain, while the same device could never run a billion-parameter Transformer that achieves state-of-the-art perplexity. Conversely, applications requiring nuanced understanding of long documents must sacrifice the speed and simplicity of n-grams for the sophisticated context modeling that only neural architectures provide. Understanding these trade-offs helps practitioners choose appropriate models for their specific constraints and

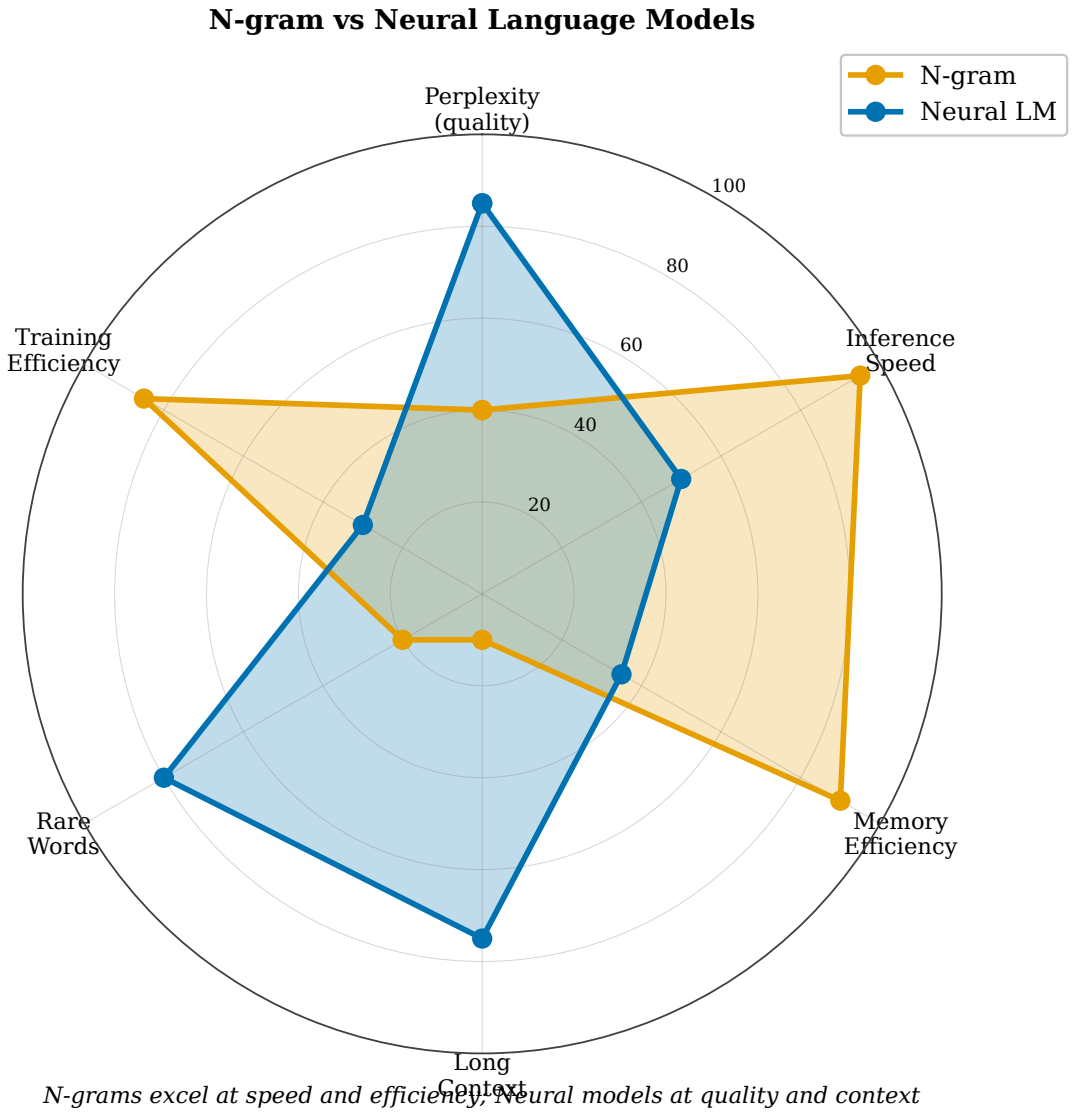

Figure 1.27: Radar chart comparing n-gram and neural language models across multiple dimensions. N-grams excel at speed and memory efficiency while neural models achieve better perplexity and long-range context.

helps researchers identify opportunities for improvement by recognizing which dimensions current approaches sacrifice and whether architectural innovations might recover that lost ground.

Figure 1.27 compares four major approaches across multiple dimensions including perplexity, inference speed, memory requirements, and interpretability. N-gram models offer the fastest inference and most interpretable predictions, since probabilities derive directly from observable count statistics, but their fixed and limited context windows prevent them from capturing long-range dependencies and lead to higher perplexity on complex text. Recurrent neural networks and LSTMs address the context limitation by maintaining hidden states that theoretically encode unbounded history, achieving substantially lower perplexity than n-grams, but their sequential processing prevents parallelization and limits training efficiency. The Transformer architecture revolutionized the field by enabling full parallelization through self-attention, achieving even lower perplexity and faster training, though the quadratic memory scaling with sequence length poses challenges for very long contexts. Large language models built on Transformer foundations exhibit emergent abilities—including in-context learning, chain-of-thought reasoning, and instruction following—that smaller models lack entirely, but these capabilities require enormous computational resources for both training and inference. No single model is best for all tasks, and the choice depends on computational budget, latency requirements, task complexity, and whether emergent capabilities are necessary for the application at hand.

1.10 Prediction Spotlight

Throughout this book, we track prediction quality using a running example that demonstrates how different modeling approaches handle the same prediction task, allowing readers to build intuition for the progressive improvements in language modeling that each chapter introduces. This running example serves as a concrete touchstone that grounds abstract architectural discussions in observable prediction behavior: rather than simply asserting that Transformers outperform n-grams, we show how the predictions differ on identical input. The example we use involves predicting the continuation of a sentence about research findings, a context that requires both syntactic knowledge about English sentence structure and semantic knowledge about how researchers typically describe their results. By returning to this same example across multiple chapters, readers can directly compare how a bigram model's prediction based on only the previous word differs from an LSTM's prediction that incorporates the full sentence history, which in turn differs from a Transformer's attention-weighted integration of all context positions. This comparative approach makes the benefits of architectural advances tangible and memorable, transforming abstract perplexity improvements into concrete differences in predicted word distributions. Consider the prompt:

"The researcher analyzed the data and found that the results _____"

Model	Prediction	Quality
Uniform (50K vocab)	Random word	Terrible
Unigram	"the" (most common)	Poor
Human (Shannon-level)	"were" or "showed"	Good
Modern LLM	"were consistent with..."	Excellent

Table 1.1: Prediction quality across model types. This chapter establishes the baseline; subsequent chapters show progressive improvement.

The journey from uniform random to sophisticated context-aware prediction is the story of this book.

1.11 Context Representation: The Central Challenge

How This Chapter Represents Context
The fundamental question in language modeling is: How do we represent the context $w_1, \ldots, w_{t-1}$ to predict w_t?

- **Context representation**: This chapter introduces context as the key variable, without specifying how to encode it
- **Key insight**: Different contexts require different predictions—"the cat sat on the" vs. "the capital of France is"
- **Limitation**: We have not yet specified how to encode context computationally

Different approaches make different trade-offs:

- **Fixed window**: Use only the last $n-1$ words (n-grams, Chapter 2)
- **Unbounded history**: Theoretical ideal, but computationally challenging
- **Compressed representation**: Encode context in a fixed-size vector (Chapters 5–6)

This chapter has introduced the fundamental problem of next-word prediction and the information-theoretic framework we use to evaluate solutions; subsequent chapters develop increasingly sophisticated approaches to actually solving this problem through better context representation. The evolution of context representation—from the fixed windows of n-gram models through the recurrent hidden states of LSTMs to the dynamic attention mechanisms of Transformers—constitutes the central technical arc of this book, with each advance enabling qualitatively better predictions by capturing more of the structure inherent in natural language. Understanding context representation as the central challenge unifies the diverse techniques we will study: every architectural innovation can be understood as a new way to encode what came before in a form that supports accurate prediction of what comes next. This perspective helps us appreciate both why older methods work as well as they do and why newer methods improve upon them. The n-gram's fixed window is a crude but computationally efficient context representation; the RNN's hidden state is a more flexible but sequentially constrained representation; the Transformer's attention-weighted combination is yet more flexible and parallelizable. Each step in this progression expands what information about the context can flow into the prediction, bringing us closer to the theoretical ideal of conditioning on all available information.

1.12 Roadmap of This Book

Figure 1.28 shows the structure of this book and the dependencies between chapters. The book begins with foundational material in Chapters 2 and 3, covering n-gram language models and tokenization respectively—these topics establish the vocabulary and baselines against which all subsequent methods are compared. Chapters 4 through 6 introduce the neural revolution, progressing from static word embeddings through recurrent neural networks to the Transformer architecture that dominates modern language modeling. Chapter 7 addresses the decoding problem: given a trained language model, how do we actually generate text? This chapter bridges the gap between having a probability distribution and producing useful output. Chapter 8 examines training at scale, covering optimization algorithms, distributed training, and data considerations. Chapters 9 and 10 explore the frontier of large language models and the scaling laws that govern their behavior, including emergent capabilities and the compute-optimal training debate. Chapter 11 covers post-training alignment: instruction tuning, RLHF, and DPO methods that transform raw language models into helpful assistants. Chapter 12 addresses efficiency through quantization, pruning, and distillation techniques that make deployment practical. Finally, Chapter 13 surveys applications from translation to code generation.

The book accommodates different reading goals through multiple paths. Researchers seeking comprehensive understanding should read sequentially from Chapters 1 through 13, building intuition at each stage. Practitioners focused on deploying modern systems can follow a condensed path through Chapters 1, 3, 6, 7, 9, 12, and 13, covering the essential knowledge for working with contemporary language models. Readers seeking a quick conceptual overview should focus on Chapters 1, 6, 9, and 13 to understand the core ideas and current state of the field. Those with a theoretical bent should prioritize Chapters 1, 2, 4, 6, 8, 10, and 11, which contain the deepest mathematical content and formal analysis.

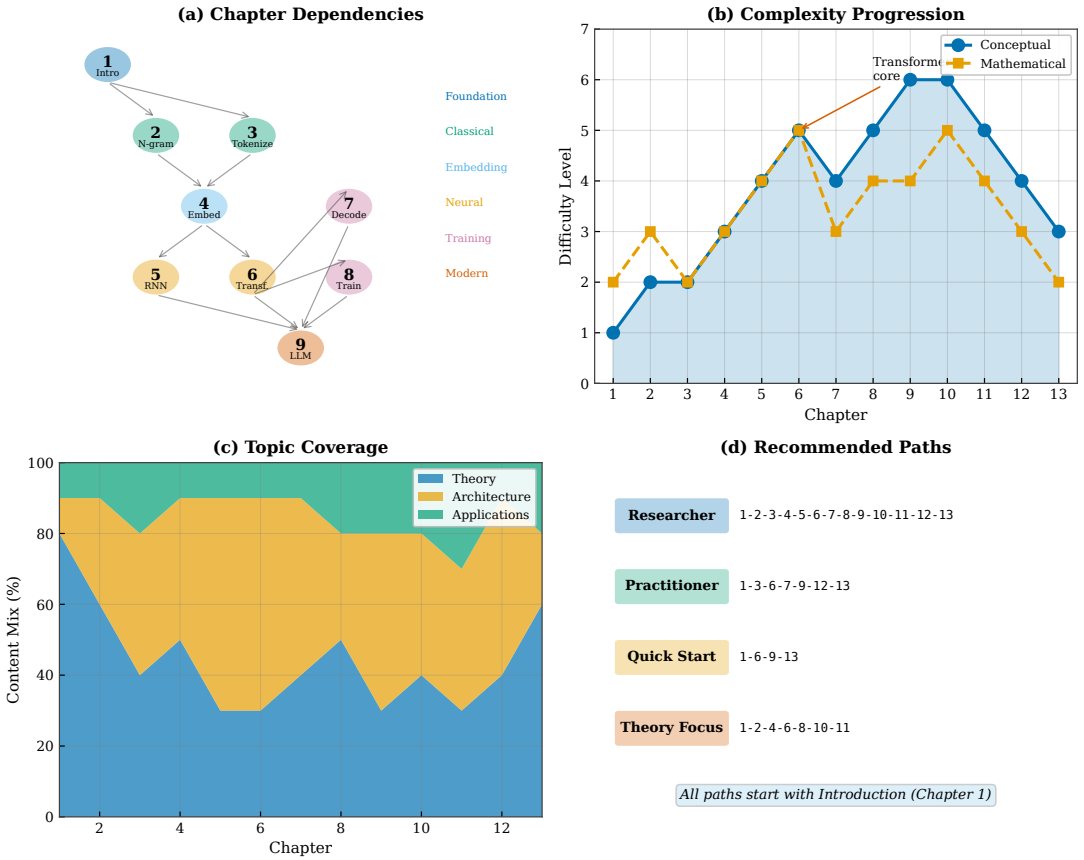

Figure 1.28: Book structure and reading paths. Panel (a) shows chapter dependencies as a directed graph. Panel (b) displays complexity progression across chapters. Panel (c) illustrates topic coverage (theory, architecture, applications). Panel (d) suggests reading paths for different goals.

1.13 Summary

We can now predict better because:

- We understand the formal objective: $P(w_t \mid w_1, \ldots, w_{t-1})$
- Information theory provides our theoretical foundation and evaluation metric (perplexity)
- The historical arc from Shannon to modern LLMs shows progressive improvements in context representation
- Linguistic regularities (Zipf's law) inform our modeling choices

Next: Chapter 2 develops our first concrete solution—n-gram language models that use fixed-length context windows.

Exercises

1. Prove that entropy $H(X)$ is maximized when X is uniformly distributed over n values.

2. Given a corpus where word w appears with frequency f_w, derive the maximum likelihood estimate for $P(w)$ and show it equals $f_w / \sum_v f_v$.

3. Show that perplexity of a uniform distribution over V words equals V.

4. Calculate the entropy of a bigram distribution where $P(\text{cat} \mid \text{the}) = 0.6$, $P(\text{dog} \mid \text{the}) = 0.3$, and $P(\text{bird} \mid \text{the}) = 0.1$.

5. Prove that cross-entropy $H(p,q) \geq H(p)$ with equality if and only if $q = p$. (Hint: Use Jensen's inequality or the non-negativity of KL divergence.)

6. * Derive the relationship between cross-entropy loss and KL divergence for language models. Show that minimizing cross-entropy is equivalent to minimizing KL divergence to the empirical distribution.

7. Show that for a language model, perplexity equals the geometric mean of the inverse probabilities: $\text{PPL} = \left(\prod_{t=1}^{T} \frac{1}{P(w_t \mid w_{<t})}\right)^{1/T}$.

8. **Research Question**: Read Shannon's 1951 paper on predicting English text. How do his human prediction experiments compare to modern language model perplexities? What does this suggest about the lower bound on English entropy?

9. **Research Question**: Why might Zipf's law emerge in natural language? Discuss at least two proposed explanations from the literature (e.g., least effort principle, preferential attachment).

10. **Research Question**: Modern LLMs achieve perplexities below 10 on common benchmarks like WikiText-103. What does this suggest about the entropy of English text? Is there a theoretical limit?

11. **Research Question**: Compare the chain rule decomposition (left-to-right) with other possible decompositions. What are the implications for bidirectional models like BERT?

12. * Prove that the entropy rate of a stationary stochastic process $\lim_{n\to\infty} \frac{1}{n} H(w_1, \ldots, w_n)$ exists and equals $\lim_{n\to\infty} H(w_n | w_1, \ldots, w_{n-1})$.

13. * Design an experiment to empirically estimate the entropy of English text using human subjects, following Shannon's methodology. What controls would you need? How would you account for individual differences in language knowledge?

Bibliography

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2015.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, et al. Language models are few-shot learners. *NeurIPS*, 33:1877–1901, 2020.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint*, 2025.

Irving J Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3-4):237–264, 1953.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Frederick Jelinek. *Self-organized language modeling for speech recognition*. Morgan Kaufmann, 1990.

Dan Jurafsky and James H Martin. *Speech and Language Processing*. Prentice Hall, 3rd edition, 2024. Draft available at https://web.stanford.edu/ jurafsky/slp3/.

Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In *ICASSP*, volume 1, pages 181–184, 1995.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Long Ouyang, Jeffrey Wu, Xu Jiang, et al. Training language models to follow instructions with human feedback. *NeurIPS*, 35:27730–27744, 2022.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

Rafael Rafailov, Archit Sharma, Eric Mitchell, et al. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*, 2023.

Claude E Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3): 379–423, 1948.

Claude E Shannon. Prediction and entropy of printed english. *Bell System Technical Journal*, 30(1):50–64, 1951.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 30, 2017.

George Kingsley Zipf. Human behavior and the principle of least effort. *Addison-Wesley Press*, 1949.

Index