Predicting the Next Word

From Shannon to ChatGPT

Test Compilation - Chapter 7

Contents

Chapter 7

Decoding Strategies: From Probabilities to Text

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

- Converting probability distributions to actual text sequences
- Balancing quality and diversity in generated output
- Handling the search space explosion of autoregressive decoding
- Adapting generation strategies to different task requirements

7.1 The Decoding Problem

The transformer architecture developed in Chapter **??** produces a probability distribution $P(w)$ over the vocabulary $\mathscr{V}$ at each position, but language models ultimately must generate discrete text sequences that humans can read and understand. This transformation from continuous probability distributions to discrete token sequences is the fundamental decoding problem that defines how models actually produce useful output in practice. The challenge arises because autoregressive language models generate one token at a time in a sequential process: after predicting and selecting token $w[t]$, this choice becomes an immutable part of the context for predicting $w[t+1]$, creating a cascade of dependent decisions where early mistakes propagate through and contaminate the entire remaining sequence. With a vocabulary size of approximately 50,000 tokens and a target sequence length of 100 tokens, the space of possible outputs grows exponentially to 50000^{100} distinct sequences, a number so astronomically large that it exceeds comprehension and makes exhaustive search computationally impossible by any conceivable technology. The decoding strategy we choose determines not only the quality and coherence of individual outputs but also the diversity of generations across multiple samples from the same prompt, influencing whether the model produces creative variety or repetitive sameness.

Consider our running example: when asked to continue "Once upon a time in a distant kingdom, there lived a young," the model computes a conditional probability distribution $P(w\,|\,\text{context})$ over all possible next words given everything that came before. Figure 7.1 shows this distribution, with "prince" having probability 0.15, "girl" at 0.12, "wizard" at 0.09, and so on through thousands of vocabulary items with monotonically decreasing probability that eventually approaches zero for rare or contextually inappropriate tokens. The decoding problem asks: how should we select from this rich probability distribution to produce compelling text? Should we always pick the most probable word using greedy decoding, should we sample according to the exact probabilities through pure stochastic sampling, or should we use some intermediate strategy that carefully balances exploitation of high-probability tokens with exploration of diverse alternatives? Each choice leads to qualitatively different outputs, from safe but potentially boring and repetitive text to creative but potentially incoherent or nonsensical generations, representing a fundamental trade-off in text generation that has no

universally optimal solution.

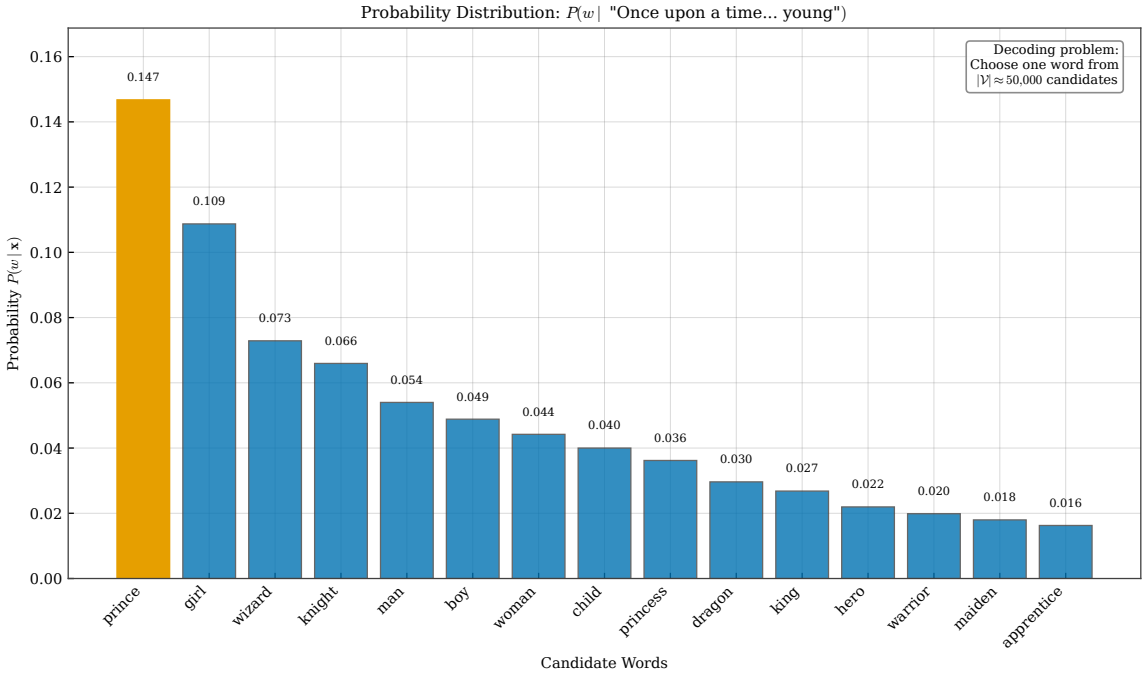

Figure 7.1: Probability distribution over next words after "Once upon a time in a distant kingdom, there lived a young." The model assigns highest probability to "prince" (0.15), followed by "girl" (0.12), "wizard" (0.09), and many other candidates with decreasing probability. Decoding strategies determine how we select from this distribution.

The search space explosion illustrated in Figure 7.2 shows why naive enumeration of all possibilities fails catastrophically. At each decoding step, we face $|\mathcal{V}|$ choices representing every token in our vocabulary, and these choices multiply combinatorially across positions because each decision affects all subsequent distributions. For a vocabulary of 50,000 tokens and sequence length T, the total number of possible sequences is $|\mathcal{V}|^T$, which grows with terrifying speed as sequence length increases. Even for a modest sequence of just 20 tokens, this equals $50000^{20} \approx 10^{94}$, a number that far exceeds not only our computational capabilities but the estimated number of atoms in the observable universe, which is approximately 10^{80}. This combinatorial explosion means we cannot simply enumerate all sequences and pick the best one according to some quality criterion; instead, we must use heuristic strategies that explore only a vanishingly tiny fraction of the search space while hopefully finding outputs that are sufficiently high-quality for our purposes. Different decoding strategies represent fundamentally different trade-offs between computational cost, output quality as measured by coherence and relevance, and output diversity as measured by variation across multiple generations.

Decoding strategies fall into two broad categories: deterministic and stochastic, as shown in Figure 7.3. Deterministic methods like greedy decoding and beam search always produce the same output for a given input, selecting tokens based purely on probability rankings without any randomness whatsoever, making their behavior completely reproducible and predictable. These methods are valuable when consistency matters: running the same prompt twice yields identical results, which simplifies debugging, testing, and deployment of language model systems. Deterministic methods often produce coherent, high-probability sequences that sound grammatical and contextually appropriate, but they can lack diversity and creativity, sometimes falling into repetitive patterns or generating generic, uninspiring text that reads like it was assembled from the most common phrases in the training data. Stochastic methods like temperature sampling, top-k, and nucleus sampling incorporate controlled randomness, sampling tokens according to modified probability distributions that can be tuned to balance exploration and exploitation. These methods can generate diverse, creative outputs by occasionally exploring lower-probability regions of the distribution where surprising but contextually valid tokens reside, but they risk producing incoherent or low-quality text if unlikely tokens are selected at critical junctures where the

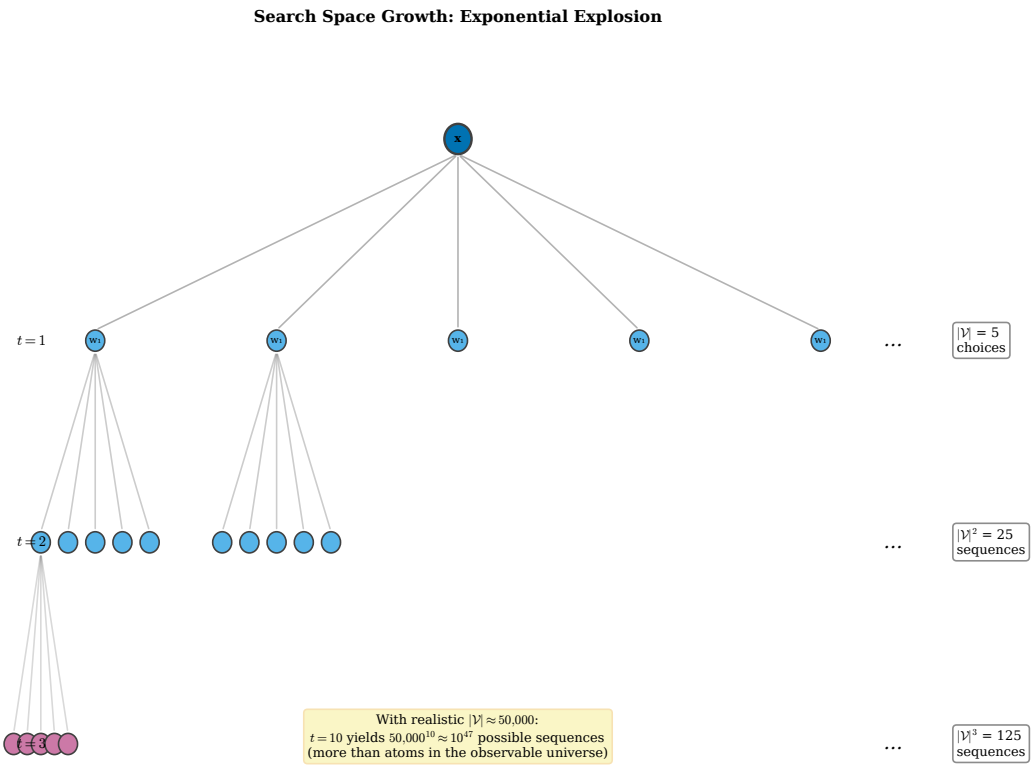

Figure 7.2: The exponential growth of the search space in autoregressive decoding. Each node represents a partial sequence, and each edge represents choosing a next token from the vocabulary $\mathcal{V}$. After just three steps, we have $|\mathcal{V}|^3$ possible sequences. For $|\mathcal{V}| = 50{,}000$ and sequence length 20, the total space contains approximately 10^{94} sequences.

wrong choice derails the entire narrative. Modern practice often combines elements of both approaches, using deterministic pruning to eliminate clearly bad options followed by stochastic sampling among the remaining candidates.

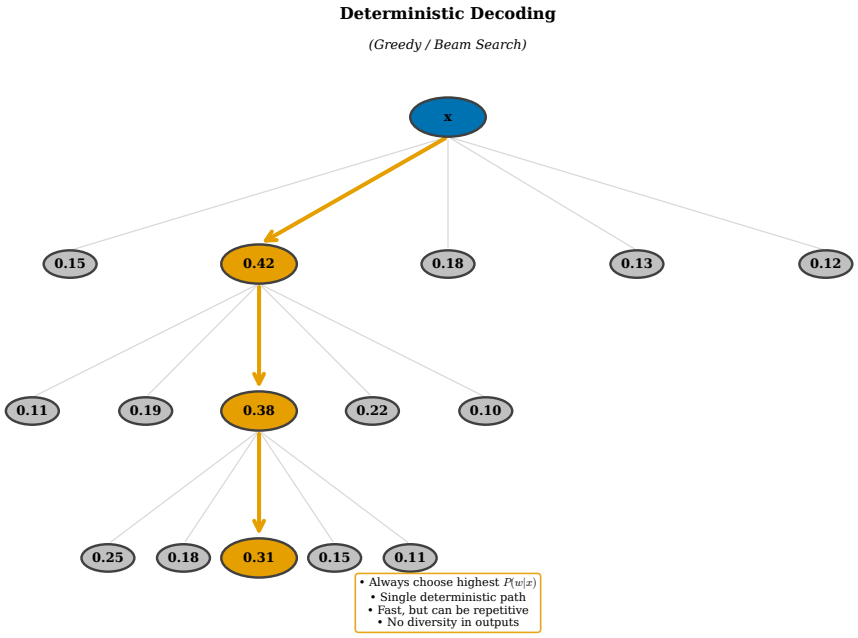

Figure 7.3: Deterministic versus stochastic decoding paradigms. Deterministic methods (left) always follow the same path through the search space, typically selecting highest-probability options. Stochastic methods (right) can explore different paths across multiple runs, potentially discovering more diverse or creative sequences.

7.2 Greedy Decoding and Its Failures

The simplest decoding strategy is greedy decoding, which selects the highest-probability token at each position without considering alternatives: $w[t] = \arg\max_{w \in \mathcal{V}} P(w \mid w[1], \ldots, w[t-1])$. This approach is computationally efficient, requiring only $O(T \cdot |\mathcal{V}|)$ operations to generate a sequence of length T, since we simply take the argmax over vocabulary probabilities at each step without maintaining multiple hypotheses or performing any sampling operations. Greedy decoding produces deterministic output: running it twice on the same prompt yields identical results, which can be advantageous for reproducibility and debugging but limits variety when multiple outputs are desired. The method implicitly assumes that locally optimal choices lead to globally optimal sequences, that picking the best word at each step will produce the best overall text when the sequence is considered as a whole. This assumption, while computationally convenient and intuitively appealing, is fundamentally flawed for language generation, where context dependencies spanning many tokens mean that a slightly less probable word now might enable much more probable and more interesting continuations later, leading to a globally superior sequence despite the locally suboptimal initial choice.

Figure 7.4 illustrates greedy decoding on our running example, showing the decision tree and the single path that greedy decoding selects. Starting with "Once upon a time in a distant kingdom, there lived a young," greedy decoding examines the probability distribution and selects "prince" (probability 0.15 as the highest), then examines the new distribution conditioned on this choice and selects "who" (probability 0.22), then "lived" (probability 0.18), and so on through the generation process. At each step, we commit fully and irrevocably to the highest-probability option without considering alternatives, even when the second-highest option has nearly equal probability and might lead to more interesting continuations. The path through the probability space is entirely deterministic: given the model weights and the input prompt, greedy decoding will always produce exactly the same sequence regardless of when or how many times we run it. The local optimization at each step ignores the global structure of the text and the complex dependencies between distant tokens, potentially missing much better overall sequences that would have required accepting a slightly less probable early choice to unlock superior later options.

The most severe failure mode of greedy decoding is the repetition loop, where the model falls into generat-

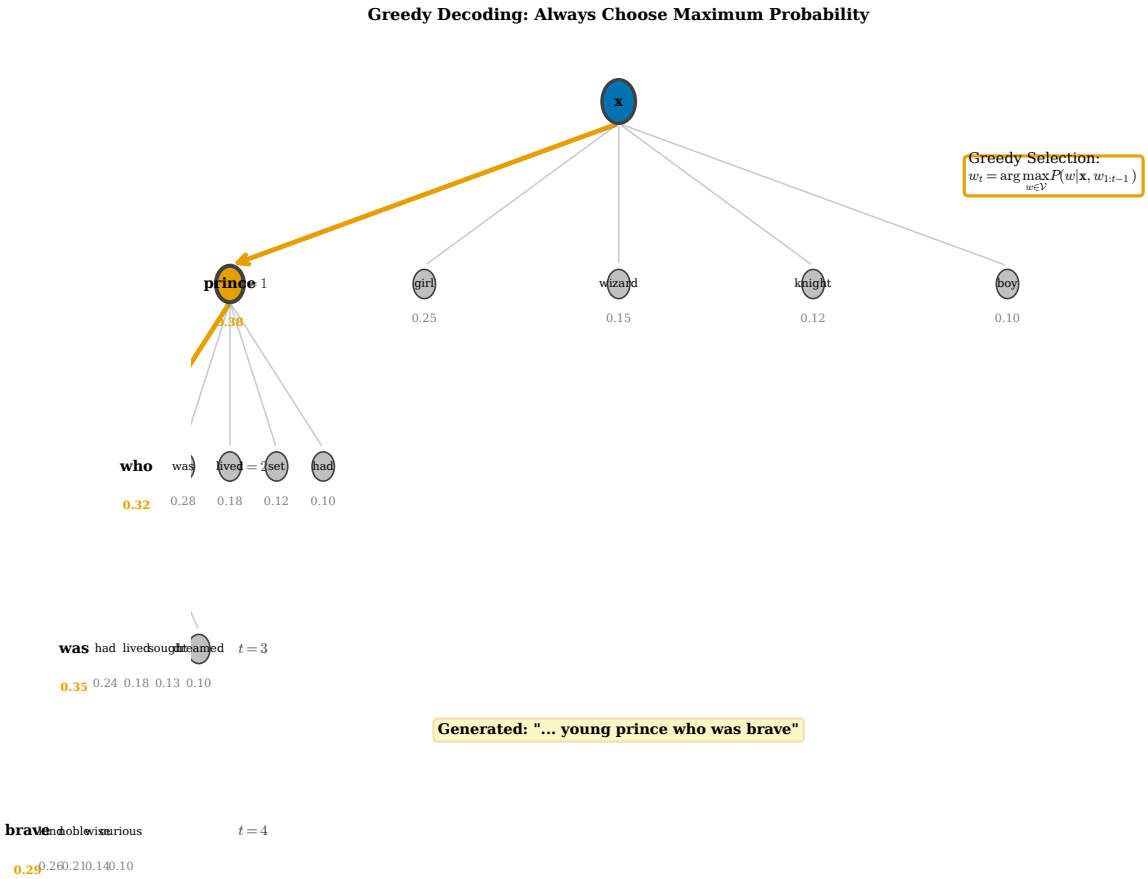

Figure 7.4: Greedy decoding selects the highest-probability token at each step (shown in orange). Alternative tokens with lower probabilities are discarded immediately. The greedy path may not be globally optimal: a different choice at an early step might have enabled higher probabilities later.

ing the same phrase or sentence repeatedly without any mechanism to escape. Figure 7.5 shows this catastrophic failure on our fantasy prompt, where greedy decoding produces: “Once upon a time in a distant kingdom, there lived a young man. The man was a man who was a man who was a man...” This degeneration occurs through a self-reinforcing feedback loop: once the model generates “man,” the phrase “was a man” becomes a high-probability continuation because similar patterns appear frequently in the training data, and after generating “was a man,” the same phrase remains the highest-probability continuation. The greedy strategy follows this local maximum repeatedly, completely unable to escape because doing so would require selecting a lower-probability token. Beyond these catastrophic repetition loops, greedy decoding also produces generic, uninspiring text even when it does not degenerate completely: it gravitates relentlessly toward common, expected continuations rather than surprising or distinctive ones, producing clichéd text filled with overused phrases like “the beautiful princess,” “the brave knight,” and “happily ever after” for creative writing, or safe, bland responses like “I’m doing well, thank you” and “That’s interesting” for dialogue systems. The probability distribution learned by language models reflects the frequency of patterns in their training data, so the highest-probability options are mathematically destined to be the most common and therefore least distinctive tokens, causing greedy decoding to systematically sacrifice diversity and interestingness for a narrow notion of quality that equates probable with good.

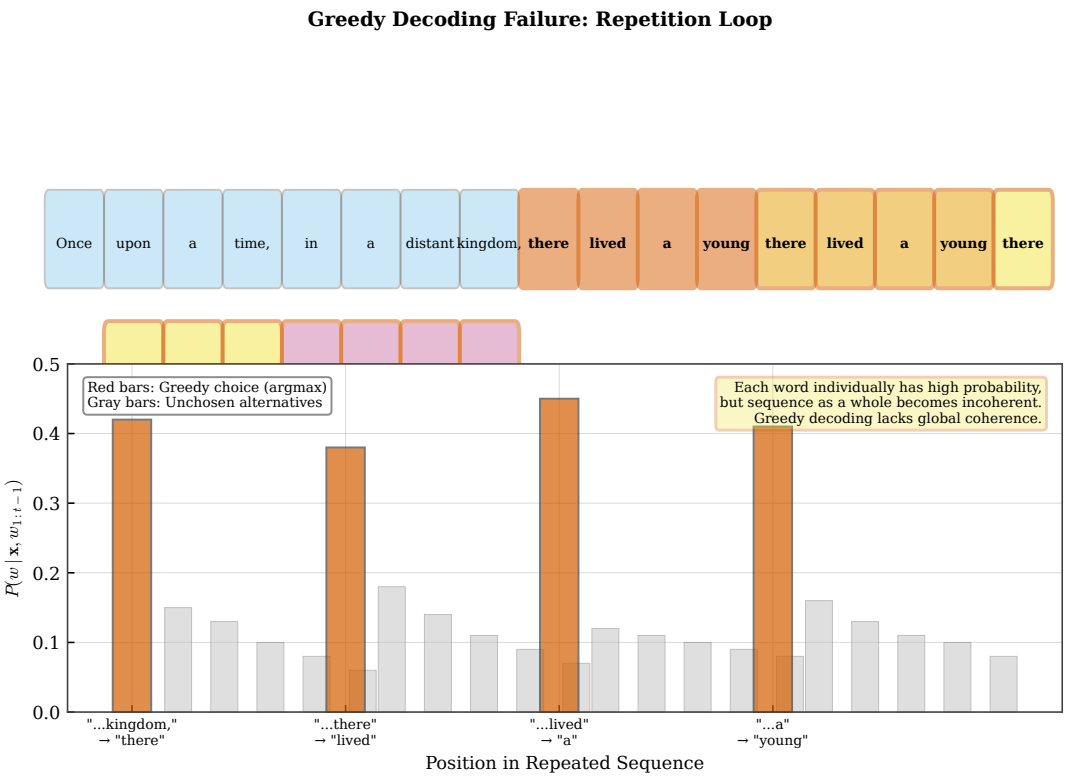

Figure 7.5: Greedy decoding failure: repetition loop. The model falls into repeating "was a man" because this phrase is always the highest-probability continuation of the previous "man." The red highlighting shows where degeneration begins. Greedy decoding cannot escape such loops without external intervention.

7.3 Temperature Scaling

Temperature scaling modifies the probability distribution before sampling by dividing the logits by a temperature parameter τ before applying the softmax function: $P(w) = \frac{\exp(z_w/\tau)}{\sum_{w' \in \mathcal{V}} \exp(z_{w'}/\tau)}$, where z_w is the raw logit (pre-softmax score) for word w. The temperature controls the sharpness or peakedness of the resulting distribution through a simple but powerful mechanism: values less than 1 make the distribution peakier by amplifying differences between logits, while values greater than 1 make it flatter by compressing those differences. At $\tau = 1$, the original distribution is preserved exactly as the model produced it. As $\tau \to 0$, the distribution approaches a one-hot vector concentrated entirely on the highest-probability token, mathematically equivalent to greedy decoding. As $\tau \to \infty$, the distribution approaches uniform over all vocabulary items, giving every token equal probability regardless of the model's preferences. Temperature scaling is motivated by the observation that greedy decoding produces boring text precisely because it always selects the mode: by flattening the distribution through increased temperature, we allow stochastic sampling to explore alternative tokens while still maintaining a preference for more probable options over less probable ones.

Figure 7.6 shows how different temperature values dramatically affect our probability distribution for the next word after "young," illustrating the trade-off between peaked and flat distributions. At $\tau = 0.3$, nearly all probability mass concentrates sharply on "prince," making all other options almost impossible to sample and effectively reducing sampling to greedy selection. At $\tau = 1.0$, the original distribution shows a clear preference for "prince" at 15% but gives meaningful probability to alternatives like "girl" at 12% and "wizard" at 9%, creating genuine randomness while respecting the model's learned preferences. At $\tau = 2.0$, the distribution is much flatter with many tokens having similar probabilities, allowing creative but potentially incoherent choices

because even low-probability tokens now have reasonable sampling chances. The three-dimensional surface in Figure 7.7 visualizes how token probabilities vary jointly with temperature and token rank across the entire distribution: at low temperatures, the surface shows a dramatic sharp peak at the first-ranked token with probabilities dropping precipitously for lower-ranked tokens, while as temperature increases this peak gradually flattens and spreads outward, with probability mass redistributing from the top tokens to the middle and tail, compressing the differences so that the 10th-ranked token becomes nearly as likely as the 2nd-ranked token. This global reshaping explains why temperature simultaneously affects both the expected quality of generated text (by changing which tokens are likely to be sampled) and its diversity (by changing how much variation occurs across multiple samples).

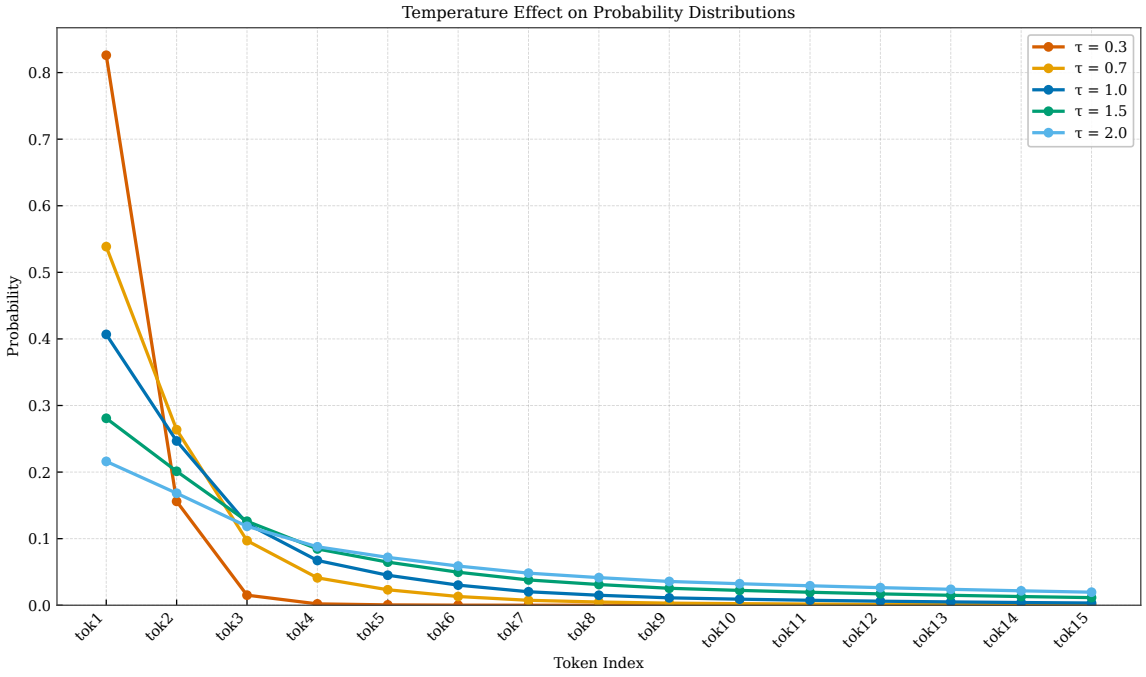

Figure 7.6: Effect of temperature on probability distributions. At $\tau = 0.3$ (red), probability concentrates sharply on the highest-probability token. At $\tau = 1.0$ (blue), the original distribution is preserved. At $\tau = 2.0$ (green), the distribution flattens significantly, giving many tokens similar probability.

Figure 7.8 quantifies the fundamental relationship between temperature and entropy using information-theoretic measures. Entropy, defined as $H(P) = -\sum_{w \in \mathcal{V}} P(w) \log P(w)$, measures the uncertainty or randomness inherent in the distribution—higher entropy means more unpredictability in which token will be sampled. At $\tau \to 0$, entropy approaches zero because all probability mass concentrates on a single token (the argmax), making the sampling outcome completely predictable and deterministic with no uncertainty whatsoever. As temperature increases, entropy increases monotonically because probability mass spreads across more tokens, each contributing to the overall uncertainty. The maximum possible entropy (achieved as $\tau \to \infty$) equals $\log |\mathcal{V}|$, corresponding to a uniform distribution where all tokens are equally likely and each sample is maximally unpredictable. Practitioners often think of temperature as controlling "creativity" or "randomness": higher entropy means more variation in sampling outcomes, which can produce more surprising and original outputs but also increases the risk of sampling contextually inappropriate tokens that damage coherence. The entropy curve provides a principled way to understand and select temperature values based on the desired level of generation variability. However, temperature scaling alone does not solve all the problems of greedy decoding, despite its intuitive appeal and widespread use, because it does not prevent sampling very low-probability tokens that might produce nonsensical, ungrammatical, or contextually inappropriate output—at $\tau = 1.5$, a token with original probability 0.001 might become probable enough to sample occasionally, and such rare tokens often represent typographical errors, proper nouns that do not fit the context, or grammatically invalid continuations. A single such problematic token sampled at a critical position can derail an entire generation, making the output

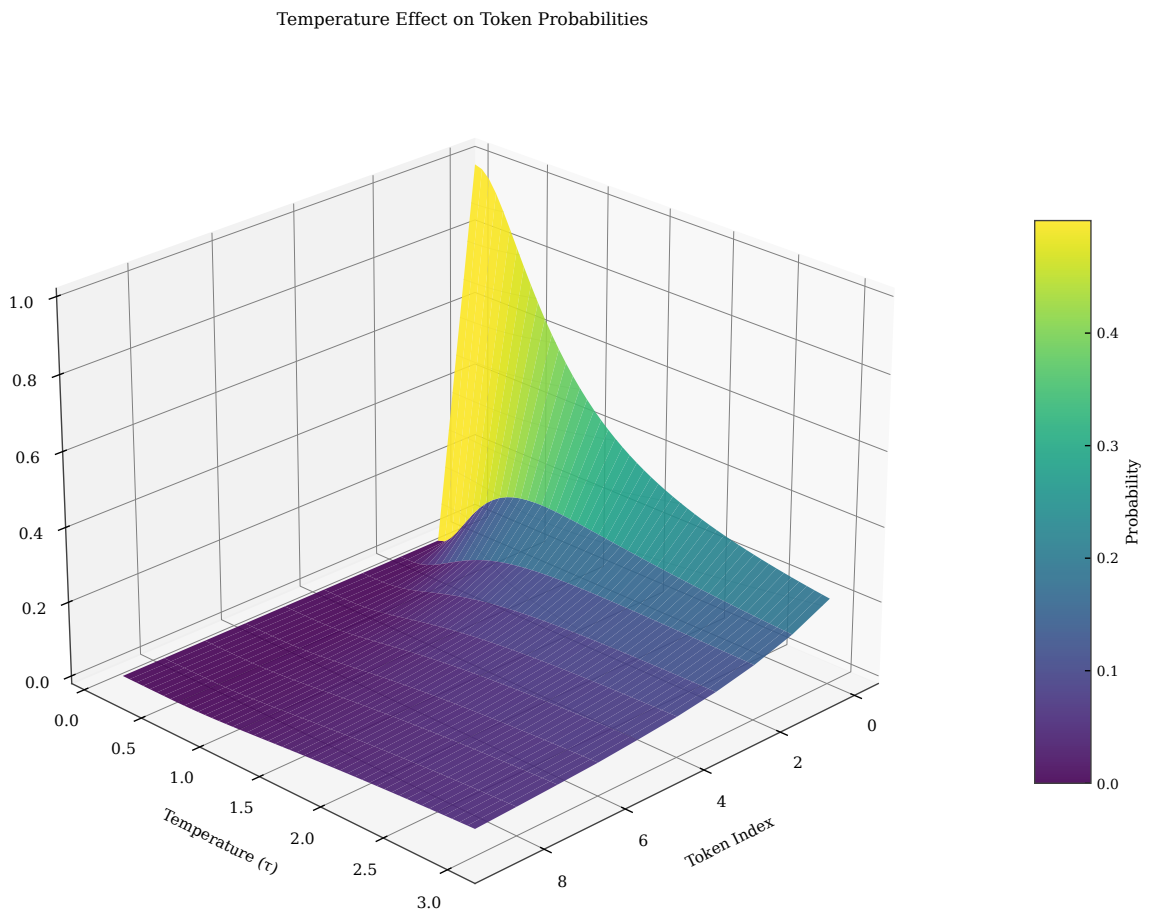

Figure 7.7: Three-dimensional surface showing probability as a function of token rank and temperature. At low temperatures (front), probability concentrates sharply on top-ranked tokens. At high temperatures (back), probability spreads more uniformly across ranks. The surface shape reveals how temperature systematically reshapes the distribution.

unusable despite all other tokens being reasonable, which motivates truncation strategies like top-k and nucleus sampling that combine temperature-like smoothing with hard cutoffs preventing sampling from the extreme tail.

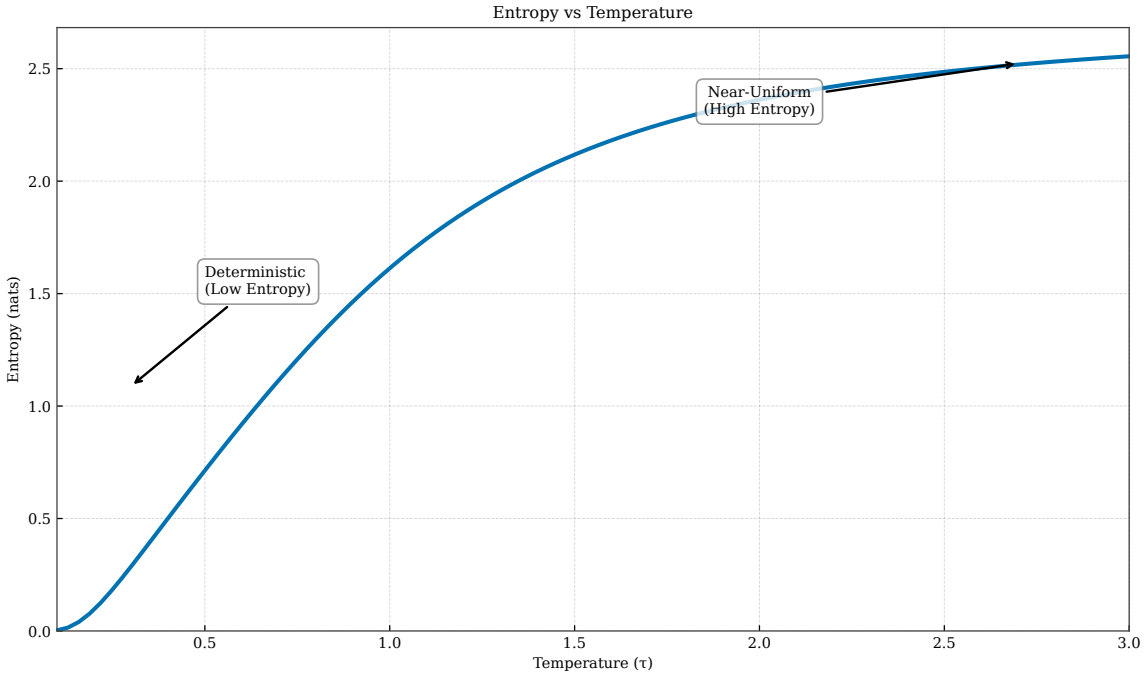

Figure 7.8: Entropy as a function of temperature. Entropy increases monotonically from near-zero (deterministic, equivalent to greedy decoding) to the maximum $\log|\mathcal{V}|$ (uniform distribution). Temperature controls the trade-off between predictable, safe outputs and diverse, creative outputs.

7.4 Top-k Sampling

Top-k sampling addresses the tail risk of pure temperature sampling by truncating the distribution to the k most probable tokens before sampling, providing a simple but effective guard against sampling contextually inappropriate tokens. Given the probability distribution $P(w)$, we first identify the k tokens with highest probability by sorting the distribution, set the probabilities of all other tokens (those ranked below k) to exactly zero, and then renormalize the remaining probabilities to sum to one so we have a valid probability distribution. Mathematically, if V_k denotes the set of k most probable tokens, the modified distribution is $P(w)' = P(w)/\sum_{w' \in V_k} P(w')$ for $w \in V_k$ and $P(w)' = 0$ otherwise. This hard truncation prevents sampling from the long tail of low-probability tokens that might produce nonsensical or jarring text, while still allowing meaningful diversity among the top candidates that the model considers contextually appropriate. The parameter k controls how much diversity is permitted: small k restricts choices severely and approaches greedy decoding at $k = 1$, while large k allows more exploration of the distribution but with greater risk of occasionally sampling mediocre options.

The three-dimensional visualization in Figure 7.9 shows how top-k truncation reshapes the probability landscape across different values of k, revealing the interaction between rank and truncation threshold. For $k = 5$, only the five highest-probability tokens receive nonzero probability after truncation, with all others masked to exactly zero and shown as a gray plane at the bottom representing the excluded portion of the vocabulary. As k increases progressively to 10, 20, and 50, more tokens become available for sampling, gradually restoring portions of the original distribution's shape. The surface reveals that truncation has two distinct effects: it completely eliminates tokens beyond rank k, preventing any possibility of sampling them, and it amplifies the relative probabilities of remaining tokens through renormalization so that a token originally at 10% of total probability mass becomes a substantially larger share when competing only against other top-ranked tokens. Top-k sampling was popularized by Fan et al. [2018] for story generation, where experiments found that k values between 10 and 50 produced good results balancing coherence with creativity, though the optimal k depends heavily on the model architecture, domain, and task requirements, with smaller models potentially needing smaller k to avoid sampling mediocre options.

Figure 7.10 compares the original untruncated distribution with the $k = 5$ filtered distribution on our run-

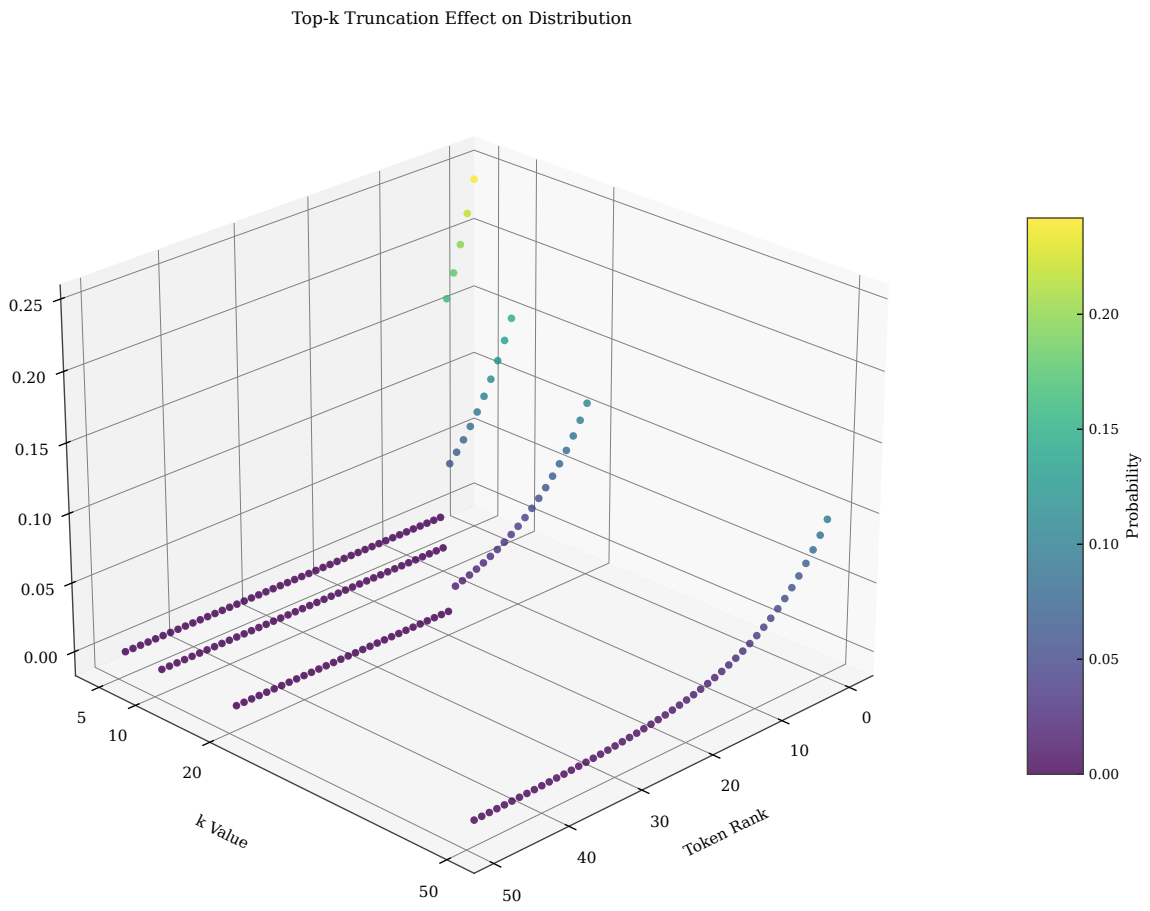

Figure 7.9: Three-dimensional visualization of top-k truncation. The surface shows probability as a function of token rank and k value. Tokens beyond rank k are masked (gray region at zero probability). Remaining tokens have their probabilities amplified through renormalization.

ning fantasy example, illustrating concretely what information is preserved and what is discarded. The original distribution assigns nonzero probability to many tokens across the vocabulary, including rare words like "cobbler," "turnip," and various grammatically awkward options that would sound strange continuing "there lived a young." After top-5 filtering, only "prince," "girl," "wizard," "knight," and "man" remain as candidates for sampling, with their probabilities renormalized to sum to one—"prince" now has approximately 30% probability rather than 15%. The filtering eliminates tail risk entirely: we cannot accidentally sample an unlikely word that would derail the narrative or break the fantasy genre conventions. However, this safety comes at the cost of diversity: if the context genuinely supports an unusual but contextually perfect word at position 6 or beyond in the original ranking, top-5 sampling will never select it regardless of how appropriate it might be. This reveals the fundamental limitation of top-k sampling: the fixed cutoff k ignores the shape of the probability distribution, applying the same rigid truncation regardless of whether the model is confident or uncertain. When the model is highly confident with probability concentrated in just a few tokens, $k = 50$ might wastefully include many nearly-zero-probability tokens that clutter the candidate set without providing meaningful diversity; when the model is genuinely uncertain with probability spread uniformly across many reasonable tokens, $k = 50$ might aggressively exclude perfectly valid options that should legitimately be considered.

Figure 7.11 illustrates the fixed-k problem with concrete examples that demonstrate why no single k value works well across all distribution shapes. For a peaked distribution where the top token has 80% probability, $k = 50$ wastefully includes 40+ tail tokens with negligible individual probability, cluttering the candidate set without adding meaningful diversity and slightly increasing the chance of sampling a mediocre option. For a flat distribution where many tokens share similar probability around 4-5% each, $k = 5$ aggressively excludes tokens

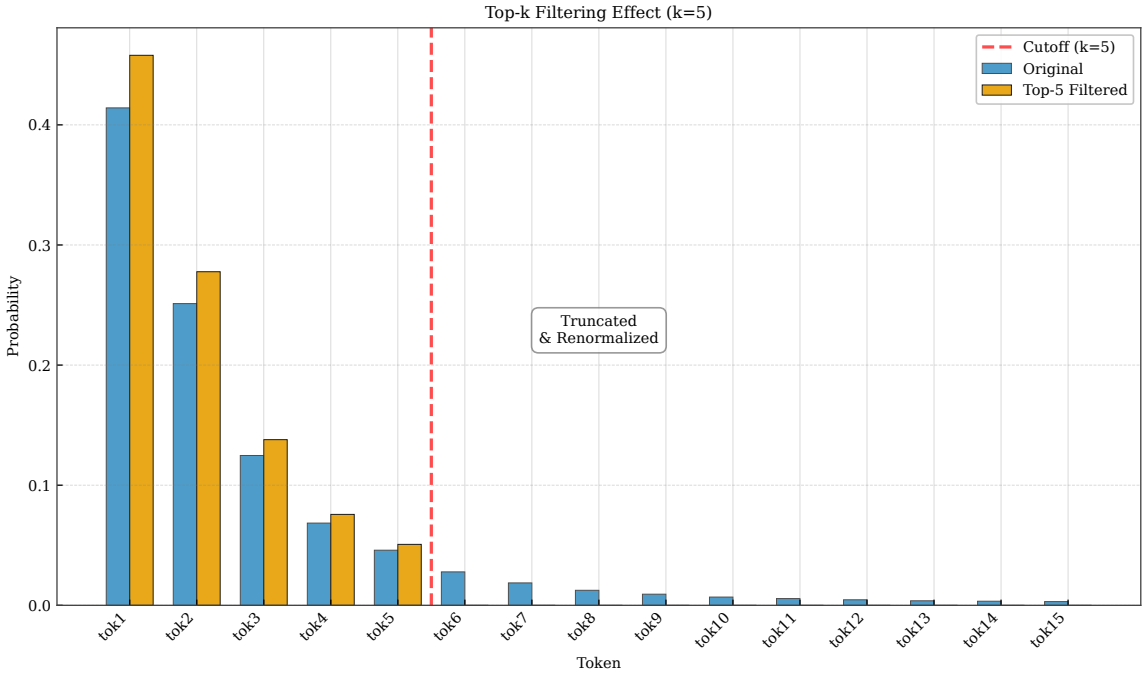

Figure 7.10: Comparison of original distribution (blue) with top-5 filtered distribution (orange). The vertical line marks the truncation point. Tokens beyond rank 5 are eliminated entirely, and remaining tokens have amplified probabilities after renormalization.

that are nearly as probable as those included, arbitrarily cutting off valid options that the model considers almost equally appropriate. The ideal cutoff depends on the distribution shape, which varies dramatically from position to position based on context, making any fixed k suboptimal for at least some positions during generation.

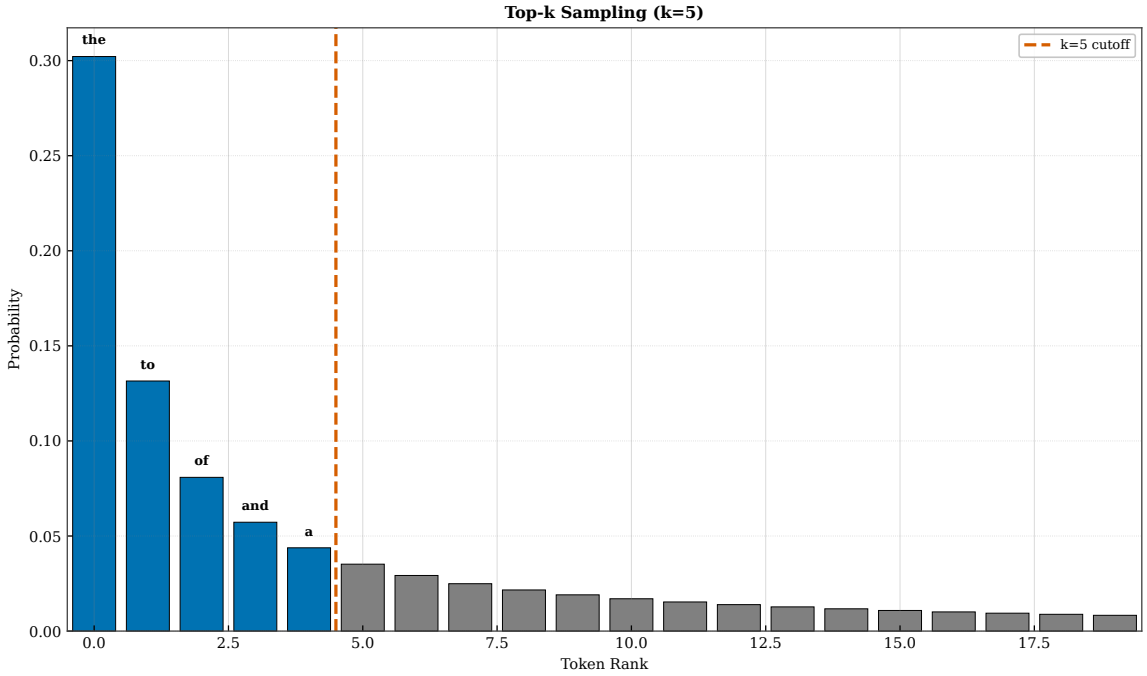

Figure 7.11: The fixed-k problem: the same k value behaves differently on peaked versus flat distributions. Left: for a peaked distribution, $k = 50$ includes many near-zero-probability tokens. Right: for a flat distribution, $k = 5$ excludes tokens nearly as probable as those included. Optimal k depends on distribution shape.

7.5 Nucleus (Top-p) Sampling

Nucleus sampling, also called top-p sampling, addresses the fixed-cutoff problem of top-k by defining the truncation in terms of cumulative probability rather than rank, allowing the size of the candidate set to adapt automatically to the distribution shape [Holtzman et al., 2020]. Instead of keeping a fixed number of tokens regardless of their probabilities, we keep the smallest set of tokens whose cumulative probability exceeds a threshold p, meaning we include exactly enough tokens to capture at least p fraction of the probability mass. Formally, we sort tokens by decreasing probability and include tokens until their cumulative probability first exceeds p: $V_p = \arg\min_{V \subseteq \mathcal{V}} |V|$ such that $\sum_{w \in V} P(w) \geq p$. This adaptive truncation means that for peaked distributions where few tokens dominate, the nucleus contains few tokens (perhaps just 3-5), while for flat distributions where probability is spread widely, the nucleus expands to include many tokens (perhaps 50-100), automatically adjusting to the model's confidence level at each position without requiring manual tuning of a rank-based cutoff.

The three-dimensional surface in Figure 7.12 visualizes how the nucleus boundary varies jointly with the probability threshold p and token rank, showing the adaptive nature of the truncation across the parameter space. At low p (e.g., 0.7), the nucleus typically contains only the very top tokens because 70% of probability mass is often captured by just a handful of tokens, while at high p (e.g., 0.95), the nucleus extends further into the tail to capture nearly all the probability mass while still excluding the extreme tail where problematic tokens reside. The surface shows that the nucleus boundary is not a simple function of rank like top-k but depends on the cumulative probability structure of each specific distribution: positions where the model is confident have sharp cutoffs at low ranks because few tokens capture most probability, while positions where the model is uncertain have gradual cutoffs extending to higher ranks because probability is distributed more evenly and more tokens are needed to reach the threshold.

Figure 7.13 demonstrates concretely how nucleus sampling adapts to different distribution shapes, contrasting its behavior on peaked versus flat distributions. In the left panel, the model is highly confident with probability concentrated in just 3 tokens that together account for 90% of the probability mass; nucleus sampling with $p = 0.9$ includes exactly these 3 tokens, avoiding the wasteful inclusion of dozens of tail tokens

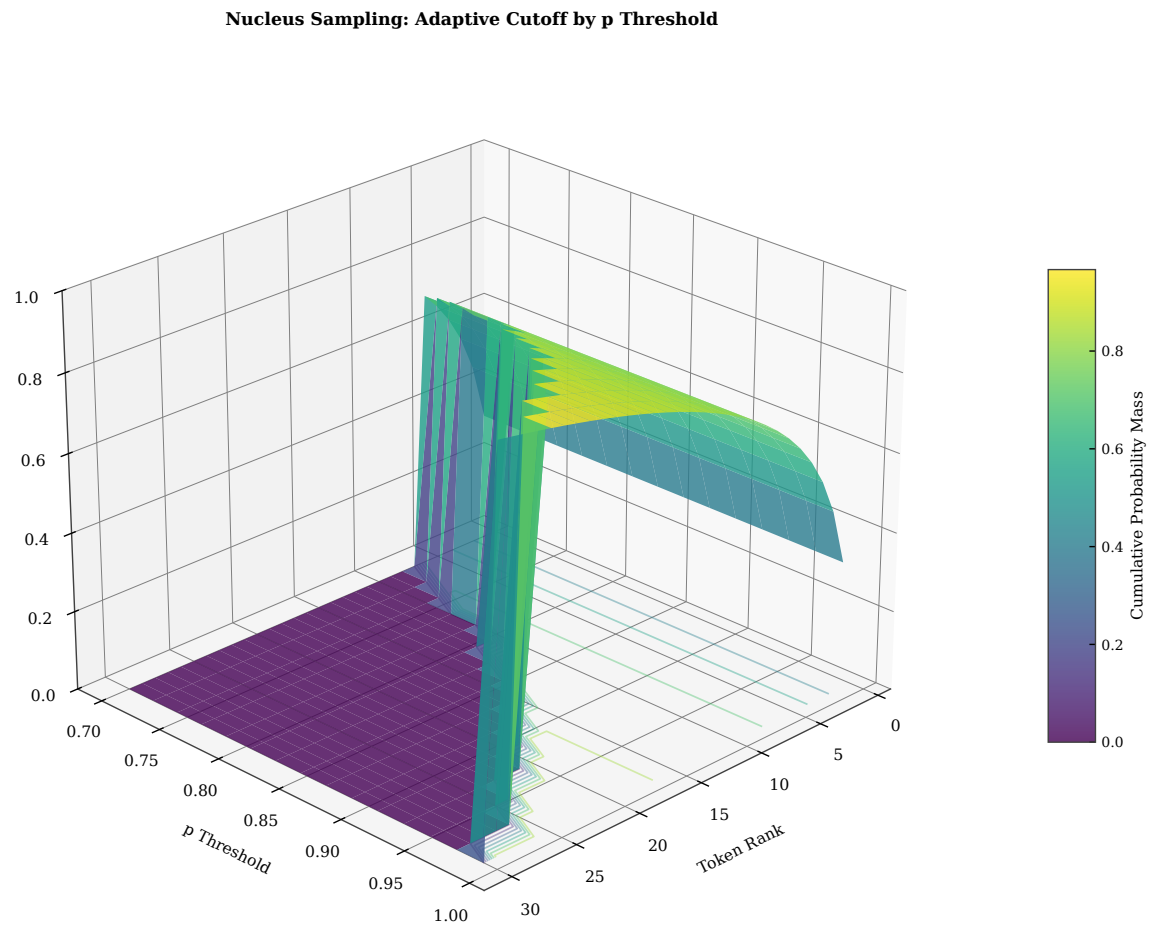

Figure 7.12: Three-dimensional surface showing nucleus membership as a function of token rank and probability threshold p. Higher p values include more tokens in the nucleus. The boundary adapts to the probability distribution: peaked distributions have sharp cutoffs, while flat distributions have gradual cutoffs.

that would occur with a fixed $k = 50$. In the right panel, the model is genuinely uncertain with probability spread across 20 tokens each around 4.5%; nucleus sampling includes all 20 tokens, capturing this genuine uncertainty rather than arbitrarily truncating at $k = 5$. This adaptivity is the key advantage of nucleus sampling over top-k: it respects the model's varying confidence level across positions. The cumulative probability view in Figure 7.14 provides the most direct visualization of how the threshold p determines the nucleus boundary: tokens are sorted by probability from highest to lowest, and the first position where cumulative probability exceeds p marks the boundary. Higher p means including more tokens to capture more probability mass, while lower p means fewer tokens; the step-function shape of the cumulative curve also reveals how peaked distributions have steep early steps and reach 90% quickly, while flat distributions have gradual steps requiring many more tokens.

Nucleus sampling can be productively combined with temperature scaling, applying both techniques in sequence: first adjust the distribution with temperature to control overall sharpness, then apply nucleus truncation to the temperature-scaled distribution to control tail risk. The combined approach gives practitioners two independent knobs to control different aspects of generation behavior: temperature controls the overall entropy and how peaked versus flat the distribution is, while the nucleus threshold controls tail truncation and prevents sampling from the extreme low-probability region regardless of temperature. Common settings in practice include $p = 0.9$ or $p = 0.95$ for the nucleus threshold combined with temperature around 0.7 to 1.0, though optimal values depend on the model and task. The Holtzman et al. [2020] paper that introduced nucleus sampling demonstrated its advantages over top-k for open-ended text generation tasks, showing through both automated

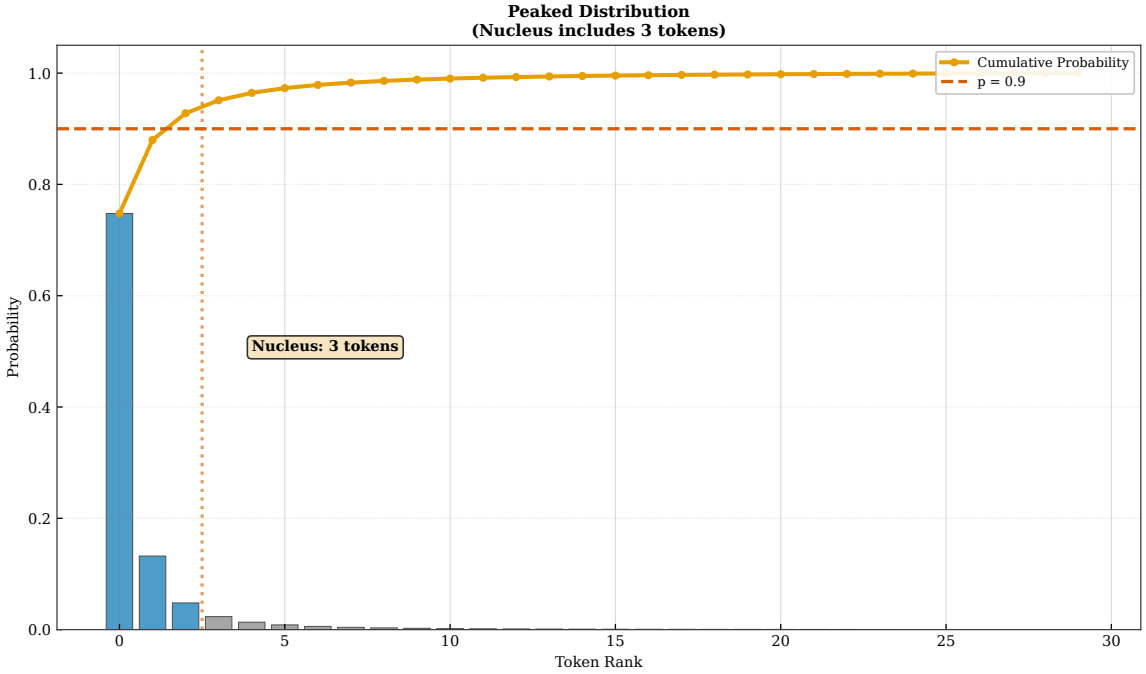

Figure 7.13: Nucleus sampling adapts to distribution shape. Left: peaked distribution, nucleus contains 3 tokens. Right: flat distribution, nucleus contains 20 tokens. Both use $p = 0.9$, but the resulting set size differs based on how probability mass is distributed.

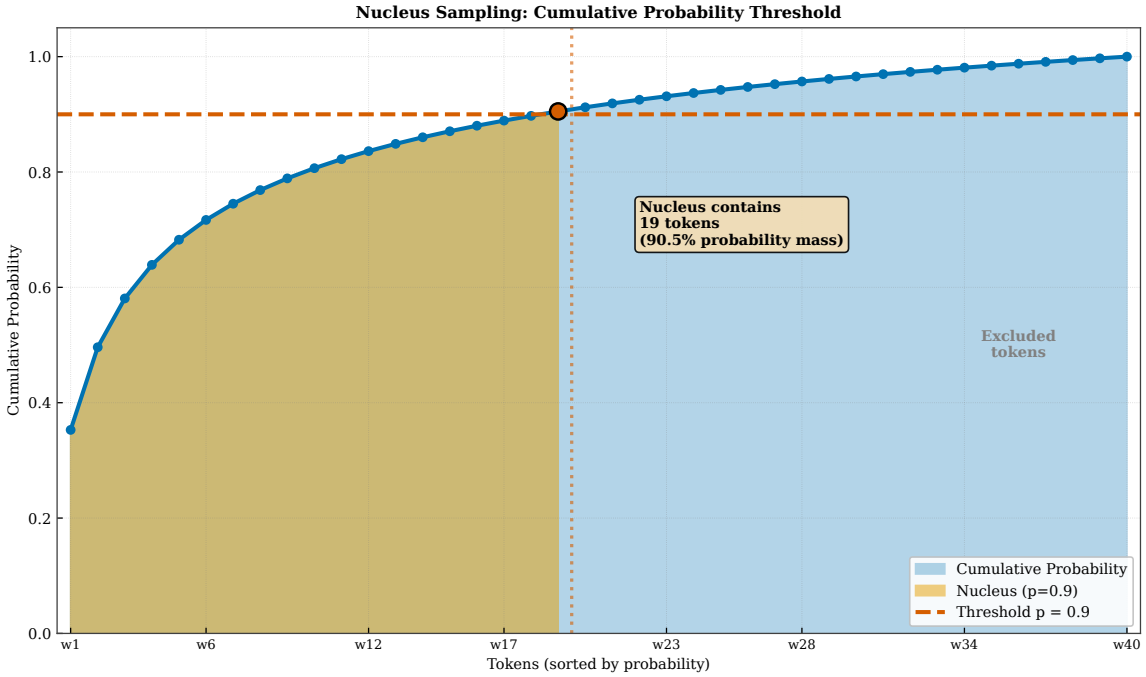

Figure 7.14: Cumulative probability view of nucleus sampling. Tokens are sorted by probability (x-axis) and we plot cumulative probability (y-axis). The nucleus includes all tokens up to where cumulative probability first exceeds p (horizontal line). The shaded region represents tokens in the nucleus.

metrics and human evaluation that nucleus sampling produces more human-like text with fewer degenerate repetition patterns than alternative methods.

7.6 Typical Sampling

Typical sampling takes an information-theoretic approach to the truncation problem, motivated by the observation that humans tend to produce text that is neither too predictable nor too surprising—natural language occupies a middle ground of "typical" information content [Meister et al., 2023]. Rather than including tokens based on their probability rank (as in top-k) or cumulative probability (as in nucleus sampling), typical sampling includes tokens whose information content is close to the expected information content (entropy) of the distribution. The information content of a token is defined as $-\log P(w)$, which is low for high-probability tokens (predictable, few bits of surprise) and high for low-probability tokens (surprising, many bits of information). Typical sampling keeps tokens whose information content falls within a specified range around the entropy $H(P) = \mathbb{E}_{[}P[-\log P(w)]$, excluding both tokens that are too predictable (very high probability, information content below the typical range) and tokens that are too surprising (very low probability, information content above the typical range). This two-sided filtering based on typicality rather than raw probability represents a fundamentally different philosophy of what makes a good token choice.

Figure 7.15 visualizes the typical set in terms of token probability and information content, showing which tokens are included and excluded by the typical sampling criterion. Each point in the visualization represents a token, with position on the x-axis showing its probability and position on the y-axis showing its information content ($-\log p$). Note that these quantities are inversely related: high probability corresponds to low information content, and vice versa. The horizontal band around the entropy value marks the typical set: tokens falling within this band are considered for sampling, while tokens outside are excluded. Critically, note that the typical set excludes both extremes: very high-probability tokens (on the left side of the plot, with low information content) that would make text too predictable and boring, and very low-probability tokens (on the right side, with high information content) that would make text too surprising and potentially incoherent. This two-sided filtering fundamentally distinguishes typical sampling from top-k and nucleus sampling, which only filter the low-probability tail and always include the highest-probability token.

Figure 7.16 compares typical sampling with nucleus sampling on a concrete example distribution, highlighting the different tokens each method selects and the philosophical difference in their approaches. For a given distribution, nucleus sampling (shown in the left panel) includes all tokens above a cumulative probability threshold, which by construction necessarily includes the highest-probability token since cumulative probability starts with that token. Typical sampling (shown in the right panel) may exclude the highest-probability token because its information content is below the typical range—the method considers it "too obvious" a choice that would make the text predictable. The overlap between the two methods, shown in the center, contains tokens that are both highly probable (included by nucleus) and informationally typical (included by typical sampling). Tokens in the nucleus set but not the typical set are "too predictable" by the typicality criterion; tokens in the typical set but not the nucleus are "too surprising" for nucleus sampling's cumulative threshold but informationally appropriate according to the typical sampling philosophy.

Typical sampling is motivated by rate-distortion theory and the concept of typical sequences in information theory, connecting text generation to fundamental principles established by Claude Shannon. In a long sequence of independent samples from a distribution, the law of large numbers implies that the average information content of the sequence converges to the entropy of the distribution. Sequences whose average information content deviates significantly from the entropy are exponentially unlikely to occur—they are "atypical" in a precise mathematical sense. This theoretical insight suggests that human-like text should have information content close to what the language model expects on average, neither too high (overly surprising, containing too much information) nor too low (overly predictable, containing too little information). While typical sampling is theoretically elegant and has strong mathematical foundations, it has seen less widespread adoption than nucleus sampling in practical applications, partly because the information-theoretic motivation is less intuitive to practitioners and partly because the exclusion of high-probability tokens can sometimes hurt coherence in domains where the obvious continuation really is the best one.

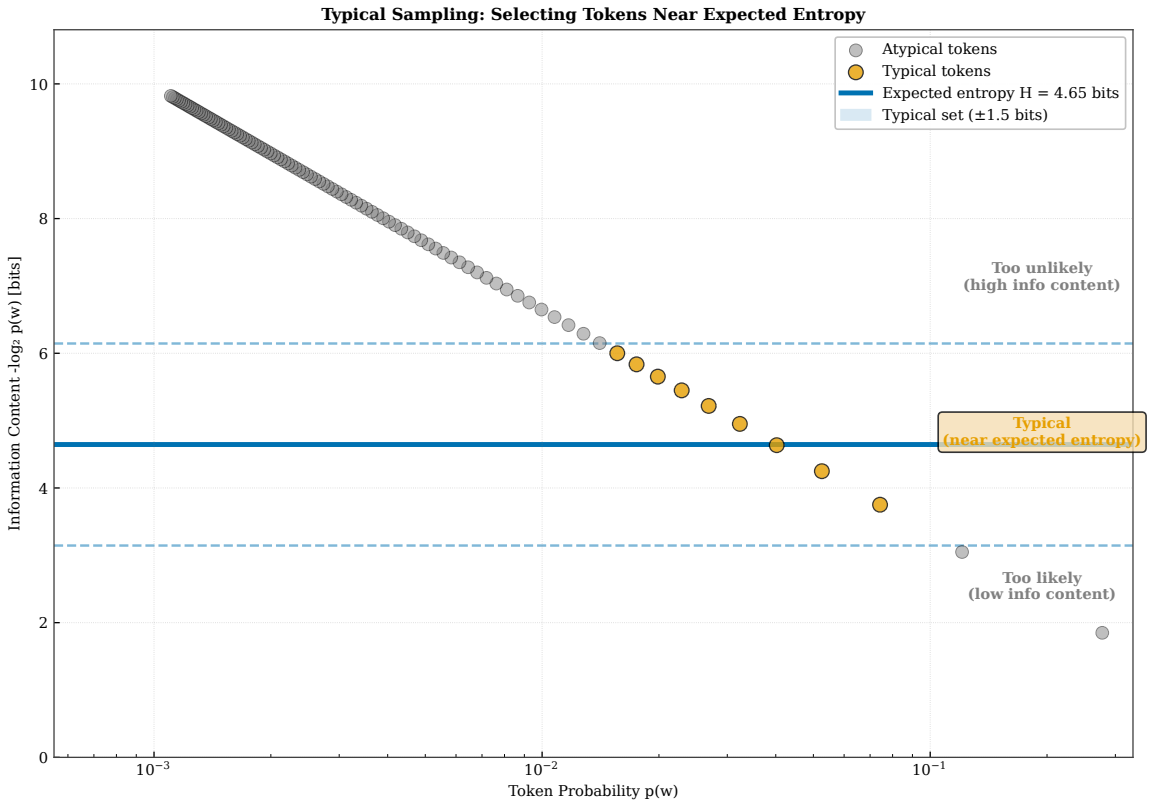

Figure 7.15: The typical set in typical sampling. X-axis shows token probability; y-axis shows information content ($-\log p$). The horizontal band around the entropy marks the typical set. Unlike nucleus sampling, typical sampling excludes both very likely tokens (too predictable) and very unlikely tokens (too surprising).

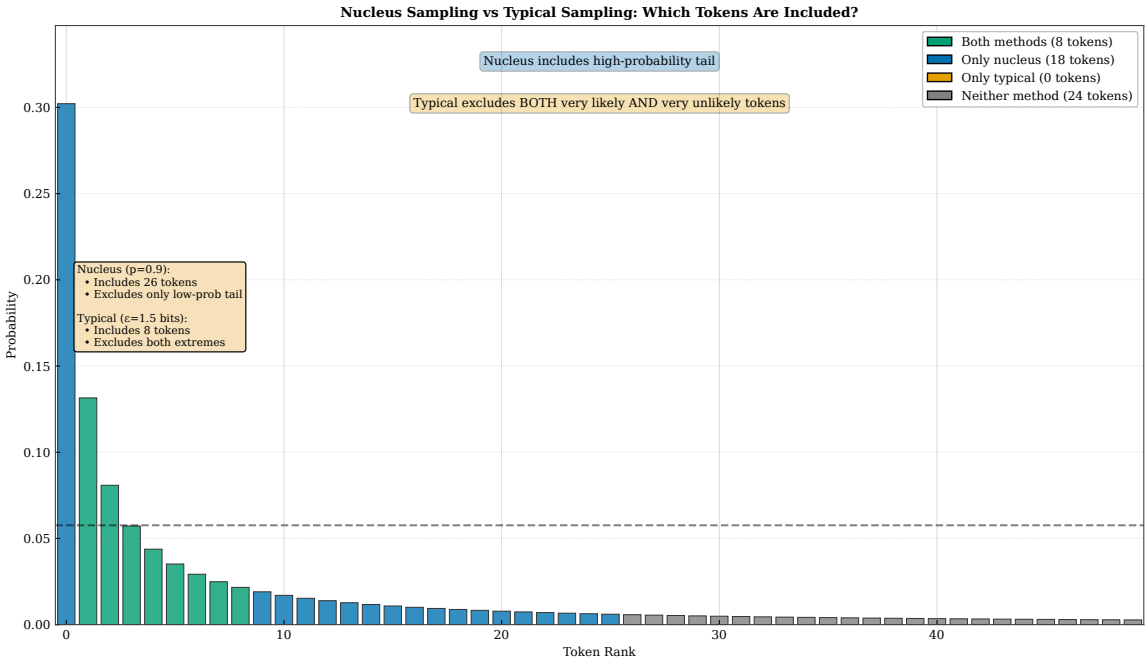

Figure 7.16: Comparison of nucleus and typical sampling. Nucleus sampling includes all tokens above a cumulative probability threshold. Typical sampling includes tokens with information content near the expected value, potentially excluding both very likely and very unlikely tokens.

7.7 Beam Search

While sampling methods generate diverse outputs by incorporating controlled randomness, beam search takes an entirely deterministic approach to finding high-probability sequences by maintaining multiple hypotheses simultaneously throughout the generation process. Beam search keeps the B most probable partial sequences (called "beams") at each step, expanding each beam with all possible next tokens to generate candidate continuations and then pruning back to the top B candidates based on cumulative probability. Unlike greedy decoding, which commits irrevocably to a single choice at each step, beam search explores B alternative paths through the search space simultaneously, allowing recovery from locally suboptimal choices if a different path leads to a higher-scoring sequence overall when considered as a complete unit. This hedging against early mistakes makes beam search particularly valuable for structured generation tasks like machine translation and summarization, where output quality is measured by metrics like BLEU that reward globally coherent sequences rather than locally optimal individual token choices, and where the correct translation of one word often depends on choices made later in the sentence.

The three-dimensional tree in Figure 7.17 visualizes beam search with $B = 3$ on our running fantasy example, showing how the algorithm maintains and updates multiple hypotheses over time. The x-axis represents time steps (token positions), the y-axis represents beam index (which of the 3 hypotheses we are examining), and the z-axis represents cumulative log-probability (the score used to rank beams). At each step, all three beams are expanded by considering all possible next tokens from the vocabulary, generating $3 \times |\mathscr{V}|$ candidate continuations, then pruned back to the three highest-scoring partial sequences based on their total log-probability. Lines connecting nodes show which beams survive to the next step; the vast majority of expansions are pruned away as lower-scoring than the survivors. The final output is typically the beam with the highest cumulative score after reaching the end-of-sequence token or a maximum length limit, though sometimes the top-k final beams are all returned for downstream reranking.

Figure 7.18 shows a two-dimensional view of beam search paths that emphasizes the token sequences and their scores, making it easier to see which words are generated and why some paths survive while others are pruned. Each path represents a different beam (hypothesis), with nodes labeled by the tokens generated at each position and edges weighted by the log-probabilities of those token choices. The cumulative score of a path is simply the sum of log-probabilities along its edges, which represents the log-probability of the entire partial sequence under the model—higher scores mean more probable sequences. At each step t, beam search keeps the B paths with highest cumulative score, discarding all others regardless of how promising they might have seemed earlier. The final step shows the three surviving beams, with the highest-scoring beam typically selected as the output. The figure illustrates a key property of beam search: it can recover from a seemingly suboptimal choice at step 1 by finding that a different choice at step 2 or 3 leads to higher overall probability, justifying the earlier deviation from the greedy path.

A critical issue with naive beam search is that it systematically favors shorter sequences, often terminating prematurely with incomplete outputs. This bias arises because we accumulate log-probabilities (which are negative since probabilities are between 0 and 1), so longer sequences necessarily have lower (more negative) cumulative scores simply by having more terms in the sum, regardless of whether the individual terms represent good choices. This length bias is addressed by length normalization [Wu et al., 2016]: instead of ranking beams by raw log-probability $\sum_{t=1}^{T} \log P(w[t] \mid w[<t])$, we divide by a function of length to get a normalized score that removes the inherent penalty for being longer: $\text{score} = \frac{1}{T^{\alpha}} \sum_{t=1}^{T} \log P(w[t] \mid w[<t])$, where α is the length penalty parameter that controls how aggressively we normalize. Figure 7.19 shows how different values of α affect the ranking of sequences of different lengths: at $\alpha = 0$, there is no length normalization and shorter sequences are strongly preferred; at $\alpha = 1$, we divide by length exactly, normalizing to per-token log-probability; values between 0 and 1, particularly around $\alpha = 0.6$, are commonly used in practice, providing partial correction for length bias while still maintaining some preference for conciseness. For sequence-to-sequence tasks like machine translation, beam search can additionally suffer from under-translation (failing to translate some source content) or over-translation (translating the same source content multiple times), which coverage penalty [Tu et al., 2016] addresses by explicitly penalizing beams that repeatedly attend to the same source positions or fail to attend to some positions at all, as shown in Figure 7.20.

The coverage mechanism is typically computed from the cross-attention weights in encoder-decoder mod-

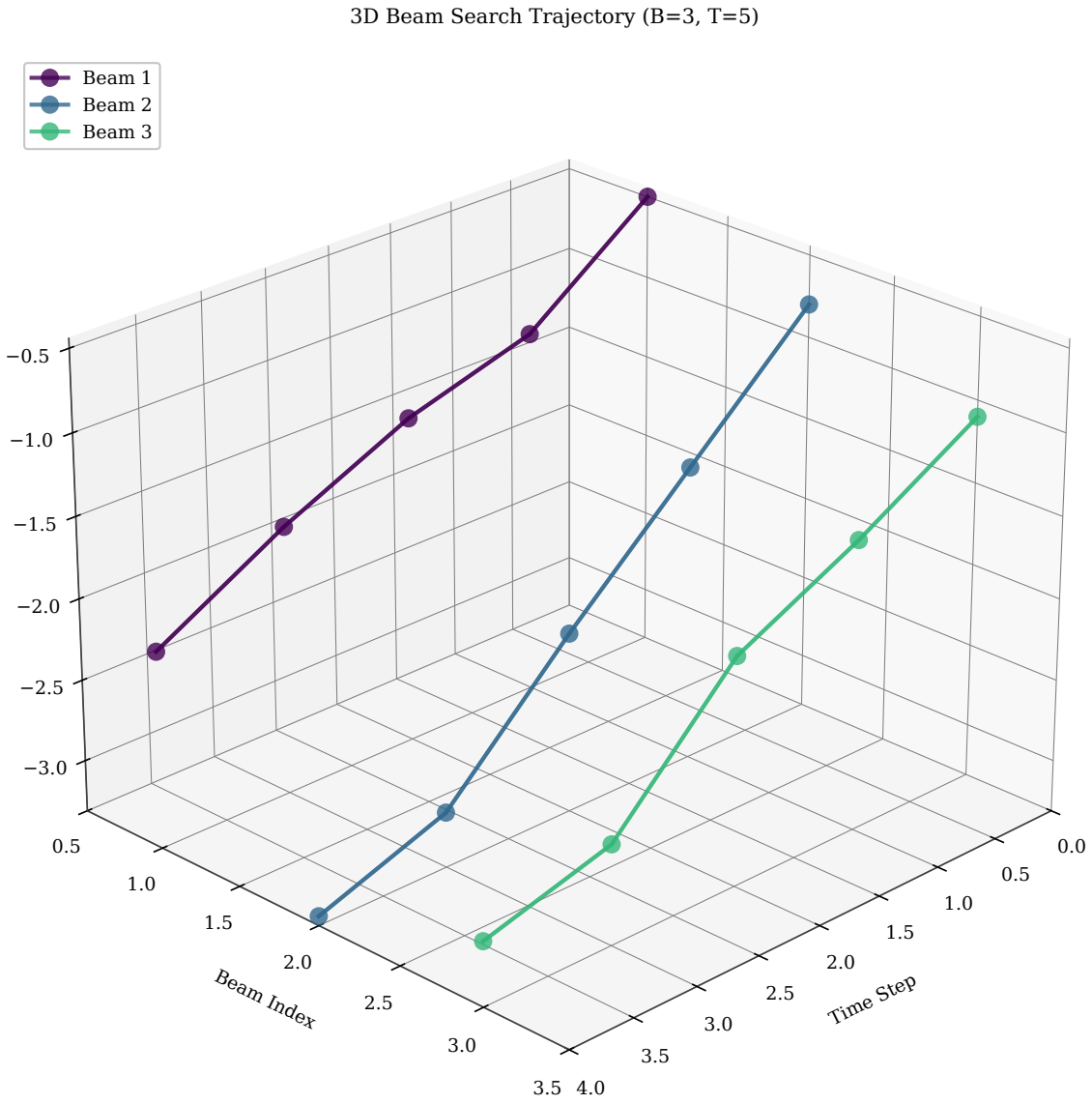

Figure 7.17: Three-dimensional visualization of beam search with $B = 3$. X-axis: time steps. Y-axis: beam index. Z-axis: cumulative log-probability score. Each beam is expanded and pruned at each step, maintaining the top 3 highest-scoring partial sequences.

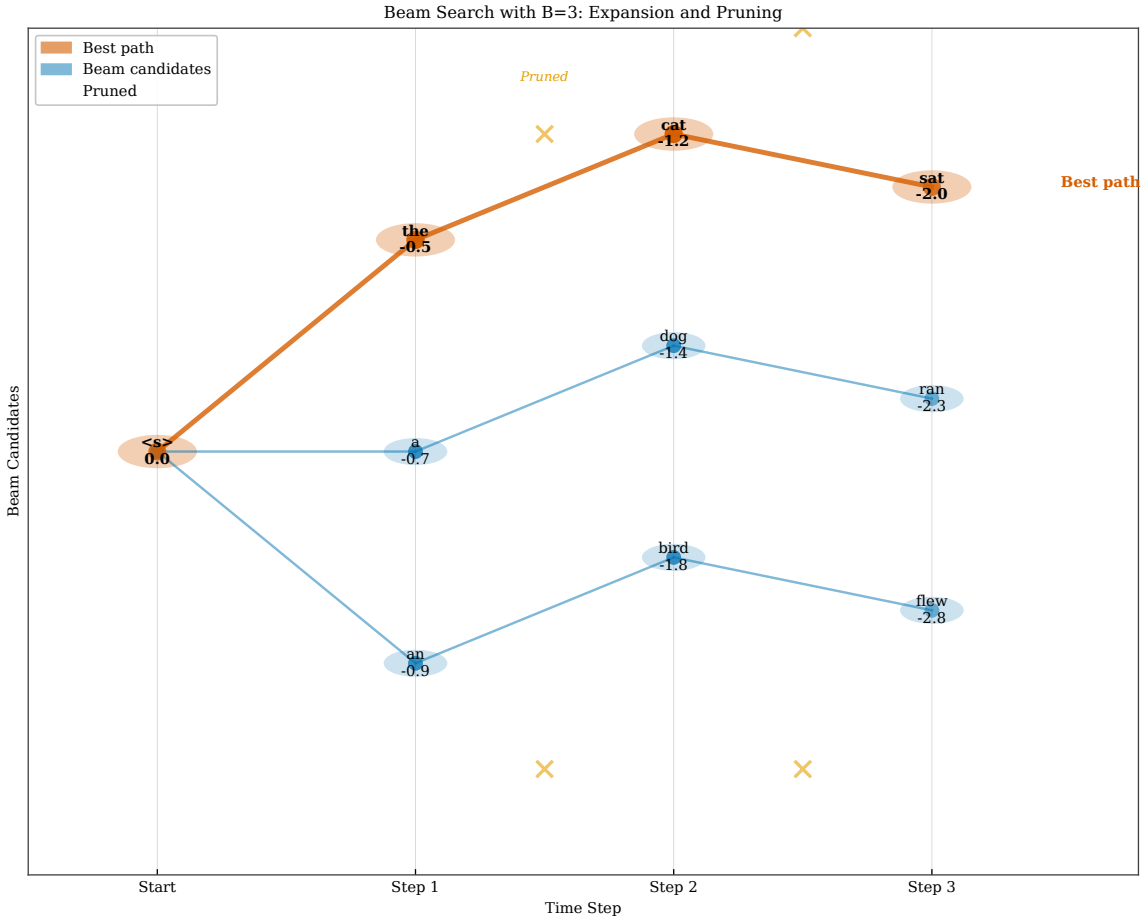

Figure 7.18: Two-dimensional view of beam search with $B = 3$. Each path shows tokens and their log-probabilities. Cumulative scores are shown at each node. Pruning at each step keeps only the highest-scoring beams, potentially dropping a path that started strong but became less probable.

els, tracking how much cumulative attention each source position has received across all decoding steps so far to ensure complete and non-redundant translation. Figure 7.20 visualizes coverage as a heatmap of attention over source positions (columns) and decoding steps (rows), where each cell shows the attention weight from that decoding step to that source position. Positions with too little cumulative attention (under-coverage, shown as cold colors) indicate content that may have been skipped, while positions with too much cumulative attention (over-coverage, shown as hot colors) indicate content that may be repeated. The coverage penalty adds a term to the beam score that encourages balanced attention, producing translations that cover the source content exactly once rather than missing or duplicating elements.

Despite its considerable success in structured generation tasks where a single high-quality output is desired, beam search has significant limitations for open-ended creative text generation. Like greedy decoding, beam search is completely deterministic, always producing the same output for a given input and beam width, which means it cannot generate the diversity of outputs that sampling methods provide. It tends to find high-probability but generic sequences that lack the diversity, creativity, and surprise expected in creative writing, dialogue, or brainstorming applications. Furthermore, beam search can suffer from a well-documented "blandness" or "genericness" problem: because it optimizes purely for probability, it systematically prefers safe, common phrases that appear frequently in training data over distinctive, interesting, or memorable formulations that might have slightly lower probability but much higher value. For these reasons, sampling-based methods like nucleus sampling are typically preferred for open-ended generation tasks, while beam search remains the standard and highly effective choice for translation, summarization, and other structured tasks where finding the single best output matters more than generating diverse alternatives.

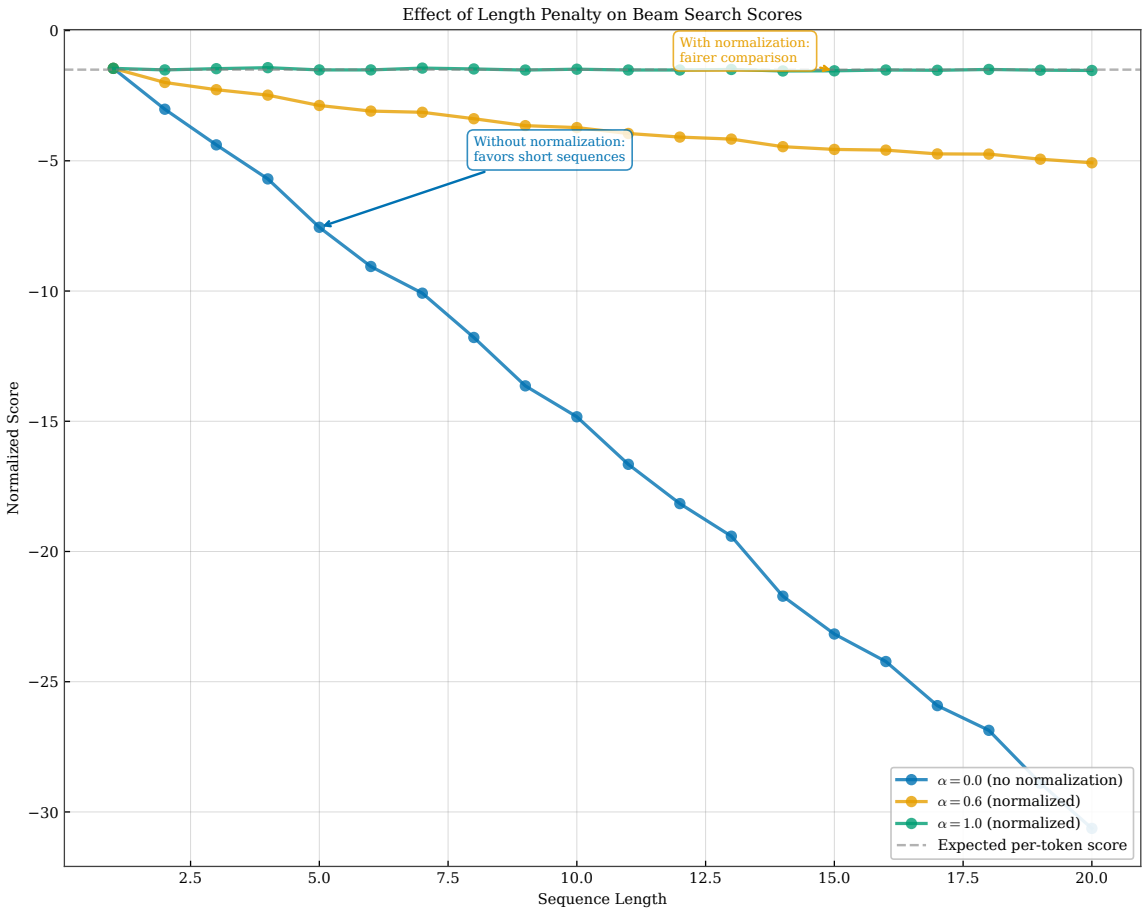

Figure 7.19: Effect of length normalization parameter α on sequence scores. Without normalization ($\alpha = 0$), longer sequences have lower scores. With full normalization ($\alpha = 1$), we compute per-token log-probability. Intermediate values like $\alpha = 0.6$ partially correct for length bias.

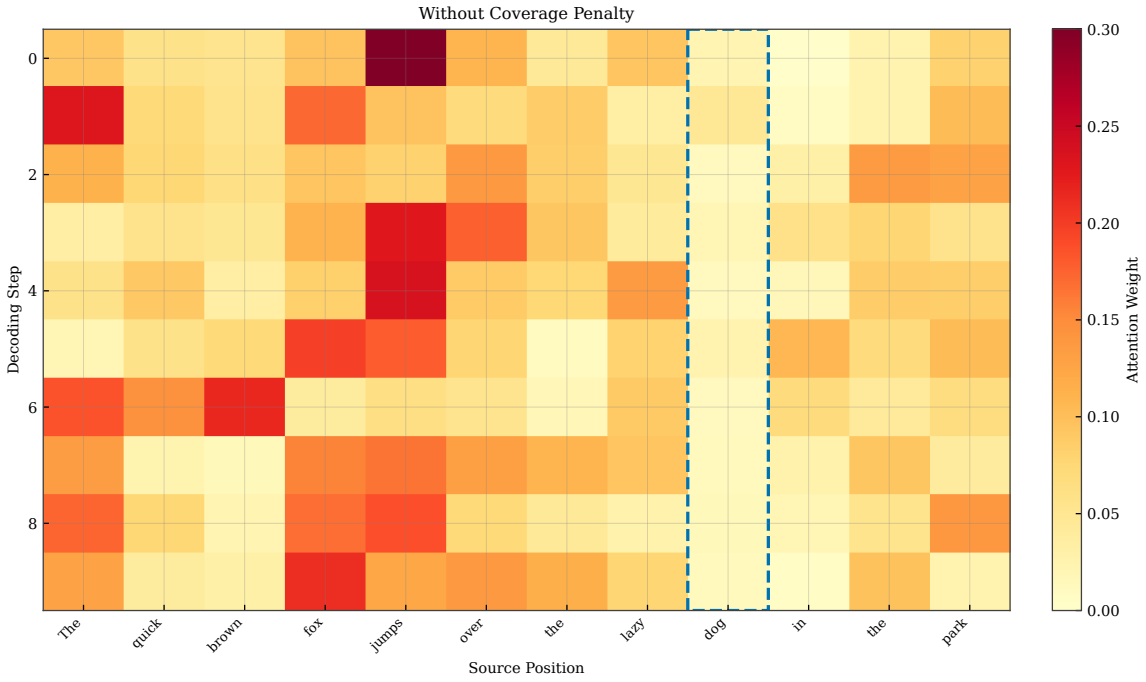

Figure 7.20: Coverage penalty visualization. Heatmap shows attention weights over source positions (columns) and decoding steps (rows). Coverage penalty penalizes under-attended positions (indicating potential missing content) and over-attended positions (indicating potential repetition).

7.8 Contrastive Decoding

Contrastive decoding improves generation quality by contrasting the predictions of a strong "expert" model with those of a weaker "amateur" model, exploiting the observation that failure modes are often more pronounced in less capable models [Li et al., 2023]. The key insight motivating this approach is that common failure modes like repetition, blandness, incoherence, and generic phrasing are more severe in smaller, less well-trained models—they lack the capacity to learn subtle preferences that distinguish high-quality text from mediocre text. By subtracting the amateur model's log-probabilities from the expert's, we mathematically amplify tokens that the expert prefers but the amateur does not, which empirically tend to be more interesting, distinctive, and less prone to degeneration. The contrastive score for token w is computed as $\log P_{\text{expert}}(w) - \lambda \log P_{\text{amateur}}(w)$, where λ controls the strength of the contrast between the two models. Additionally, a plausibility constraint ensures we only consider tokens where $P_{\text{expert}}(w)$ exceeds a minimum threshold, preventing the pathological case of selecting tokens that only appear good because the amateur model happens to strongly dislike them for idiosyncratic reasons unrelated to quality.

Figure 7.21 shows expert and amateur probability distributions for our running fantasy example, illustrating how the two models differ in their preferences. For completing "Once upon a time... young," both models assign relatively high probability to "prince" as the obvious and common continuation, but the amateur model also assigns substantial probability to repetitive or generic continuations that tend to lead to degeneration, while the expert model more strongly prefers diverse and distinctive continuations like "cartographer," "herbalist," or "tinker" that are unusual but contextually appropriate. The contrastive score amplifies these distinctive tokens through the subtraction: they have moderately high expert probability combined with low amateur probability, yielding high contrastive scores. Figure 7.22 demonstrates the qualitative improvement in practice: standard nucleus sampling produces serviceable but unremarkable text like "there lived a young prince who dreamed of adventure," while contrastive decoding produces "there lived a young cartographer whose maps revealed paths between worlds"—more specific with the unusual profession choice, more imaginative with the fantastical element, and more engaging overall by avoiding the generic patterns that appear countless times in training data.

Contrastive decoding requires running two models during inference rather than just one, which approximately doubles the computational cost and memory requirements compared to standard single-model decoding. However, the amateur model can be substantially smaller than the expert (e.g., a 125M parameter model serving as amateur contrasted with a 13B parameter expert), which limits the additional overhead to a manageable fraction of the total cost. The key requirement is that the amateur model be trained on similar data and capture similar patterns as the expert, just less well—this ensures that the difference between them captures quality-relevant distinctions rather than arbitrary differences in capability. Contrastive decoding is particularly effective for open-ended generation tasks like story writing and dialogue where the failure modes of smaller models (repetition, incoherence, blandness) are well-documented and where distinctiveness is valued. For structured tasks like translation where smaller models perform adequately and where the goal is accuracy rather than creativity, the benefits of contrastive decoding are less pronounced and the computational overhead may not be justified.

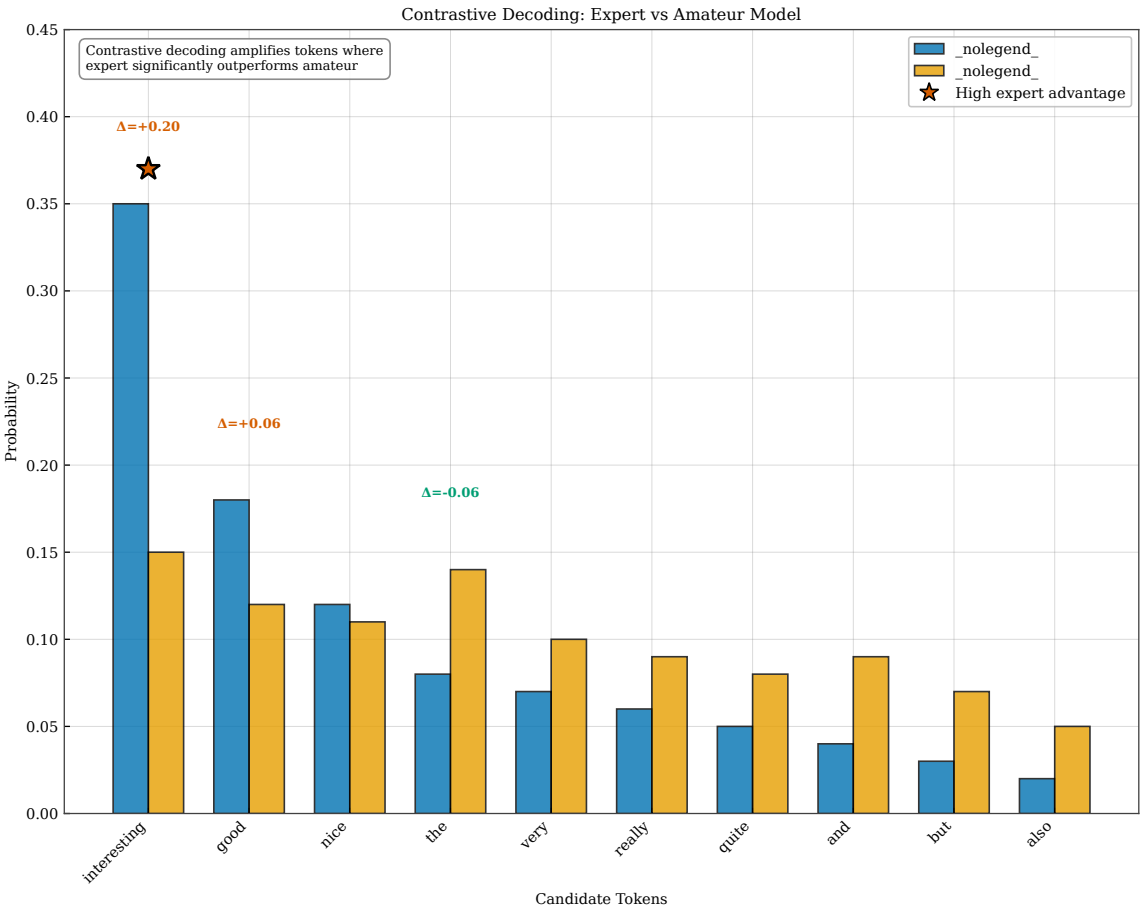

Figure 7.21: Expert and amateur probability distributions for contrastive decoding. Both models agree on common choices like “prince,” but the expert more strongly prefers distinctive options. Contrastive decoding amplifies tokens that the expert likes but the amateur does not.

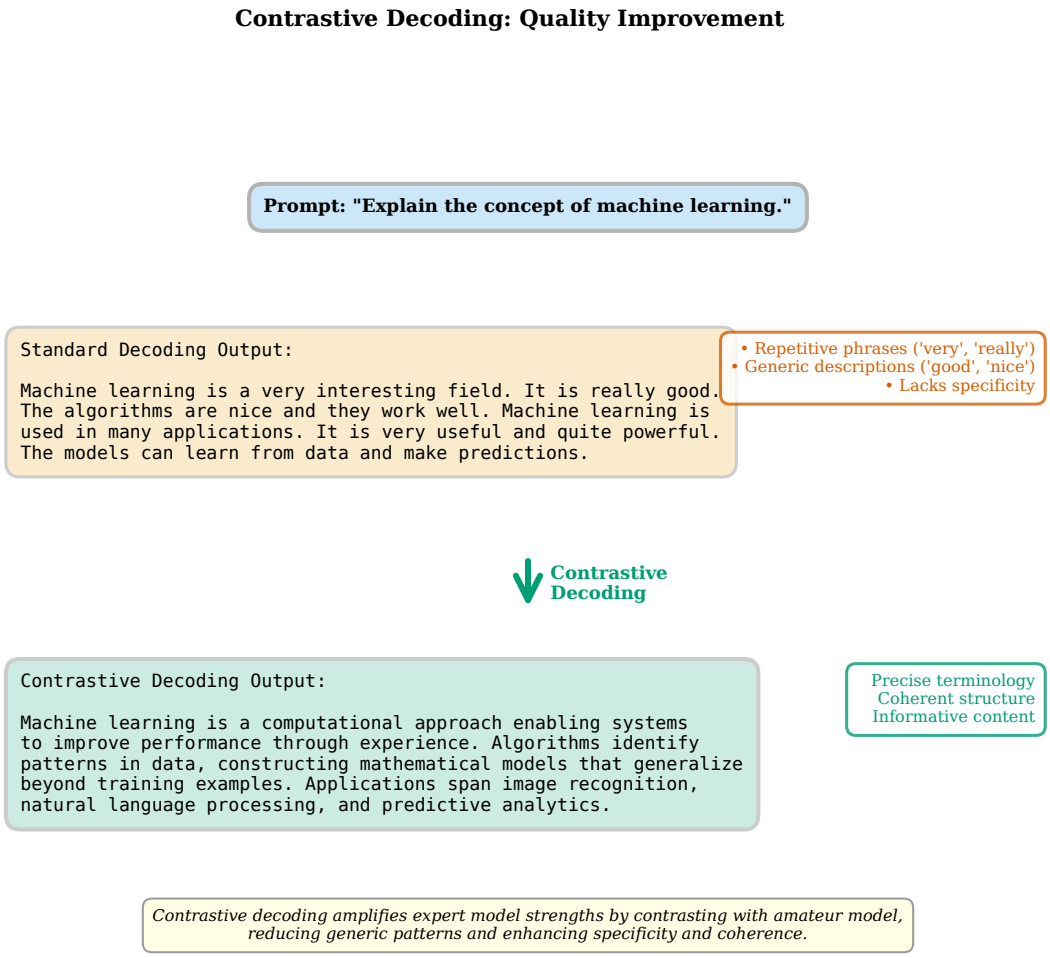

Figure 7.22: Comparison of standard versus contrastive decoding outputs. Standard decoding produces generic text following common patterns. Contrastive decoding produces more distinctive, creative text by amplifying tokens that differentiate the expert from the amateur model.

7.9 Constrained Decoding

Constrained decoding enforces that generated text must satisfy specific requirements, such as containing certain words, following particular formats, or adhering to grammatical structures, providing a way to control generation beyond what prompt engineering alone can achieve. Unlike the free-form decoding strategies discussed so far that allow any grammatically valid continuation, constrained decoding modifies the search process to guarantee constraint satisfaction, ensuring that the output definitely includes required elements rather than merely hoping the model produces them. The most common form is lexical constraint: requiring that specific words or phrases appear somewhere in the output regardless of where the model's probability distribution would naturally place them. For example, we might require that a story about dragons must include the word "dragon" at least once, or that a product description must include specific feature keywords like "waterproof" and "lightweight." Grid beam search [Hokamp and Liu, 2017] and NeuroLogic decoding [Lu et al., 2022] are sophisticated algorithms that efficiently search for high-probability sequences satisfying such constraints while avoiding the combinatorial explosion that naive approaches would encounter.

Figure 7.23 illustrates grid beam search for generating our fantasy story with the constraint that the output must include both "dragon" and "treasure," demonstrating the two-dimensional structure of the constrained search. The grid has decoding steps on the x-axis (progress through the sequence) and constraint satisfaction level on the y-axis (how many required constraints have been satisfied). Beams move rightward by generating tokens and upward when they generate a token satisfying a pending constraint; the algorithm ensures only complete sequences at the top level (all constraints satisfied) are considered as final outputs. Figure 7.24 shows the tree of valid sequences, with paths successfully including both words highlighted in green as valid candidates, while paths reaching end-of-sequence without satisfying all constraints are grayed out and discarded. The constrained decoding algorithm actively prunes invalid paths during search rather than generating freely and filtering afterward, which is crucial for efficiency since rejection sampling would be extremely slow for rare constraints that most unconstrained generations would fail to satisfy. The algorithm might produce output like "there lived a young knight who sought the dragon's treasure hidden in the mountain cave," where both constraint words appear naturally in a coherent narrative.

Beyond simple lexical constraints requiring specific words, constrained decoding can enforce a wide variety of structural requirements that would be difficult or impossible to achieve through prompting alone. These include JSON formatting where the output must be valid parseable JSON with specific required fields, grammatical patterns where certain syntactic structures must appear, length limits where the output must fall within a specific word or token count range, and rhyme or meter constraints for poetry generation. NeuroLogic decoding extends the basic grid beam search framework with lookahead heuristics that estimate the probability of satisfying remaining constraints from any partial sequence, improving efficiency by prioritizing beams that are likely to succeed and deprioritizing beams that would require low-probability token sequences to satisfy their remaining constraints. These techniques are particularly valuable in applications like information extraction (where outputs must follow specific schemas to be parsed by downstream systems), code generation (where outputs must be syntactically valid in the target programming language), and controlled text generation (where outputs must match specific style or content requirements specified by users).

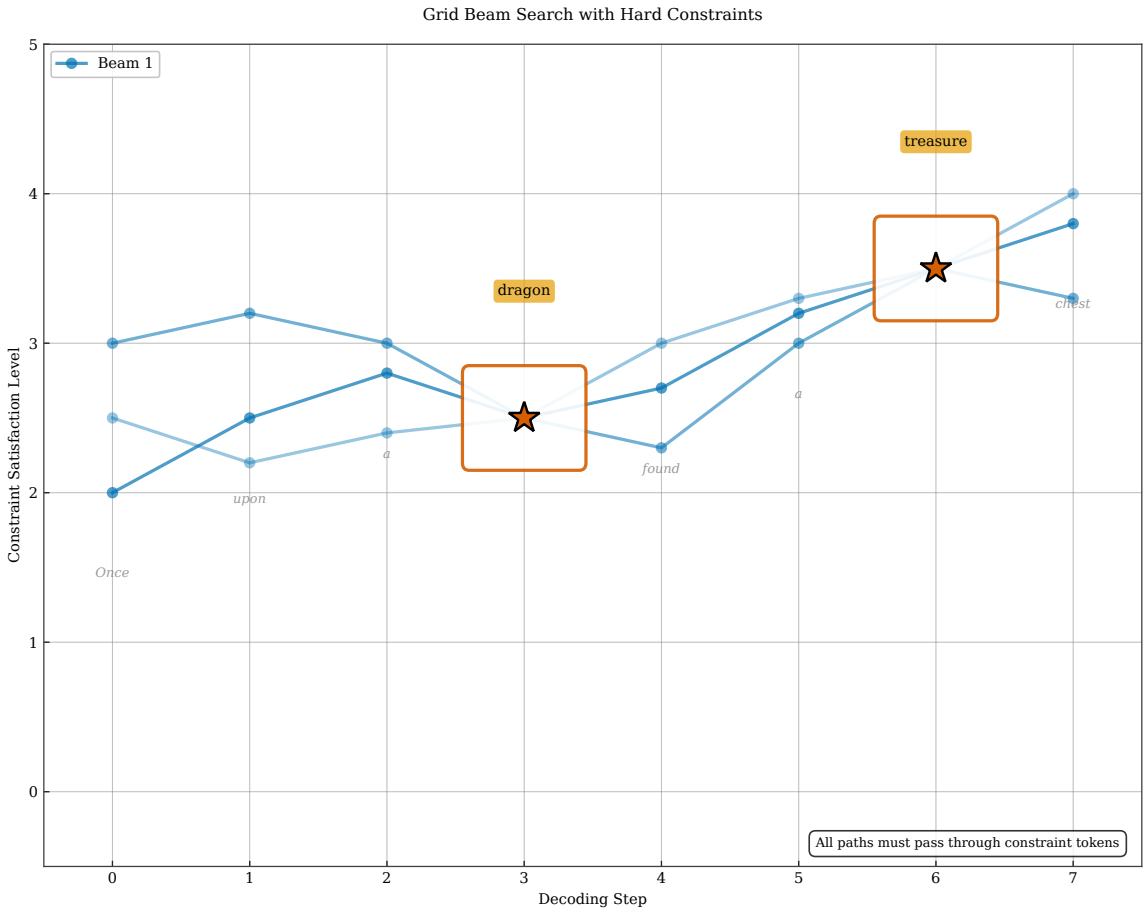

Figure 7.23: Grid beam search for constrained decoding. X-axis: decoding steps. Y-axis: number of constraints satisfied. Beams progress rightward and upward, with the final output required to reach the top level (all constraints satisfied). Red markers indicate where constraint words are generated.

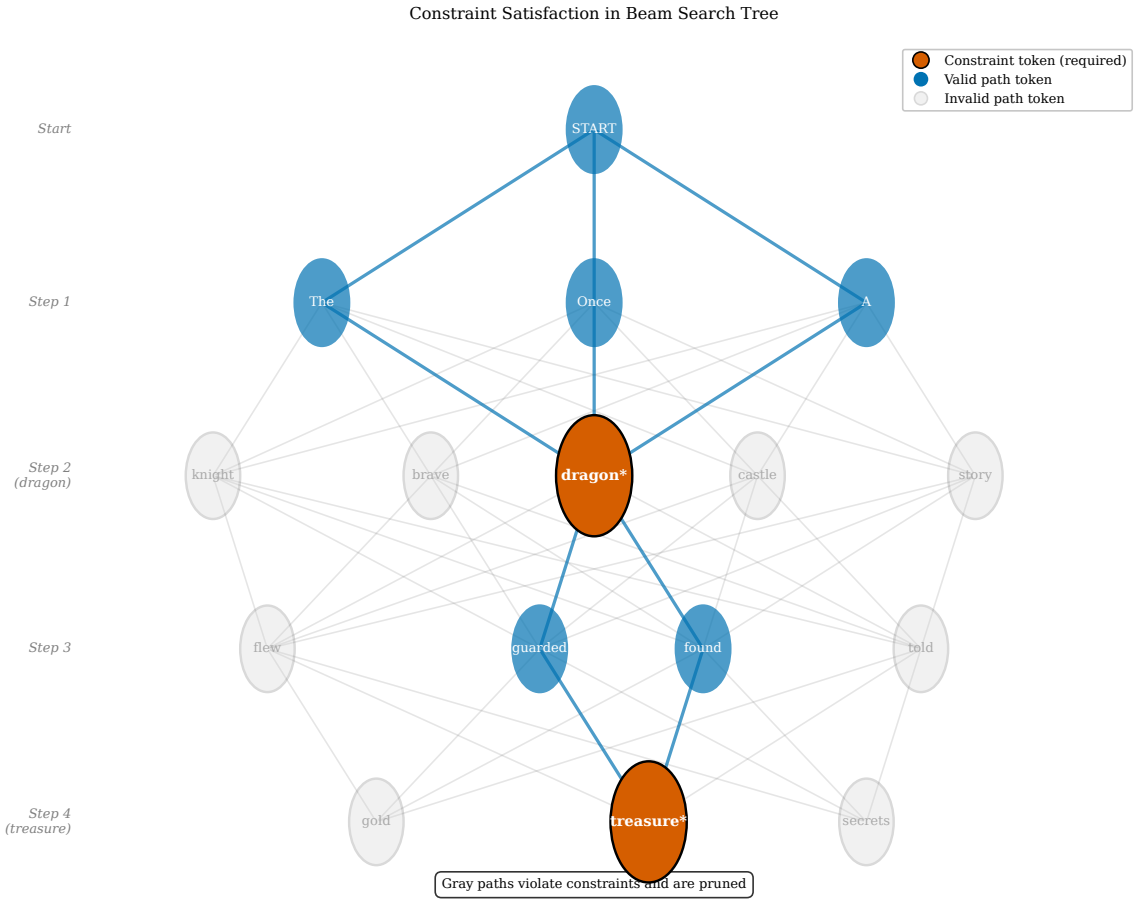

Figure 7.24: Constraint satisfaction paths in the decoding tree. Valid paths (green) include all required words (“dragon” and “treasure”). Invalid paths (gray) fail to satisfy constraints. Constrained decoding prunes invalid paths during search.

7.10 Speculative Decoding

Speculative decoding addresses a fundamental inefficiency inherent in autoregressive generation: because each token depends on all previous tokens through the attention mechanism, we cannot parallelize the generation of consecutive positions using the main model, leaving valuable GPU parallelism unexploited [Leviathan et al., 2023, Chen et al., 2023]. The standard approach to autoregressive generation computes tokens one at a time, running a full forward pass of the large model for each token, making complete use of the model's capacity but achieving only sequential throughput regardless of available parallel compute resources. Speculative decoding breaks this serialization bottleneck by using a small, fast "draft" model to generate multiple candidate tokens speculatively, exploiting parallelism in the draft model, and then using the large target model to verify these candidates in a single batched forward pass that processes all candidates simultaneously. When the draft model's predictions match what the large model would have produced through standard sequential decoding, we accept multiple tokens at once and advance by several positions; when predictions differ, we reject from the point of divergence and fall back to the large model's prediction for that position.

The sampling trajectory visualization in Figure 7.25 shows multiple speculative paths through the probability space, illustrating how the draft model explores possible futures that the target model will later evaluate. The draft model generates candidate sequences shown as branching colored paths, each representing a possible continuation of the current prefix based on the draft model's probability distribution, extending for several tokens into the future to create a tree of speculative continuations. The large target model then evaluates all these candidates simultaneously in a single forward pass by using appropriate causal attention masks, computing what it would have sampled at each position. For each path, we determine the longest prefix where the draft model's tokens match the target model's preferences according to a probabilistic acceptance criterion. This parallel verification is the key source of speedup: instead of running the large model k times for k tokens sequentially, we run it once to verify all k candidates, achieving up to k-fold speedup when the draft model's predictions are accepted. The draft-then-verify pipeline shown in Figure 7.26 operates in two phases: in the draft phase, the small fast draft model (perhaps 10-100x fewer parameters than the target) generates a candidate sequence of k tokens autoregressively; in the verify phase, the target model computes probabilities for all k positions simultaneously using a single forward pass with appropriate attention masking, accepting tokens that match its preferences according to a carefully designed probabilistic acceptance criterion and resampling from the rejected position onward when predictions differ, with acceptance rates of 70-90% achievable when the models are well-matched.

Speculative decoding provides practical speedups of 2-3x on real systems without changing the output distribution: accepted sequences are provably exactly what the target model would have generated through standard sequential decoding, maintaining perfect distributional equivalence with no quality degradation. This property distinguishes speculative decoding from approximation methods like quantization or distillation that trade quality for speed. The method is particularly effective when the draft model is a smaller version of the target model trained on the same data, or a model that has been distilled specifically to match the target's distribution, because such models achieve high acceptance rates with minimal distribution mismatch. The primary cost is running the draft model, which should be fast enough that its computational overhead is more than offset by the parallelism gains from accepting multiple tokens per target model forward pass. Speculative decoding has become increasingly important as large language models grow to hundreds of billions of parameters, offering a practical way to reduce inference latency and cost without sacrificing the quality that large models provide.

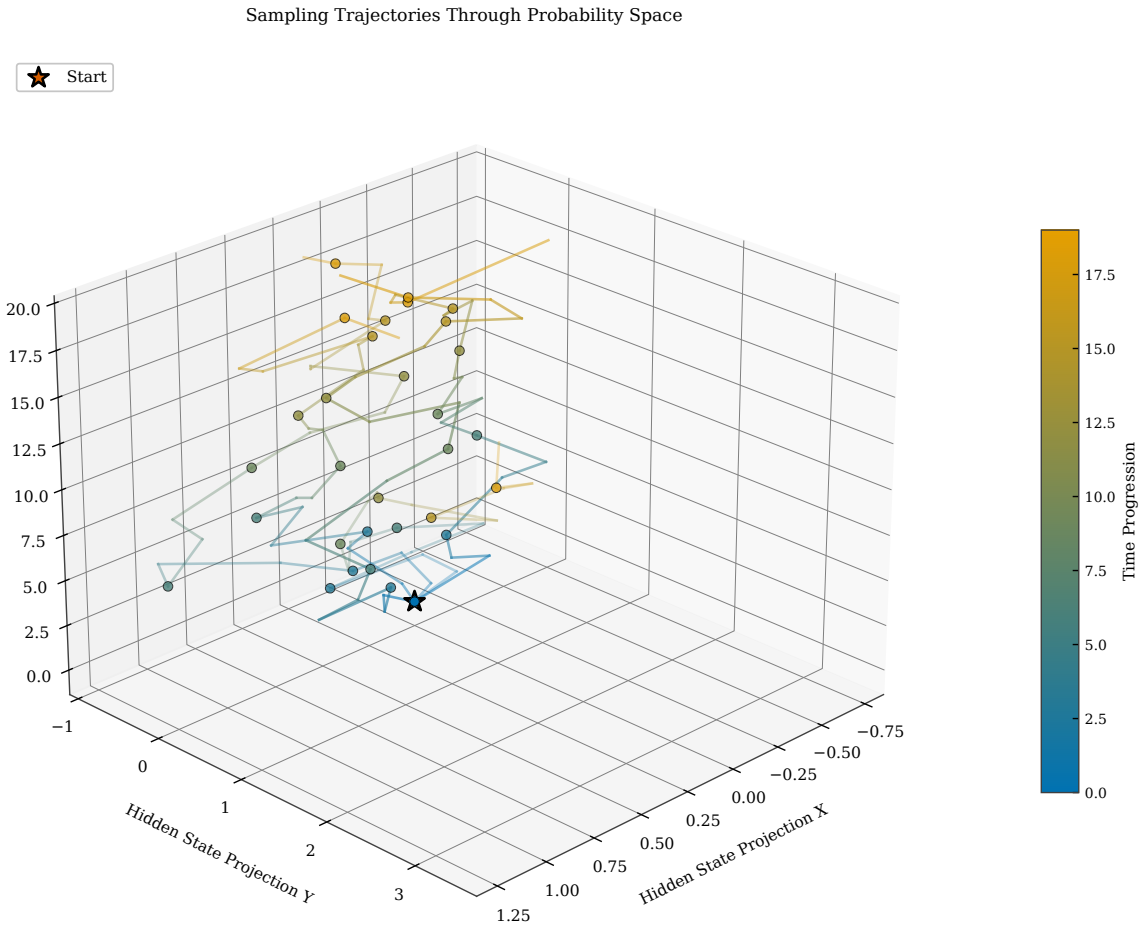

Figure 7.25: Speculative decoding trajectories in probability space. The draft model generates multiple candidate paths (colored trajectories). The large model verifies all paths in parallel, accepting matching prefixes. Accepted paths advance generation by multiple tokens at once.

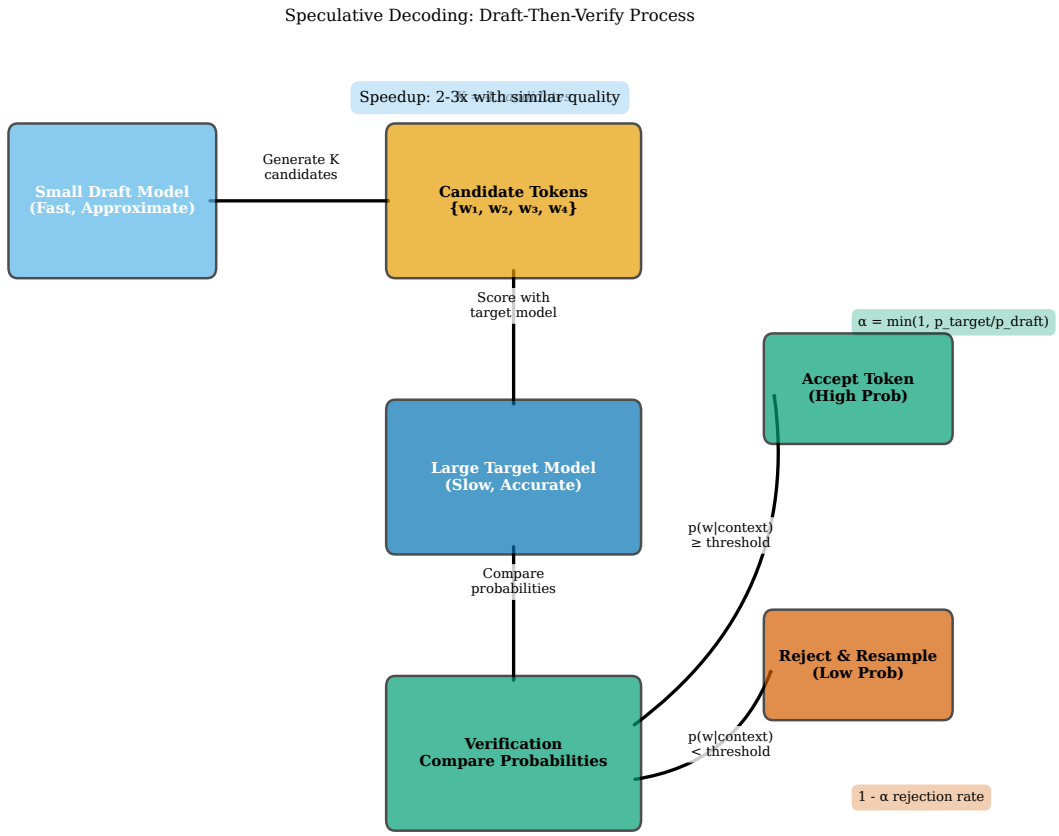

Figure 7.26: The speculative decoding pipeline. The draft model generates candidate tokens quickly. The target model verifies all candidates in parallel. Matching tokens are accepted; mismatches trigger resampling from the target model. Speedup depends on acceptance rate.

7.11 Context Representation in Decoding

Decoding strategies convert the transformer's rich context representations into discrete token sequences, but the efficiency and quality of this conversion depend critically on how context is accessed, maintained, and updated during the generation process. Chapter **??** explained how transformers represent context through the key-value (KV) cache: instead of recomputing attention over all previous positions from scratch at each step, we cache the key and value projection vectors and append only the new position's KV pair after each token is generated, reducing the complexity of each decoding step from $O(T^2)$ to $O(T)$ while maintaining access to the full context representation. The probability distribution at each decoding step reflects all context learned by the model through the entire training process and the specific content of the current generation: the KV cache encodes the complete prefix including both the user's prompt and all tokens generated so far, and the attention mechanism allows the current position to access any previous position with learned attention weights. Decoding strategies navigate this rich probability landscape using different algorithmic approaches: greedy decoding follows the steepest gradient of probability, always moving deterministically toward the mode; sampling methods explore the probability surface more broadly, visiting lower-probability regions with frequency proportional to their probability mass, enabling diversity and creativity; beam search maintains multiple paths simultaneously, hedging against locally optimal but globally suboptimal choices. Each strategy extracts different information from the same underlying context representation, producing characteristically different outputs from identical prompts and revealing the fundamental trade-offs between quality, diversity, and computational cost that practitioners must navigate.

Figure 7.27 visualizes the quality-diversity trade-off surface that different decoding strategies navigate, showing how the choice of decoding method determines which region of this trade-off space we occupy. Quality (measured by metrics like perplexity on held-out data, BLEU score against references, or human evaluation of coherence and fluency) and diversity (measured by metrics like self-BLEU across multiple generations, count of distinct n-grams, or entropy of the output distribution) are often in tension: strategies that maximize quality tend to produce repetitive, similar outputs across multiple runs, while strategies that maximize diversity risk incoherence or low fluency. Greedy decoding and beam search occupy the high-quality, low-diversity corner of this surface—they produce coherent text but always the same text. Pure sampling with high temperature occupies the high-diversity, variable-quality region—outputs differ across runs but quality is inconsistent. Nucleus sampling, top-k, and contrastive decoding find different trade-off points on this surface, with the optimal choice depending on whether the application values consistency or variety more.

How This Chapter Represents Context:

- The KV cache stores context as key-value pairs for efficient attention during generation
- Each decoding step produces a probability distribution reflecting the full context
- Decoding strategies navigate the probability landscape defined by the context representation
- Trade-offs between quality and diversity reflect different ways of exploiting context

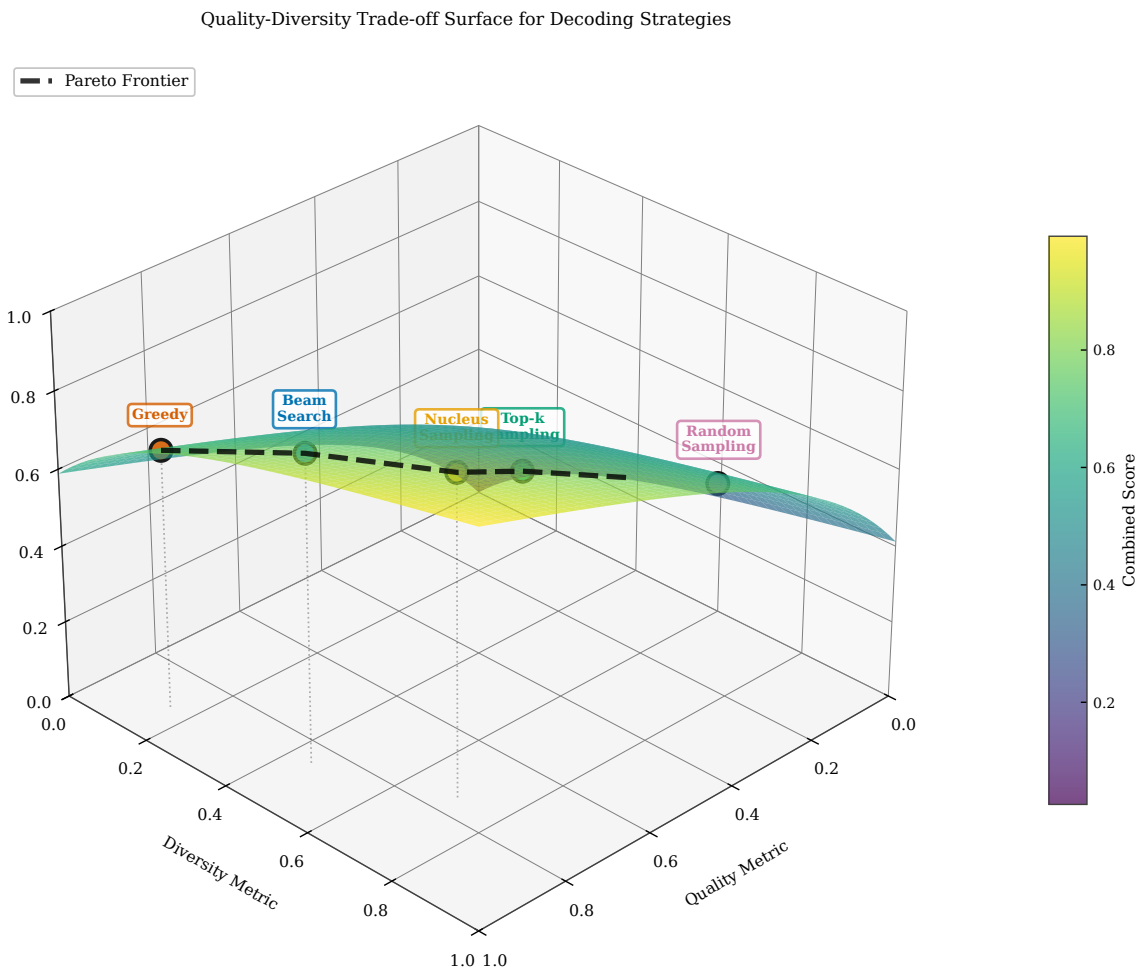

Figure 7.27: Quality-diversity trade-off surface for decoding strategies. Different strategies occupy different positions on this surface: greedy decoding maximizes quality but sacrifices diversity; pure sampling provides diversity but risks quality; nucleus and beam search find different trade-off points.

We can now predict better because:

- Temperature, top-k, and nucleus sampling convert probability distributions to diverse, high-quality text by controlling the quality-diversity trade-off
- Beam search finds globally coherent sequences by maintaining multiple hypotheses, particularly valuable for translation and summarization
- Contrastive decoding improves output quality by contrasting expert and amateur models, reducing degeneration and blandness
- Speculative decoding accelerates generation without changing output distribution, achieving 2-3x speedup through parallel verification

Next: Chapter **??** explores how we train models to produce probability distributions that decode well, examining loss functions, optimization algorithms, and the challenges of training at scale.

Exercises

1. Calculate the greedy decoding output for the first 5 tokens given a toy probability distribution where $P(\text{the}) = 0.3$, $P(\text{cat}) = 0.2$, and other tokens share the remaining probability uniformly. What potential issues might arise with this simple distribution?

2. Derive the relationship between temperature τ and the entropy of the softmax distribution for a vocabulary of size 2 with logits $z_1 = 1$ and $z_2 = 0$. Plot entropy as a function of temperature.

3. Compare top-k sampling with $k = 10$ to nucleus sampling with $p = 0.9$ on a peaked distribution where the top token has probability 0.8 and on a flat distribution where each of the top 100 tokens has probability 0.008. Which method adapts better to the distribution shape?

4. Implement the nucleus (top-p) sampling algorithm in pseudocode. Your algorithm should take a probability distribution and threshold p as input and return a sampled token.

5. Explain why length normalization is necessary in beam search. Calculate the raw and normalized scores (with $\alpha = 0.6$) for sequences of length 5 and length 10 that have the same per-token log-probability of -2.0.

6. For a beam search with beam width $B = 4$ and vocabulary size $|\mathcal{V}| = 10000$, how many candidate sequences are considered at each expansion step? How many survive after pruning?

7. The coverage penalty in beam search tracks attention over source positions. Describe a scenario where under-translation would occur without coverage penalty and explain how the penalty prevents it.

8. In contrastive decoding, what happens if the amateur model assigns probability 0 to a token that the expert model considers likely? How does the plausibility constraint address this issue?

9. Compare the computational costs of greedy decoding, beam search with $B = 5$, and speculative decoding with a draft model generating 4 candidate tokens. Express costs in terms of forward passes of the target model.

10. Design a constrained decoding task for generating product descriptions that must include specific feature keywords. What constraints would you impose, and how would grid beam search handle them?

11. * Derive the acceptance probability formula for speculative decoding that ensures the output distribution exactly matches standard sampling from the target model. Why is this distributional equivalence important?

12. * Analyze the failure mode of typical sampling where excluding high-probability tokens hurts coherence. Under what distribution conditions would typical sampling underperform nucleus sampling, and vice versa?

Bibliography

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. pages 889–898, 2018.

Chris Hokamp and Qun Liu. Lexically constrained decoding for sequence generation using grid beam search. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1535–1546, 2017.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2020.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. pages 19274–19286, 2023.

Xiang Lisa Li, Ari Holtzman, Daniel Fried, Percy Liang, Jason Eisner, Tatsunori Hashimoto, Luke Zettlemoyer, and Mike Lewis. Contrastive decoding: Open-ended text generation as optimization. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 12286–12312, 2023.

Ximing Lu, Sean Welleck, Peter West, Liwei Jiang, Jungo Kasai, Daniel Khashabi, Ronan Le Bras, Lianhui Qin, Youngjae Yu, Rowan Zellers, and Yejin Choi. Neurologic a*esque decoding: Constrained text generation with lookahead heuristics. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 780–799, 2022.

Clara Meister, Tiago Pimentel, Gian Wiher, and Ryan Cotterell. Locally typical sampling. *Transactions of the Association for Computational Linguistics*, 11:102–121, 2023.

Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. Modeling coverage for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 76–85, 2016.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Index