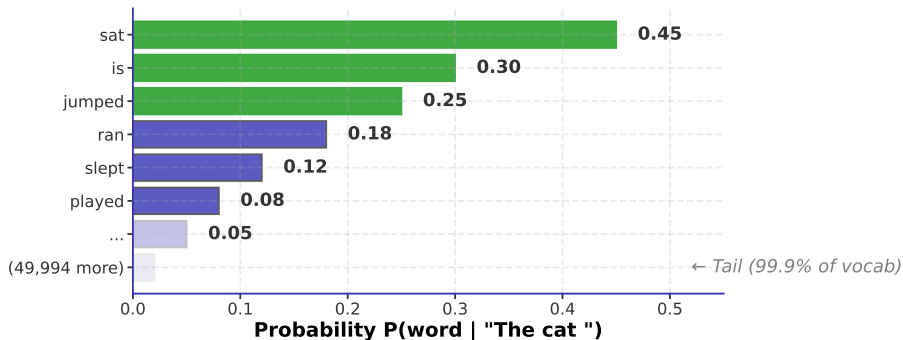


Decoding Strategies

Week 9: From Probabilities to Text

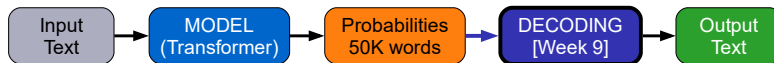
November 2025

The Decoding Challenge: Choose From 50,000 Words



The Question: Given these probabilities for “The cat __”, which word should we pick?

At each step, model outputs probability distribution over entire vocabulary - how do we choose?

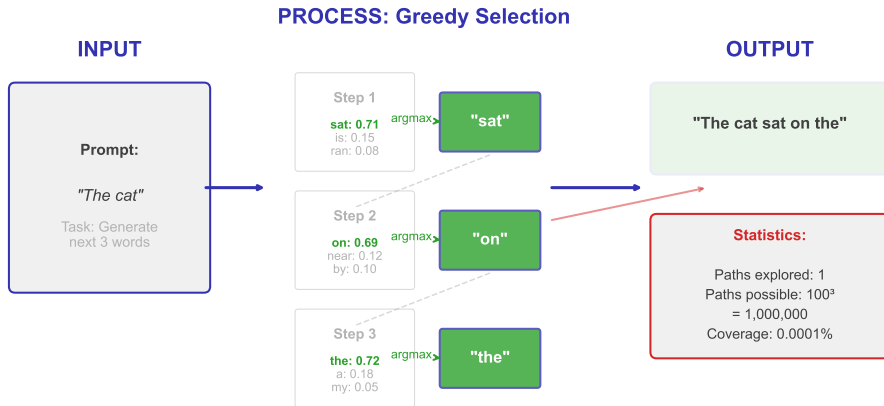


Our Journey:

1. We trained models (Weeks 3-7: RNN \rightarrow Transformers \rightarrow BERT/GPT)
2. They learned to predict: $P(\text{word}|\text{context})$
3. They output probability distributions over 50,000+ words
4. **Today:** How do we convert these probabilities into actual text?

Models predict probabilities. Decoding converts probabilities to text.

Extreme Case 1: Greedy Decoding (Too Narrow)



Extreme 1: Too narrow - misses 99.999999% of search space

What If We Explored More Paths?

Greedy chose: “The cat **sat**” ($P=0.68$)

But it ignored these alternatives:

“The cat walked ”	$P=0.12$	(might lead to better text)
“The cat jumped ”	$P=0.08$	(more interesting)
“The cat slept ”	$P=0.06$	(different story)
“The cat ran ”	$P=0.04$	(action-oriented)

Question: What if we kept ALL 100 words at each step?

Think: $100 \times 100 \times 100 \times 100 \times 100 = ?$

What If We Explored More Paths?

Greedy chose: "The cat **sat**" ($P=0.68$)

But it ignored these alternatives:

"The cat walked "	$P=0.12$	(might lead to better text)
"The cat jumped "	$P=0.08$	(more interesting)
"The cat slept "	$P=0.06$	(different story)
"The cat ran "	$P=0.04$	(action-oriented)

Question: What if we kept ALL 100 words at each step?

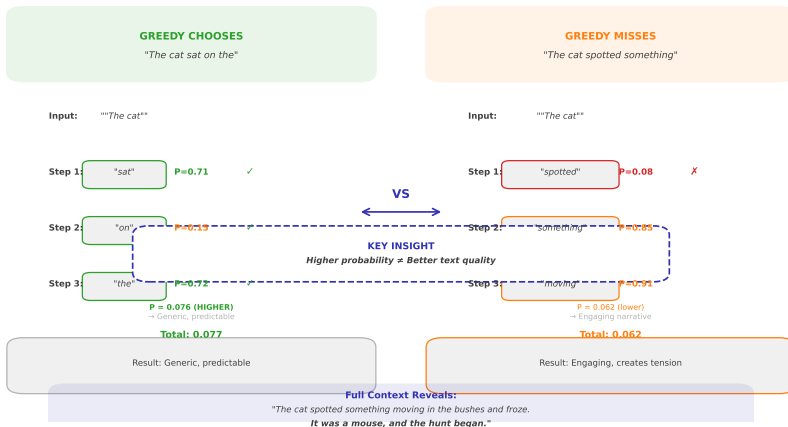
Think: $100 \times 100 \times 100 \times 100 \times 100 = ?$

Answer: 10 billion paths! Let's see what happens...

From 1 path to ALL paths - what could go wrong?

Greedy's Fatal Flaw: Missing Better Paths

Two Paths: Greedy Choice vs. Better Alternative



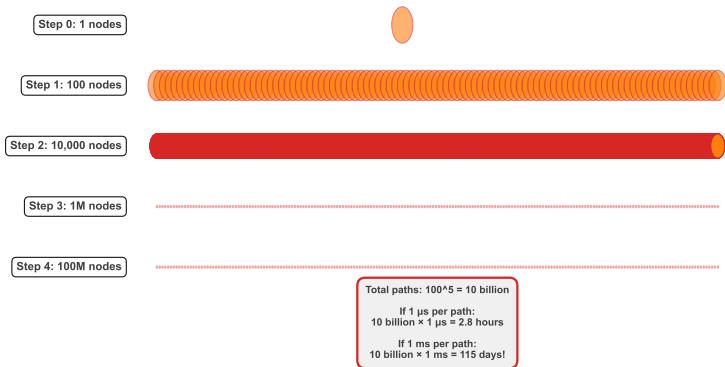
The Problem: High first-step probability \neq Best complete sentence

Greedy commits early, missing narratively richer paths despite lower initial probability

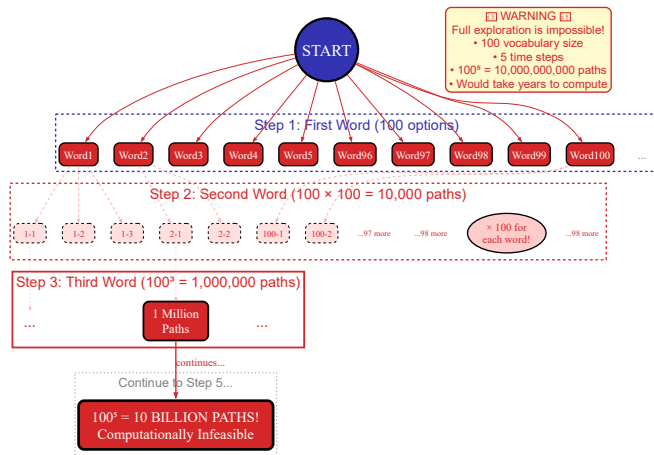
Extreme Case 2: Full Search Space (Too Broad)

Extreme Case 2: Full Search Space

Vocabulary size = 100, explore ALL paths



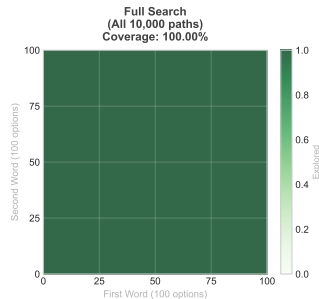
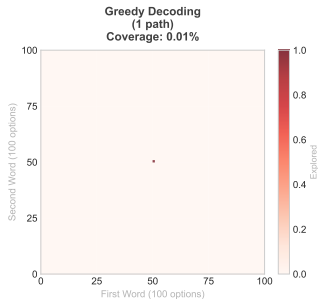
Full Exploration: From One Path to Billions



The Problem: How do we explore more than 1 but less than 10 billion paths?

The Extremes: Why Neither Works

The Extremes: Coverage Comparison
(Vocabulary=100, showing first 2 words only)



Greedy (0.01% coverage):

- Too narrow - misses better paths
- Fast but low quality
- Prone to repetition loops

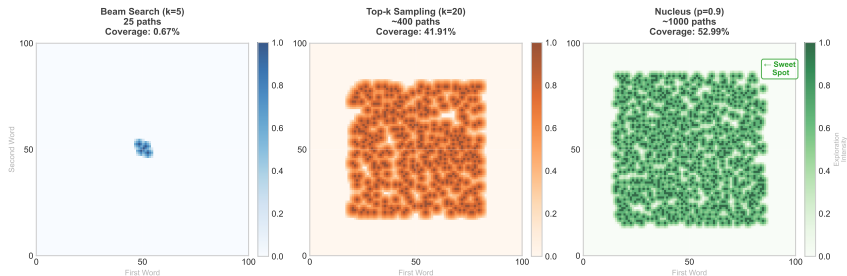
Full Search (100% coverage):

- Too broad - computationally infeasible
- Perfect in theory, impossible in practice
- Would take days/years to complete

Key Insight: We need methods that explore 1-5% of space intelligently

The Sweet Spot: Balanced Exploration

Practical Solutions: Balancing Coverage and Computation



The Solution: 1-5% Coverage

- **Not too narrow:** Explores enough paths to find good sequences
- **Not too broad:** Computationally feasible (seconds, not days)
- **Strategic exploration:** Focus on promising regions of search space

Coming Next: Learn 6 specific methods that achieve this balance

The sweet spot: Methods that intelligently explore 1-5% of the search space

Method 1: Greedy Decoding

Core Mechanism:

$$w_t = \operatorname{argmax}_{w \in V} P(w \mid w_1, \dots, w_{t-1})$$

At each step, pick the single word with highest probability

Characteristics:

- Deterministic (same input \rightarrow same output)
- Fast: $O(1)$ per step
- No exploration

Method 1 of 6: Greedy = always pick argmax

Method 2: Beam Search

Core Mechanism:

Maintain k hypotheses ("beams") at each step

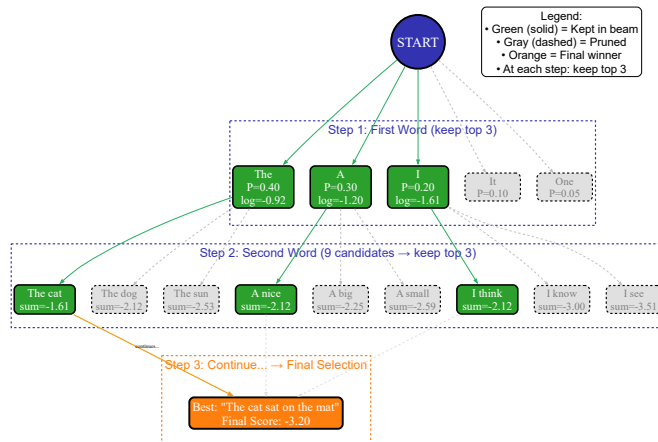
Expand each hypothesis, keep top- k by cumulative probability

Characteristics:

- Explores k paths simultaneously (typically $k=3-5$)
- Trade exploration vs computation
- Still deterministic for fixed k

Method 2 of 6: Beam = keep top- k paths

Beam Search: Step-by-Step Example



Worked example shows why beam search finds better sequences than greedy

Algorithm:

1. Start: Keep top-k tokens
2. Expand: Generate continuations for each
3. Score: Multiply probabilities
4. Prune: Keep top-k sequences
5. Repeat until END token

Scoring:

$$\text{score}(y_1 \dots y_t) = \prod_{i=1}^t P(y_i | y_{<i})$$

With length normalization:

$$\text{score} = \frac{1}{t} \sum_{i=1}^t \log P(y_i | y_{<i})$$

Best For:

- Machine translation
- Summarization
- Question answering
- Tasks with “correct” answer

Parameters:

Width = 3-5 (translation)

Width = 10 (diverse outputs)

Tradeoffs:

- + Better quality than greedy
- + Diverse hypotheses
- Still deterministic
- 4-5× slower than greedy

Beam search is the workhorse for deterministic tasks

Method 3: Temperature Sampling

Core Mechanism:

$$P_T(w_i) = \frac{\exp(z_i / T)}{\sum_j \exp(z_j / T)}$$

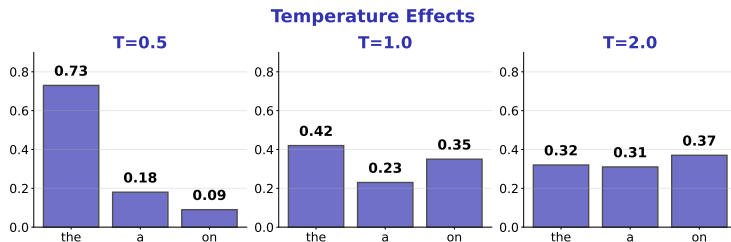
Reshape probability distribution with temperature T , then sample

Characteristics:

- $T \downarrow 1$: More focused (sharper distribution)
- $T \uparrow 1$: More random (flatter distribution)
- Stochastic: different output each time

Method 3 of 6: Temperature = control randomness

Temperature Sampling: Control Randomness



Key Insight: Temperature reshapes probability distribution

$T < 1$: more focused. $T = 1$: unchanged. $T > 1$: more random

Temperature: Step-by-Step Calculation

Given: Logits = [2.0, 1.0, 0.5, 0.2]

Tokens = ["cat", "dog", "bird", "fish"]

$T=0.5$: [4.0, 2.0, 1.0, 0.4] \rightarrow [0.73, 0.18, 0.07, 0.02]

\rightarrow **73% on "cat" (FOCUSED)**

$T=1.0$: [2.0, 1.0, 0.5, 0.2] \rightarrow [0.42, 0.23, 0.16, 0.13]

\rightarrow **42% on "cat" (BALANCED)**

$T=2.0$: [1.0, 0.5, 0.25, 0.1] \rightarrow [0.32, 0.26, 0.23, 0.19]

\rightarrow **32% on "cat" (FLAT)**

Lower T = more peaked | Higher T = more flat

Concrete numbers show how temperature scaling works

Method 4: Top-k Sampling

Core Mechanism:

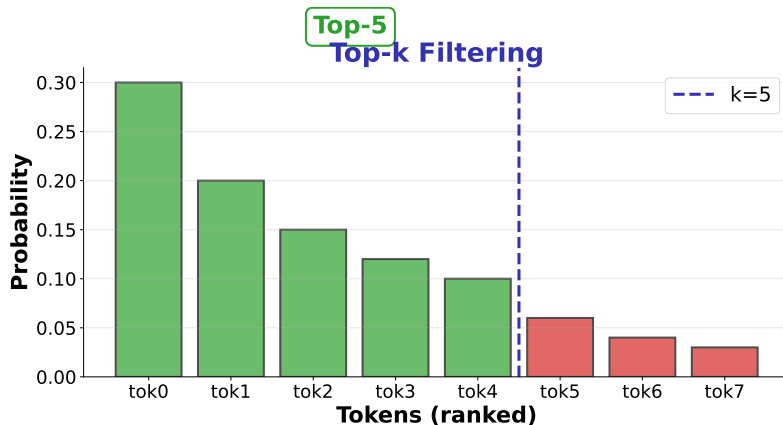
1. Sort words by probability
2. Keep only top k words (e.g., $k=50$)
3. Renormalize and sample from these k

Characteristics:

- Filters out low-probability “junk” words
- Fixed cutoff (always k words)
- Can combine with temperature

Method 4 of 6: Top-k = filter then sample

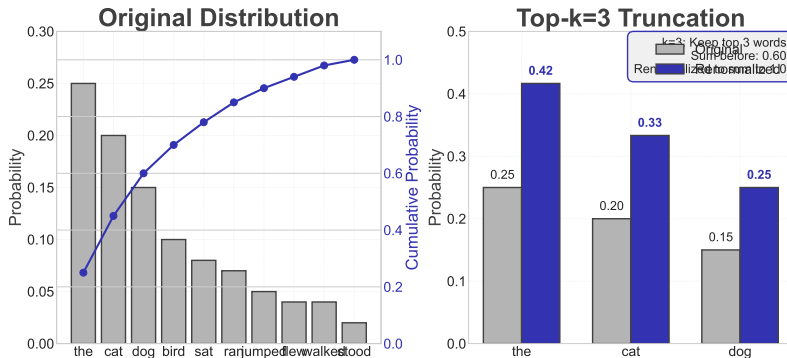
Top-k Sampling: Filter the Tail



Key Insight: Only sample from top-k most likely tokens

Prevents sampling from long tail of unlikely words

Top-k Sampling: Numerical Example



Concrete numbers show k=50 filtering process

Method 5: Nucleus (Top-p) Sampling

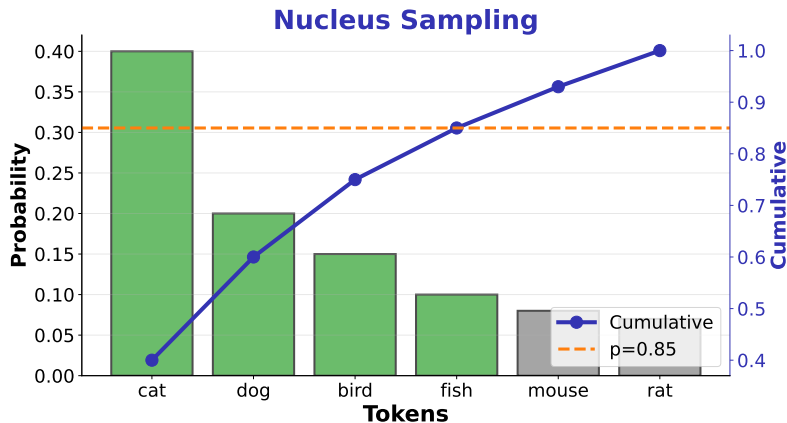
Core Mechanism:

1. Sort words by probability
2. Keep minimum set where cumulative probability $\geq p$
3. Sample from this set

Characteristics:

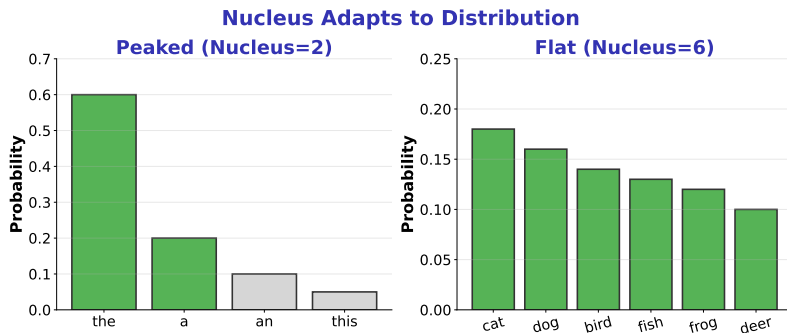
- Adaptive: number of words varies
- Focuses on “nucleus” of probability mass (typically $p=0.9$)
- Adjusts to distribution shape

Method 5 of 6: Nucleus = adaptive probability mass



Key Insight: Adapt vocabulary size to distribution shape

Nucleus size grows/shrinks based on probability spread



Same p value gives different vocabulary sizes for peaked vs flat distributions

Method 6: Contrastive Search

Core Mechanism:

Choose word that maximizes:

$$\text{score} = (1 - \alpha) \cdot \text{model probability} - \alpha \cdot \text{similarity to previous}$$

Penalize words similar to already-generated text

Characteristics:

- Explicitly avoids repetition
- Balances coherence and diversity
- Deterministic with hyperparameter α

Method 6 of 6: Contrastive = penalize repetition

Greedy Decoding Problem

"The city is a major city.
The city has many attractions.
The city is known..."

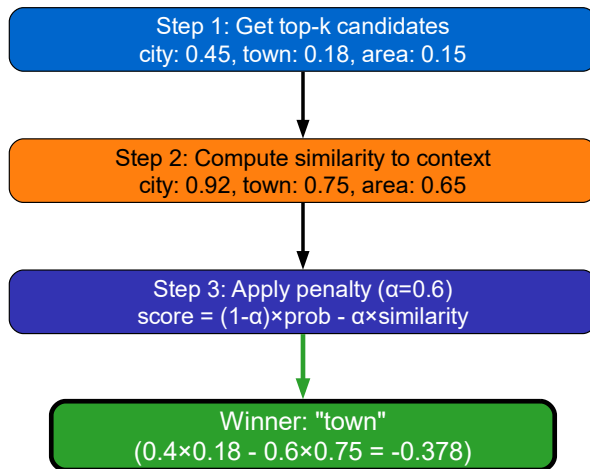
"the city" appears 3 times!



Solution: Contrastive Search

Discovery Question: Why do models repeat themselves?

Greedy and beam search maximize probability - but high probability = repeating recent context



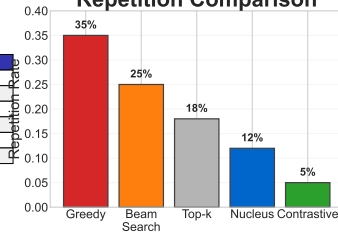
Key Insight: Balance probability with diversity penalty

Contrastive Search vs Nucleus Sampling

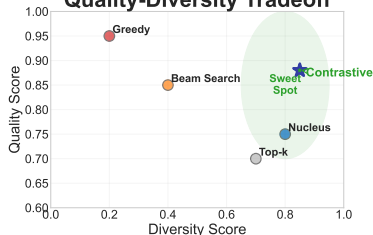
Method Comparison

Aspect	Nucleus (Top-p)	Contrastive
Diversity	High	High
Repetition	Medium	Very Low
Coherence	Good	Excellent
Speed	Fast $O(V \log V)$	Slower $O(k \times T^2)$
Parameters	p , temperature	α , k , penalty
Best for	General use	Long generation

Repetition Comparison



Quality-Diversity Tradeoff



Context: "The weather today is"

Nucleus Sampling:

- beautiful and sunny with clear skies
- perfect for a walk in the park

Contrastive Search:

- beautiful with clear blue skies
- perfect for outdoor activities today

Contrastive adds explicit similarity penalty that Nucleus lacks

Checkpoint Quiz 1: Match the Method

Methods:

1. Greedy
2. Beam Search
3. Temperature
4. Top-k
5. Nucleus
6. Contrastive

Match to Mechanisms:

- A. Sample from reshaped distribution
- B. Keep top-k paths at each step
- C. Always pick argmax
- D. Filter to k words, then sample
- E. Penalize similarity to previous
- F. Adaptive probability mass cutoff

Checkpoint Quiz 1: Match the Method

Methods:

1. Greedy
2. Beam Search
3. Temperature
4. Top-k
5. Nucleus
6. Contrastive

Match to Mechanisms:

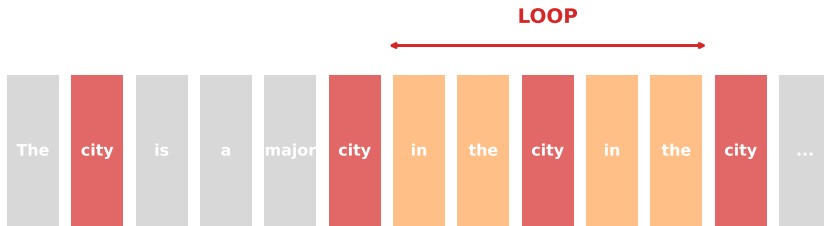
- A. Sample from reshaped distribution
- B. Keep top-k paths at each step
- C. Always pick argmax
- D. Filter to k words, then sample
- E. Penalize similarity to previous
- F. Adaptive probability mass cutoff

Answers: 1→C, 2→B, 3→A, 4→D, 5→F, 6→E

Now you know the toolbox - let's see WHY each tool exists!

Quiz 1: Can you match each method to its mechanism?

Greedy Decoding Gets Stuck



Output: "The city is a major city in the city in the city..."

Greedy's Problem: Trapped in loops, can't escape

Why Beam Helps: Explores $k=3-5$ paths, avoids greedy trap

Problem 1 of 6: Greedy decoding creates loops → Beam search explores alternatives

High Temperature Creates Nonsense

The	glorp	is	very	blorptastic	
She	likes	to	eat	qwerty	food
I	went	to	the	flurb	yesterday
The	weather	is	zxqp	today	

Generated words not in vocabulary!

Greedy & Beam's Problem: Same input → same output always

Why Temperature Helps: Sampling introduces randomness, enables creativity

Problem 2 of 6: Deterministic methods lack variation → Temperature adds controlled randomness

Zero Creativity: Always Same Output

#9:	The weather is nice today.	#10:	The weather is nice today.
#7:	The weather is nice today.	#8:	The weather is nice today.
#5:	The weather is nice today.	#6:	The weather is nice today.
#3:	The weather is nice today.	#4:	The weather is nice today.
#1:	The weather is nice today.	#2:	The weather is nice today.

100x

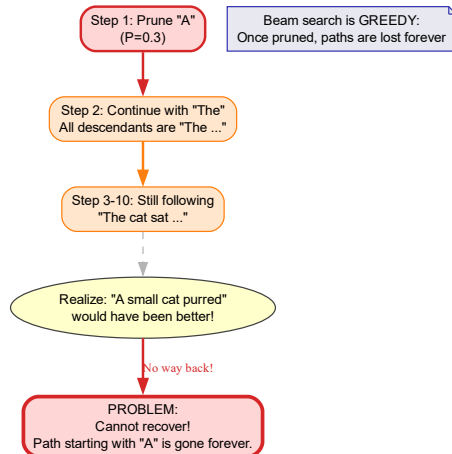
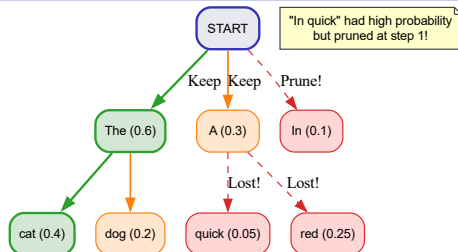
Asked 100 times → Always: "The weather is nice today."

Temperature's Problem: Pure sampling includes low-quality words

Why Top-k Helps: Filter to $k=50$ best words, then sample

Problem 3 of 6: Can't balance quality & creativity → Top-k filters unlikely words

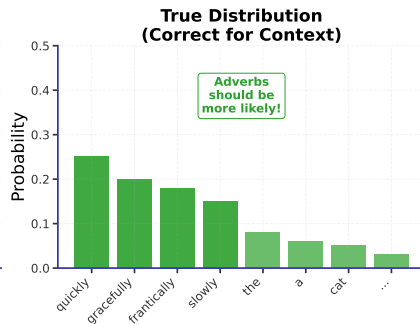
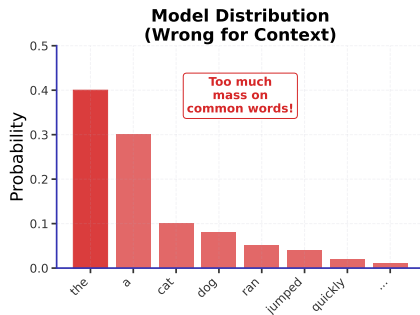
Beam Search Limitation: Missing Better Paths



Problem 4 of 6: Even beam search prunes early, cannot recover optimal path - 4 perspectives

Why Nucleus? Problem: Distribution Tail Contains Junk

Context: "The cat ran ____"

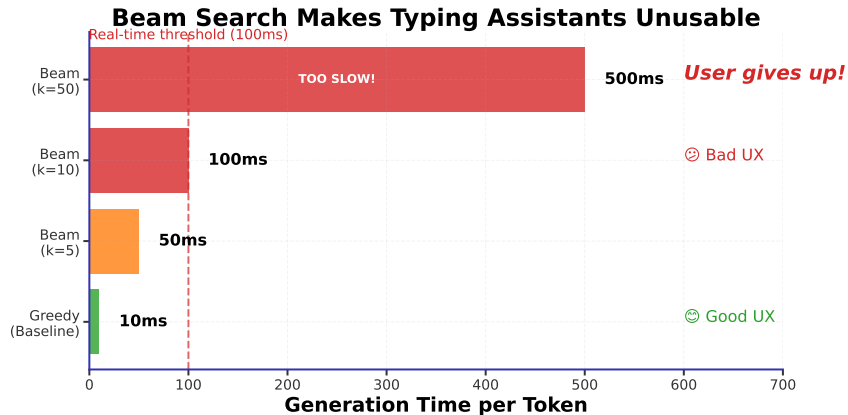


Top-k's Problem: Fixed k doesn't adapt to distribution shape

Why Nucleus Helps: Adaptive cutoff at $p=0.9$ probability mass

Problem 5 of 6: Tail contains junk → Nucleus adapts to distribution

Why Contrastive? Problem: Generic Repetitive Text

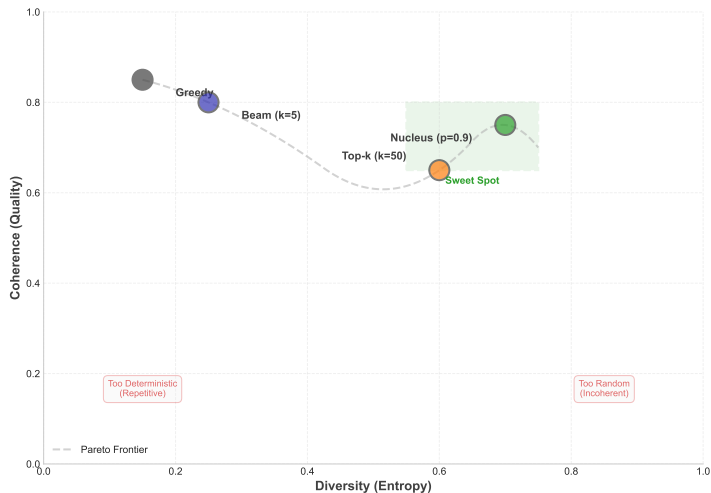


All Methods' Problem: Can still produce generic, repetitive text

Why Contrastive Helps: Explicitly penalizes similarity to previous tokens

Problem 6 of 6: Generic text persists → Contrastive reduces repetition

The Quality-Diversity Tradeoff



Key Insight: We saw problems and their solutions. Each method balances quality vs diversity.

All 6 problems relate to balancing coherence with creativity

Checkpoint Quiz 2: Which Method for Which Problem?

Match Solution to Problem:

1. Beam Search → ?
2. Temperature → ?
3. Top-k → ?
4. Nucleus → ?
5. Contrastive → ?

Problems to Solve:

- A. Too boring OR too crazy
- B. Repetition loops
- C. No diversity
- D. Fixed k doesn't adapt
- E. Generic repetitive text

Checkpoint Quiz 2: Which Method for Which Problem?

Match Solution to Problem:

1. Beam Search → ?
2. Temperature → ?
3. Top-k → ?
4. Nucleus → ?
5. Contrastive → ?

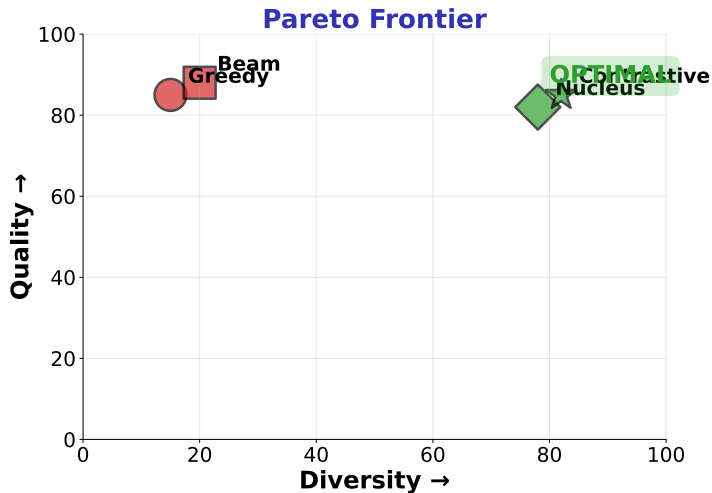
Problems to Solve:

- A. Too boring OR too crazy
- B. Repetition loops
- C. No diversity
- D. Fixed k doesn't adapt
- E. Generic repetitive text

Answers: 1→B, 2→C, 3→A, 4→D, 5→E

Each method targets a specific failure mode!

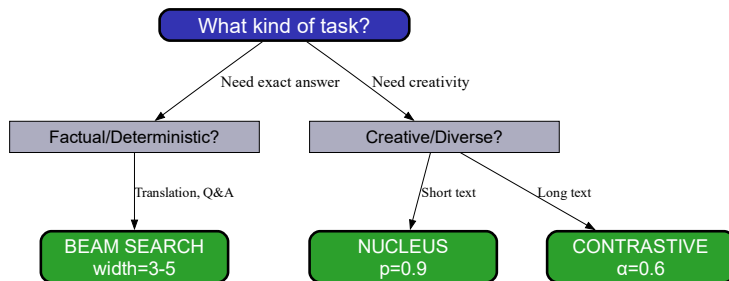
Quiz 2: Understanding the method-problem mapping



Pareto Frontier: No method dominates all others

Choose based on task: deterministic tasks (left), creative tasks (right)

Choosing the Right Method: Decision Tree



Start with task requirements, follow tree to recommended method

Task-Specific Recommendations

Task	Method	Parameters
Translation	Beam	$w=3-5$
Factual QA	Greedy	$T=0.3$
Code	Greedy	$T=0$
Dialogue	Nucleus	$p=0.9$
Creative	Nucleus	$p=0.95$
Long Stories	Contrastive	$\alpha=0.6$

Comprehensive mapping from 8 common tasks to optimal decoding strategies

Checkpoint Quiz 3: Choose the Right Method

Given these tasks, which method would you use?

1. Medical report summary

- Needs: Accuracy, no hallucination

2. Creative story writing

- Needs: Diversity, creativity

3. Code generation

- Needs: Correctness, explore options

4. Customer service chat

- Needs: Natural, varied responses

5. Legal document

- Needs: Precise, formal

6. Long blog post

- Needs: Coherent, no repetition

Checkpoint Quiz 3: Choose the Right Method

Given these tasks, which method would you use?

1. **Medical report summary**

- Needs: Accuracy, no hallucination

2. **Creative story writing**

- Needs: Diversity, creativity

3. **Code generation**

- Needs: Correctness, explore options

4. **Customer service chat**

- Needs: Natural, varied responses

5. **Legal document**

- Needs: Precise, formal

6. **Long blog post**

- Needs: Coherent, no repetition

Answers:

1. Greedy/Low temp ($T=0.1-0.3$) 2. Nucleus ($p=0.95$, $T=1.0$) 3. Beam Search ($k=3-5$)
4. Nucleus ($p=0.9$, $T=0.7$) 5. Greedy ($T=0$) 6. Contrastive ($\alpha=0.6$)

Quiz 3: Real-world task selection is crucial for quality

Key Takeaways

1. **6 Problems** → **6 Solutions**: Each method solves specific failure mode
2. **Deterministic** (Greedy, Beam): High quality, no diversity - factual tasks
3. **Stochastic** (Temperature, Top-k, Nucleus): Diverse but variable quality
4. **Balanced** (Contrastive): Explicit degeneration prevention
5. **Task matters**: Translation → Beam — Dialogue → Nucleus — Stories → Contrastive
6. **Tradeoffs**: Speed vs Quality, Diversity vs Coherence

Modern Standard: Nucleus (top-p=0.9) + Temperature (T=0.7) for most applications

Next: Lab - Implement all 6 methods, measure quality-diversity tradeoffs

Decoding strategy matters as much as model architecture



What We Learned:

- Models give us probability distributions (Week 3-7)
- Converting to text has 6 fundamental challenges
- Each decoding method addresses specific problems
- No universal best - choose based on task requirements
- Production systems use hybrid methods (Nucleus + Temperature)

Complete pipeline from model training to text generation

Technical Appendix

25 slides: Complete mathematical treatment

A1-A5: Beam Search Mathematics

A6-A10: Sampling Mathematics

A11-A14: Contrastive Search & Degeneration

A15-A19: Advanced Topics & Production

A20-A25: The 6 Problems - Technical Analysis (NEW)

A1: Beam Search Formulation

Objective: Find sequence $y^* = \operatorname{argmax} P(y|x)$

Decomposition:

$$P(y|x) = \prod_{t=1}^T P(y_t|y_{<t}, x)$$

Log-probability (more stable):

$$\log P(y|x) = \sum_{t=1}^T \log P(y_t|y_{<t}, x)$$

Beam Search Approximation:

Instead of exploring all V^T sequences, maintain top-k hypotheses at each step

Complexity:

Time: $O(k \cdot V \cdot T)$ where k = beam width, V = vocabulary, T = length

Space: $O(k \cdot T)$ to store hypotheses

Beam search is tractable approximation to exact search

A2: Length Normalization

Problem: Longer sequences have lower probabilities (more terms multiplied)

$$P(y_1, y_2, y_3, y_4) = \underbrace{0.5}_{y_1} \times \underbrace{0.5}_{y_2} \times \underbrace{0.5}_{y_3} \times \underbrace{0.5}_{y_4} = 0.0625$$

$$P(y_1, y_2) = 0.5 \times 0.5 = 0.25 > 0.0625$$

Bias toward shorter sequences!

Solution: Length normalization

$$\text{score}(y) = \frac{1}{|y|^\alpha} \log P(y)$$

where $\alpha \in [0.5, 1.0]$ (typically 0.6-0.7)

Effect:

Without: Beam search heavily biases toward short outputs

With: Fair comparison across different lengths

Length normalization is essential for beam search quality

A3: Beam Search Variants

Diverse Beam Search:

Partition beams into groups
Penalize within-group similarity
Result: More diverse hypotheses

Constrained Beam Search:

Force certain tokens to appear
Useful for: Keywords, entities
Applications: Controllable generation

Stochastic Beam Search:

Sample beams instead of argmax
Combines beam + sampling
More diverse than standard beam

Block n-gram Beam:

Penalize n-gram repetition
Prevents “the city is a city” loops
Common in summarization

Many beam search variants exist for specific requirements

A4: Beam Search Stopping Criteria

When to stop expanding beams?

Method 1: Fixed length

Stop at T_{\max} tokens (simple but rigid)

Method 2: END token

Stop when beam generates special token (most common)

Method 3: Score threshold

Stop when best score cannot improve enough

$$\frac{\text{best_incomplete}}{\text{best_complete}} < \text{threshold}$$

Method 4: Timeout

Computational budget exceeded (production systems)

Choice of stopping criterion affects output length distribution

A5: Beam Search Limitations

Fundamental Issues:

1. **Exposure bias:** Trained with teacher forcing, tested with own outputs
2. **Label bias:** Cannot compare sequences of different prefixes fairly
3. **Repetition:** Still can loop (“the city is a major city”)
4. **Bland outputs:** Maximizes probability, not interestingness
5. **Search errors:** May miss better sequences outside beam

When Beam Search Fails:

Open-ended generation (dialogue, stories)

Long-form text (repetition accumulates)

Creative tasks (probability \neq quality)

→ Need sampling-based methods

Beam search optimizes wrong objective for creative tasks

A6: Sampling as Inference

Goal: Sample $y \sim P(y|x)$ instead of $\operatorname{argmax} P(y|x)$

Ancestral Sampling:

For $t = 1$ to T :

 Compute $P(y_t|y_{<t}, x)$

 Sample $y_t \sim P(\cdot|y_{<t}, x)$

Properties:

Stochastic: Different output each time

Explores full distribution (in expectation)

Can generate low-probability sequences

Variants:

Temperature: Reshape distribution before sampling

Top-k: Truncate distribution before sampling

Nucleus: Dynamic truncation before sampling

Sampling enables diversity but loses quality guarantees

Softmax with Temperature:

$$p_i(T) = \frac{\exp(z_i/T)}{\sum_{j=1}^V \exp(z_j/T)}$$

Limiting Cases:

$$T \rightarrow 0: p_i \rightarrow \begin{cases} 1 & \text{if } i = \operatorname{argmax} z \\ 0 & \text{otherwise} \end{cases} \quad (\text{greedy})$$

$$T \rightarrow \infty: p_i \rightarrow 1/V \quad (\text{uniform})$$

Entropy Analysis:

Entropy $H(p) = -\sum p_i \log p_i$ measures randomness

H increases monotonically with T

Low T (<0.5): $H \approx 0$ (deterministic)

High T (>2.0): $H \approx \log V$ (maximum entropy)

Temperature provides continuous control over distribution entropy

Formal Definition:

Let σ = permutation sorting probabilities descending

$$V_k = \{w_{\sigma(1)}, w_{\sigma(2)}, \dots, w_{\sigma(k)}\}$$

Truncated distribution:

$$p'(w) = \begin{cases} \frac{p(w)}{\sum_{w' \in V_k} p(w')} & \text{if } w \in V_k \\ 0 & \text{otherwise} \end{cases}$$

Information Loss:

Original entropy: $H(p) = -\sum_{i=1}^V p_i \log p_i$

After top-k: $H(p') = -\sum_{i=1}^k p'_i \log p'_i < H(p)$

Loss $\approx \sum_{i=k+1}^V p_i \log(1/p_i)$ (tail information)

Top-k sacrifices tail probability mass for sampling quality

A9: Nucleus (Top-p) Mathematics

Formal Definition:

$$V_p = \min \left\{ V' \subseteq V : \sum_{w \in V'} p(w) \geq p \right\}$$

Smallest set with cumulative mass $\geq p$

Dynamic Vocabulary Size:

$$|V_p| = \min \left\{ k : \sum_{i=1}^k p_{\sigma(i)} \geq p \right\}$$

Adapts to distribution shape:

Peaked: Small $|V_p|$ (2-5 tokens)

Flat: Large $|V_p|$ (50+ tokens)

Why Nucleus > Top-k:

Top-k: Fixed k regardless of $p(w)$ distribution

Nucleus: Adapts k to achieve consistent probability mass

Nucleus automatically adjusts vocabulary to distribution characteristics

A10: Sampling Quality Metrics

Quality Metrics:

Perplexity: $\exp(-\frac{1}{T} \sum \log p(y_t))$
Lower = better

BLEU (translation):

N-gram overlap with reference
0-100 scale

Human evaluation:

Fluency (1-5)
Relevance (1-5)

Diversity Metrics:

Distinct-n: $\frac{\text{unique n-grams}}{\text{total n-grams}}$
Higher = more diverse

Self-BLEU:

BLEU of output vs other outputs
Lower = more diverse

Repetition Rate:

$\frac{\text{repeated n-grams}}{\text{total n-grams}}$
Lower = less repetitive

Need both quality AND diversity metrics to evaluate decoding

A11: The Degeneration Problem (Formal)

Definition: Model-generated text with unnatural repetitions

Why It Happens:

1. Model trained on natural text (low repetition)
2. But generation maximizes $P(y_t|y_{<t})$
3. Recent context $y_{<t}$ influences P
4. Creates positive feedback: high prob word \rightarrow context \rightarrow same high prob word

Quantifying Degeneration:

Repetition rate in greedy: 15-30% (depending on domain)

Repetition rate in human text: 2-5%

Gap = degeneration problem

Examples:

"The city is a major city in the United States. The city..."

"I think that I think that I think..."

Maximizing probability does not equal natural text

A12: Contrastive Search Objective

Scoring Function:

$$\text{score}(w_t) = (1 - \alpha) \times \underbrace{P(w_t | y_{<t})}_{\text{model confidence}} - \alpha \times \underbrace{\max_{w_i \in y_{<t}} \text{sim}(w_t, w_i)}_{\text{context similarity}}$$

where $\alpha \in [0, 1]$ controls tradeoff

Similarity Function:

$$\text{sim}(w_i, w_j) = \frac{h_i \cdot h_j}{||h_i|| \cdot ||h_j||}$$

(cosine similarity)
using token embeddings h

Algorithm:

1. Get top-k candidates by probability
2. For each candidate, compute similarity to all tokens in $y_{<t}$
3. Apply penalty: $\text{score} = \text{prob} - \alpha \times \text{max_similarity}$
4. Select candidate with highest score

Contrastive search explicitly penalizes copying recent context

A13: Contrastive Search Parameters

Alpha (α):

$\alpha = 0$: Pure greedy (no penalty)
 $\alpha = 0.6$: Balanced (recommended)
 $\alpha = 1.0$: Maximum diversity (risky)

Typical Settings:

Short text (<100 tokens): $\alpha = 0.4 - 0.5$
Medium (<500): $\alpha = 0.5 - 0.6$
Long (500+): $\alpha = 0.6 - 0.7$

Top-k for Candidates:

$k = 4$: Fast, focused
 $k = 6$: Balanced (default)
 $k = 10$: Diverse

Computational Cost:

For each step:

- Compute similarities: $O(k \times t)$
- t grows with generation

Total: $O(k \times T^2)$

12× slower than greedy

Hugging Face default: $\alpha=0.6$, $k=4$

A14: Degeneration Analysis

Research Findings (2024-2025):

- Greedy decoding repetition: 18-25% (GPT-2), 12-18% (GPT-3)
- Nucleus sampling repetition: 8-12% (still above human 3-5%)
- Contrastive search repetition: 4-7% (closest to human)

Why Probability Maximization Fails:

Training objective: Next token prediction

But generation requires: Global coherence

Mismatch: Local optimum \neq global quality

Solutions Hierarchy:

1. Temperature/Top-k/Nucleus: Reduce greedy's determinism
2. Contrastive: Explicit degeneration penalty
3. RLHF/DPO: Align model with human preferences (different lecture)

Contrastive search addresses fundamental limitation of likelihood-based decoding

Combining Strategies:

Nucleus + Temperature:

Apply temperature THEN nucleus

$$p_i(T) = \text{softmax}(z/T), \quad \text{then} \quad V_p \leftarrow \text{nucleus}(p_i(T))$$

Used by GPT-3 API, ChatGPT

Beam + Sampling:

Beam search with stochastic selection

Keep top-k, sample from them (not argmax)

Contrastive + Nucleus:

Nucleus for candidate generation

Contrastive scoring for selection

Best of both worlds

Hybrid methods leverage complementary strengths

A16: Constrained Decoding (2025)

Goal: Force certain tokens/patterns to appear

Lexically Constrained:

Must include keywords: { "AI", "ethics", "safety" }

Beam search variant: Track constraint satisfaction

Format Constraints:

JSON output: Force structure { "key": "value" }

Code: Force syntactic validity

NeuroLogic Decoding (2021):

Beam search + constraint satisfaction

Optimal for: Keyword-based generation

Production Use Cases:

Structured data extraction (force JSON)

Controllable summarization (force keywords)

Code generation (force syntax)

Constrained decoding enables controllable generation

A17: Computational Complexity Comparison

Method	Time per token	Total complexity	Relative speed
Greedy	$O(V)$	$O(V \times T)$	1.0× (baseline)
Temperature	$O(V)$	$O(V \times T)$	1.1× (softmax overhead)
Top-k	$O(V)$	$O(V \times T)$	1.2× (sorting)
Nucleus	$O(V \log V)$	$O(V \log V \times T)$	1.3× (sort + cumsum)
Beam (k=5)	$O(k \times V)$	$O(k \times V \times T)$	4.5× (k=5)
Contrastive	$O(k \times T)$	$O(k \times T^2)$	12× (similarity)

Key Insight: Contrastive's T^2 term makes it expensive for long sequences

Practical Impact (1000-token generation):

Greedy: 2.5 seconds

Nucleus: 3.2 seconds (best choice)

Beam: 11 seconds

Contrastive: 30 seconds (only if quality critical)

Computational cost matters for production deployment

Production System Settings (2025)

Default decoding parameters used by major LLM APIs and platforms

System/API	Default Method	Temperature	Top-p	Top-k	Other Parameters	Notes
GPT-4 API	Nucleus	0.7	1.0	—	frequency_penalty=0 presence_penalty=0	Can adjust all params
Claude API	Nucleus	1.0	0.999	—	max_tokens required	Temperature $\in [0,1]$
ChatGPT Web	Nucleus+Temp	0.7	0.95	—	Not adjustable	Optimized for chat
Gemini API	Top-k + Top-p	1.0	0.95	40	candidate_count=1	Both k and p used
Llama 2 (HF)	Configurable	1.0	0.9	50	repetition_penalty=1.0	Full control
Cohere API	Nucleus	0.75	0.999	0	frequency_penalty=0	k=0 means disabled
Mistral API	Nucleus	0.7	1.0	—	safe_mode=false	Similar to OpenAI
Together AI	Configurable	0.7	0.7	50	repetition_penalty=1.0	Multiple options

- Temperature: Lower = more deterministic, Higher = more creative
- Top-p (Nucleus): Cumulative probability threshold (typically 0.9-1.0)
- Top-k: Number of top tokens to consider (often 40-50 when used)
- Most production systems use Nucleus sampling as default
- ChatGPT and Claude optimize for conversational quality
- APIs generally allow full parameter customization

Provider Types:

- Proprietary LLM
- Open Source
- Optimized for Chat

What ChatGPT, Claude, and other production systems actually use

Active Research Areas (2025):

1. **Quality-diversity optimization:** Multi-objective search methods
2. **Learned decoding:** Train models to decode better (RLHF, DPO)
3. **Speculative decoding:** Parallel generation for speed ($4-8\times$ faster)
4. **Adaptive methods:** Choose strategy dynamically during generation
5. **Energy-based decoding:** Score sequences globally (not token-by-token)

Open Problems:

How to automatically select best T , p , k , α for new task?

How to balance fluency + factuality + creativity simultaneously?

How to decode efficiently for 100K+ token outputs?

Trend: Moving from hand-tuned parameters to learned decoding strategies

Decoding is an active research area with many open questions