Decoding Strategies

Choosing the Next Word: Accuracy vs Creativity

Week 9: Natural Language Processing Course

Why Do Some Systems Always Give the Same Answer?

Google Autocomplete:

``The weather is__''

Always suggests:

- nice today
- beautiful
- perfect

Same every time!

ChatGPT Response:

``The weather is__''

Try 1: "absolutely gorgeous"
Try 2: "quite unpredictable"
Try 3: "exceptionally mild"

Different each time!

The Core Problem:

Your language model gives you probabilities:

Word	Probability
nice	0.60
beautiful	0.20
perfect	0.10
gorgeous	0.05
mild	0.03
unpredictable	0.02

Checkpoint: Design Challenge

How would YOU pick the next word?

Option A: Always pick 0.60? (safe but boring)
Option B: Pick randomly? (creative but risky)
Option C: Something in between?

Key Question: Can you be 100% accurate AND 100% creative?

Your answer: __________

What's Wrong with Each Output?

The Output:

What's Wrong with Each Output?

The Output:

1. `‘‘The cat sat on the cat sat on the cat...’’`

What's Wrong with Each Output?

The Output:

1. ``The cat sat on the cat sat on the cat...''

2. Ask 100 times: ``The weather is __''
Always get: ``nice''

What's Wrong with Each Output?

The Output:

1. `''The cat sat on the cat sat on the cat...''`

2. Ask 100 times: `''The weather is __''`
Always get: `''nice''`

3. `''The weather is xylophone dancing''`

What's Wrong with Each Output?

The Output:

1. ``The cat sat on the cat sat on the cat...''
2. Ask 100 times: ``The weather is __''
   Always get: ``nice''
3. ``The weather is xylophone dancing''
4. ``Paris is capital of France and zlorfnik''

What's Wrong with Each Output?

The Output:

1. ``The cat sat on the cat sat on the cat...''

2. Ask 100 times: ``The weather is __''
Always get: ``nice''

3. ``The weather is xylophone dancing''

4. ``Paris is capital of France and zlorfnik''

5. Set k=40. Peaked: top word P=0.9, rest junk. Flat: 100 words at P=0.01, but only see 40.

What's Wrong with Each Output?

What's Wrong:

The Output:

1. ``The cat sat on the cat sat on the cat...''

2. Ask 100 times: ``The weather is __''
Always get: ``nice''

3. ``The weather is xylophone dancing''

4. ``Paris is capital of France and zlorfnik''

5. Set k=40. Peaked: top word P=0.9, rest junk. Flat: 100 words at P=0.01, but only see 40.

6. Poetry vs factual QA vs code:
One strategy for all?

What's Wrong with Each Output?

The Output:

1. ``The cat sat on the cat sat on the cat...''

2. Ask 100 times: ``The weather is __''
Always get: ``nice''

3. ``The weather is xylophone dancing''

4. ``Paris is capital of France and zlorfnik''

5. Set k=40. Peaked: top word P=0.9, rest junk. Flat: 100 words at P=0.01, but only see 40.

6. Poetry vs factual QA vs code:
One strategy for all?

What's Wrong:

Stuck in repetition loop

(needs path exploration)

What's Wrong with Each Output?

The Output:

1. ``The cat sat on the cat sat on the cat...''

2. Ask 100 times: ``The weather is __''
Always get: ``nice''

3. ``The weather is xylophone dancing''

4. ``Paris is capital of France and zlorfnik''

5. Set k=40. Peaked: top word P=0.9, rest junk. Flat: 100 words at P=0.01, but only see 40.

6. Poetry vs factual QA vs code:
One strategy for all?

What's Wrong:

Stuck in repetition loop
(needs path exploration)

No diversity, always picks max
(needs randomness)

What's Wrong with Each Output?

The Output:

1. ``The cat sat on the cat sat on the cat...''

2. Ask 100 times: ``The weather is __''
Always get: ``nice''

3. ``The weather is xylophone dancing''

4. ``Paris is capital of France and zlorfnik''

5. Set k=40. Peaked: top word P=0.9, rest junk. Flat: 100 words at P=0.01, but only see 40.

6. Poetry vs factual QA vs code:
One strategy for all?

What's Wrong:

Stuck in repetition loop
(needs path exploration)

No diversity, always picks max
(needs randomness)

Too random, nonsensical
(needs controlled randomness)

What's Wrong with Each Output?

The Output:

1. ``The cat sat on the cat sat on the cat...''

2. Ask 100 times: ``The weather is __''
Always get: ``nice''

3. ``The weather is xylophone dancing''

4. ``Paris is capital of France and zlorfnik''

5. Set k=40. Peaked: top word P=0.9, rest junk. Flat: 100 words at P=0.01, but only see 40.

6. Poetry vs factual QA vs code:
One strategy for all?

What's Wrong:

Stuck in repetition loop
(needs path exploration)

No diversity, always picks max
(needs randomness)

Too random, nonsensical
(needs controlled randomness)

Picks very rare words (P=0.00001)
(needs vocabulary limit)

What's Wrong with Each Output?

The Output:

1. ``The cat sat on the cat sat on the cat...''

2. Ask 100 times: ``The weather is __''
Always get: ``nice''

3. ``The weather is xylophone dancing''

4. ``Paris is capital of France and zlorfnik''

5. Set k=40. Peaked: top word P=0.9, rest junk. Flat: 100 words at P=0.01, but only see 40.

6. Poetry vs factual QA vs code:
One strategy for all?

What's Wrong:

Stuck in repetition loop
(needs path exploration)

No diversity, always picks max
(needs randomness)

Too random, nonsensical
(needs controlled randomness)

Picks very rare words (P=0.00001)
(needs vocabulary limit)

Fixed vocab size doesn't adapt
(needs dynamic cutoff)

What's Wrong with Each Output?

The Output:

1. ``The cat sat on the cat sat on the cat...''

2. Ask 100 times: ``The weather is __''
Always get: ``nice''

3. ``The weather is xylophone dancing''

4. ``Paris is capital of France and zlorfnik''

5. Set k=40. Peaked: top word P=0.9, rest junk. Flat: 100 words at P=0.01, but only see 40.

6. Poetry vs factual QA vs code:
One strategy for all?

What's Wrong:

Stuck in repetition loop
(needs path exploration)

No diversity, always picks max
(needs randomness)

Too random, nonsensical
(needs controlled randomness)

Picks very rare words (P=0.00001)
(needs vocabulary limit)

Fixed vocab size doesn't adapt
(needs dynamic cutoff)

Different tasks need different approaches
(needs task-specific strategy)

What's Wrong with Each Output?

The Output:

1. ``The cat sat on the cat sat on the cat...''

2. Ask 100 times: ``The weather is __''
Always get: ``nice''

3. ``The weather is xylophone dancing''

4. ``Paris is capital of France and zlorfnik''

5. Set k=40. Peaked: top word P=0.9, rest junk. Flat: 100 words at P=0.01, but only see 40.

6. Poetry vs factual QA vs code:
One strategy for all?

What's Wrong:

Stuck in repetition loop
(needs path exploration)

No diversity, always picks max
(needs randomness)

Too random, nonsensical
(needs controlled randomness)

Picks very rare words (P=0.00001)
(needs vocabulary limit)

Fixed vocab size doesn't adapt
(needs dynamic cutoff)

Different tasks need different approaches
(needs task-specific strategy)

Next slide: The solutions to each problem!

Each Problem Has Its Decoding Strategy

Problem:

Each Problem Has Its Decoding Strategy

Problem:

1. Repetition loops

Each Problem Has Its Decoding Strategy

Problem:

1. Repetition loops

2. No diversity (always max)

Each Problem Has Its Decoding Strategy

Problem:

1. Repetition loops

2. No diversity (always max)

3. Too random/nonsensical

Each Problem Has Its Decoding Strategy

Problem:

1. Repetition loops

2. No diversity (always max)

3. Too random/nonsensical

4. Very rare words

Each Problem Has Its Decoding Strategy

Problem:

1. Repetition loops

2. No diversity (always max)

3. Too random/nonsensical

4. Very rare words

5. Fixed vocab size

Each Problem Has Its Decoding Strategy

Solution:

Problem:

1. Repetition loops
2. No diversity (always max)
3. Too random/nonsensical
4. Very rare words
5. Fixed vocab size
6. Different task types

Each Problem Has Its Decoding Strategy

Problem:

1. Repetition loops

2. No diversity (always max)

3. Too random/nonsensical

4. Very rare words

5. Fixed vocab size

6. Different task types

Solution:

Beam Search

Explore multiple paths simultaneously

Each Problem Has Its Decoding Strategy

Problem:

1. Repetition loops

2. No diversity (always max)

3. Too random/nonsensical

4. Very rare words

5. Fixed vocab size

6. Different task types

Solution:

Beam Search

Explore multiple paths simultaneously

Sampling

Add randomness to selection

Each Problem Has Its Decoding Strategy

Problem:

1. Repetition loops

2. No diversity (always max)

3. Too random/nonsensical

4. Very rare words

5. Fixed vocab size

6. Different task types

Solution:

Beam Search

Explore multiple paths simultaneously

Sampling

Add randomness to selection

Temperature

Control the randomness level

Each Problem Has Its Decoding Strategy

Problem:

1. Repetition loops

2. No diversity (always max)

3. Too random/nonsensical

4. Very rare words

5. Fixed vocab size

6. Different task types

Solution:

Beam Search
Explore multiple paths simultaneously

Sampling
Add randomness to selection

Temperature
Control the randomness level

Top-k Sampling
Limit to top-k most likely words

Each Problem Has Its Decoding Strategy

Problem:

1. Repetition loops

2. No diversity (always max)

3. Too random/nonsensical

4. Very rare words

5. Fixed vocab size

6. Different task types

Solution:

Beam Search
Explore multiple paths simultaneously

Sampling
Add randomness to selection

Temperature
Control the randomness level

Top-k Sampling
Limit to top-k most likely words

Top-p (Nucleus)
Adaptive vocabulary based on cumulative probability

Each Problem Has Its Decoding Strategy

Problem:

1. Repetition loops

2. No diversity (always max)

3. Too random/nonsensical

4. Very rare words

5. Fixed vocab size

6. Different task types

Solution:

Beam Search
Explore multiple paths simultaneously

Sampling
Add randomness to selection

Temperature
Control the randomness level

Top-k Sampling
Limit to top-k most likely words

Top-p (Nucleus)
Adaptive vocabulary based on cumulative probability

Task-Specific Strategy
Match decoding method to task requirements

Each Problem Has Its Decoding Strategy

Problem:

1. Repetition loops

2. No diversity (always max)

3. Too random/nonsensical

4. Very rare words

5. Fixed vocab size

6. Different task types

Solution:

Beam Search
Explore multiple paths simultaneously

Sampling
Add randomness to selection

Temperature
Control the randomness level

Top-k Sampling
Limit to top-k most likely words

Top-p (Nucleus)
Adaptive vocabulary based on cumulative probability

Task-Specific Strategy
Match decoding method to task requirements

Today we'll learn each of these strategies in detail!

Six Problems We Need to Solve Today

Each Problem = One Method We'll Learn

Problem 1: Repetition Loop

``The cat sat on the cat sat on the cat...''
Model gets stuck repeating!
→ **Beam Search**

Problem 2: Always Boring

Ask 100 times: ``The weather is __''
Always: ``nice'' (never "gorgeous")
→ **Sampling**

Problem 3: Random Nonsense

``The weather is xylophone dancing''
Too creative = gibberish!
→ **Temperature**

Problem 4: Weird Words

``Paris is the capital of France and zlorfnik''
Picks ultra-rare words (P=0.00001)
→ **Top-k**

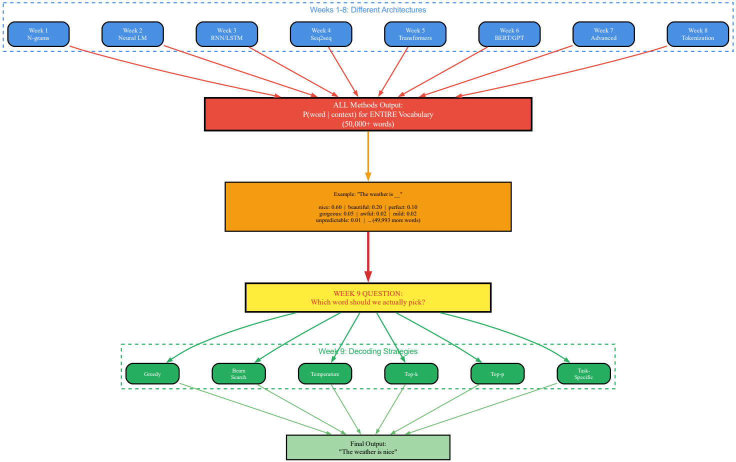
Problem 5: Fixed Vocabulary

Peaked dist: top-40 = overkill
Flat dist: top-40 = too few
→ **Top-p (Nucleus)**

Problem 6: Which Method?

Creative writing? Factual QA?
Code generation? Different needs!
→ **Task Guide**

What We've Learned So Far



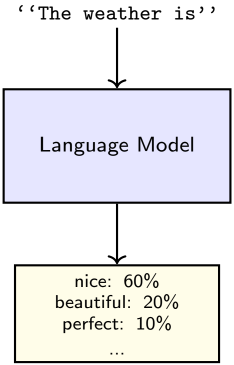
The Big Picture

All methods output the same format:
Probability distribution over vocabulary

Week 9 = Final step in text generation pipeline:
Choose which word to actually use

The Core Task: Choose the Next Word

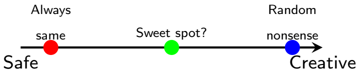
What Your Model Gives You:



What You Need to Do:

Decoding: Convert probabilities into actual text

The Trade-off:



We'll explore:

- Greedy (always safe)
- Beam search (explore paths)
- Sampling (add randomness)
- How to find YOUR sweet spot

Strategy 1: Greedy Decoding

The Rule:

Always pick the highest probability

Example:

Step	Probs	Pick
"The"	nice:0.6, good:0.3	nice
"nice"	day:0.7, weather:0.2	day
"day"	is:0.5, was:0.3	is

Result: "The nice day is..."

Pros:

- Fast
- Deterministic (same every time)
- High probability output

The Problem:

Greedy often gets stuck!

Example:
"The dog likes the dog likes the dog likes..."

Why? Each step picks "the" (0.4) over "a" (0.3), but the FULL sequence "a cat" ($0.3\times0.5=0.15$) beats "the dog" ($0.4\times0.2=0.08$)!

Cons:

- Repetitive
- Gets stuck in loops
- Misses better sequences
- Boring/generic output

Intuition: Why Greedy Fails

Locally optimal $\neq$ globally optimal!

Idea: Keep Multiple Paths Open

The Analogy:

GPS Navigation:
Greedy: Take fastest road NOW
→ Might hit traffic later
Beam: Consider top 3-5 routes
→ Pick best COMPLETE path

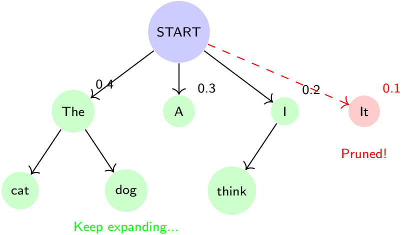
The Rule:

Keep top-k hypotheses at each step

Beam Size = 3:

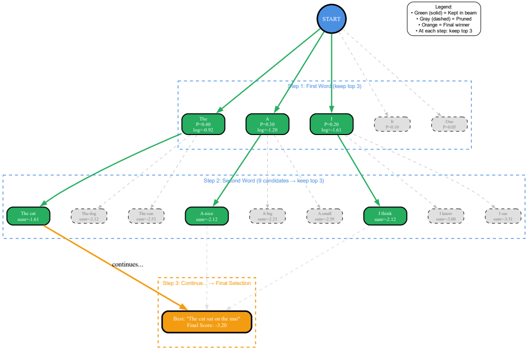
- Track 3 best sequences
- Expand each by V words
- Keep top 3 of all candidates
- Repeat until done

Concrete Example (beam=3):



Scoring:
Score $= \sum \log P(\text{word}_i)$
Example:
"The cat" $= \log(0.4) + \log(0.5) = -0.61$
"A dog" $= \log(0.3) + \log(0.4) = -1.22$

Beam Search: The Full Picture



Pros:

- Better than greedy
- Considers context

Cons:

- Still deterministic
- Still can be repetitive

Test Your Understanding

Question 1:
Why does greedy decoding sometimes produce worse sequences than beam search?

- A) Greedy is slower
- B) Greedy only looks one step ahead
- C) Greedy uses wrong probabilities
- D) Greedy is random

Question 2:
If beam size = 1, what is beam search equivalent to?

- A) Random sampling
- B) Greedy decoding
- C) Exhaustive search
- D) Top-k sampling

Answer 1: B

Greedy picks best word NOW, ignoring future consequences. Beam search considers multiple paths and picks best FULL sequence.

Example: "the dog" (0.4×0.2) loses to "a cat" (0.3×0.5) overall!

Answer 2: B

Beam size = 1 means we only keep 1 hypothesis at each step = greedy decoding!

Beam search generalizes greedy.

Problem: Greedy and Beam are TOO Predictable

Why Randomness?

Real World: Human Writing

Humans don't always pick the most probable word!

Boring: "The weather is nice today"
Better: "The weather is absolutely gorgeous"

"gorgeous" might have P=0.05, but it's more interesting!

Pure Sampling:

Sample from the full probability distribution

If P(nice)=0.6, P(gorgeous)=0.05:
→ 60% chance of "nice"
→ 5% chance of "gorgeous"

Concrete Example:

Word	P
nice	0.60
beautiful	0.20
perfect	0.10
gorgeous	0.05
mild	0.03
weird	0.02

5 samples might give:

1. nice (highest prob)
2. nice (again)
3. beautiful
4. gorgeous (surprise!)
5. nice

Problem: Sometimes picks "weird"!

Temperature: Control the Randomness

The Formula:

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

where z_i = logit, T = temperature

What Temperature Does:

- $T < 1$: Sharper (more confident)
- $T = 1$: Original distribution
- $T > 1$: Flatter (more random)

Analogy:
High temperature = melted ice cream
(everything mixes together, uniform)
Low temperature = frozen ice cream
(distinct flavors, concentrated)

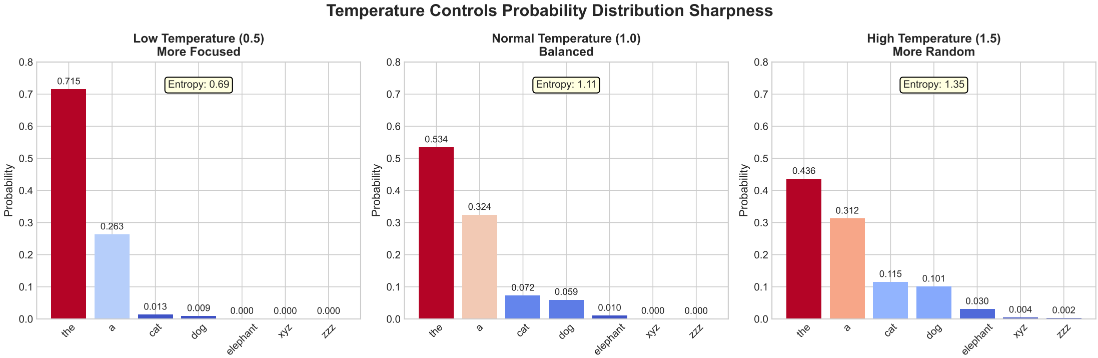
Concrete Example:
Original: [0.6, 0.2, 0.1, 0.05, 0.05]

T	nice	beautiful	perfect	gorgeous	mild
0.5	0.82	0.13	0.03	0.01	0.01
1.0	0.60	0.20	0.10	0.05	0.05
1.5	0.45	0.25	0.15	0.08	0.07
2.0	0.35	0.27	0.18	0.11	0.09

Recommendations:

- Factual Q&A: T=0.1-0.3
- Translation: T=0.3-0.7
- Dialogue: T=0.7-1.0
- Creative writing: T=0.9-1.5

Temperature in Action



Key Insight: Temperature lets you control the creativity-accuracy tradeoff!

Top-k Sampling: Limit the Vocabulary

The Problem with Pure Sampling:

With 50,000 word vocabulary, might sample very unlikely words!

"The weather is xylophone" (P=0.00001)

The Solution:

Only sample from top-k most likely words

Example (k=10):

1. Sort words by probability
2. Keep only top 10
3. Renormalize probabilities
4. Sample from these 10

Concrete Example:

Full vocabulary (50,000 words):
nice (0.6), beautiful (0.2), ... xylophone (0.00001)

Top-k=5:

Word	Original	Renormalized
nice	0.60	0.632
beautiful	0.20	0.211
perfect	0.10	0.105
gorgeous	0.05	0.053
mild	0.03	0.032
Others (49,995 words)		0.000

Typical values:

- k=10: Very focused
- k=40: Balanced
- k=100: Diverse

Top-p Sampling: Dynamic Vocabulary

Problem with Top-k:

Fixed k doesn't adapt to distribution!

Flat distribution: k=40 might be too few
Peaked distribution: k=40 might include junk

Top-p Solution:

Keep smallest set with cumulative probability $\geq$ p

Algorithm:

1. Sort by probability
2. Add words until cumsum $\geq$ p
3. Sample from this "nucleus"

Example (p=0.9):

Word	P	Cumsum	Include?
nice	0.60	0.60	✓
beautiful	0.20	0.80	✓
perfect	0.10	0.90	✓
gorgeous	0.05	0.95	×
mild	0.03	0.98	×
weird	0.02	1.00	×

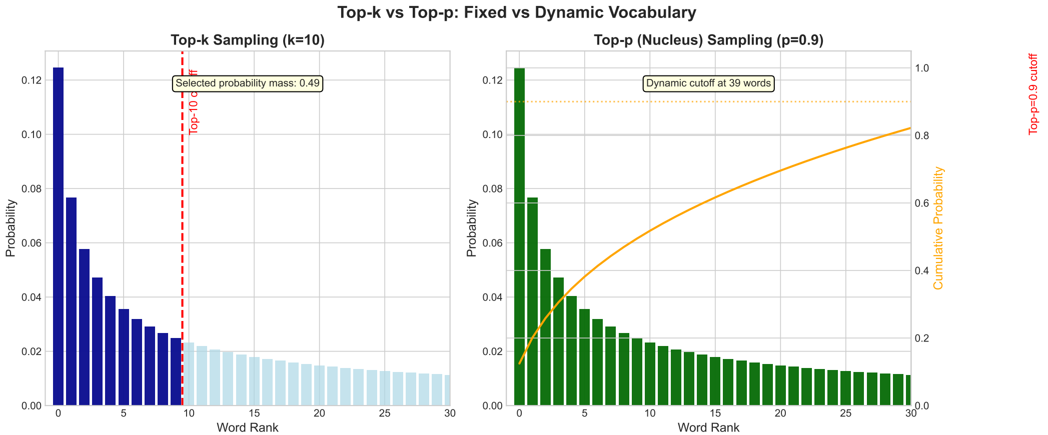
Keep 3 words (dynamic!)

Advantages:

- Adapts to distribution shape
- Prevents sampling tail
- More robust than top-k

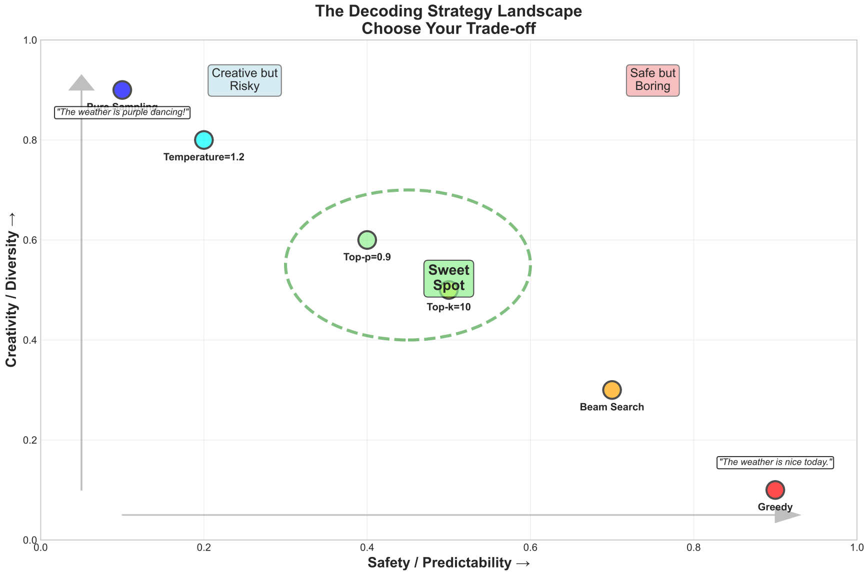
Typical values: p=0.9 or p=0.95

Top-k vs Top-p: Side by Side



Key Difference: Top-k is fixed, Top-p adapts to distribution shape

The Complete Decoding Landscape



Test Your Understanding

Question 1:
What does temperature=0.1 do?

- A) Makes distribution flatter
- B) Makes distribution sharper
- C) Removes low-prob words
- D) Adds randomness

Question 2:
Distribution: [0.7, 0.15, 0.10, 0.03, 0.02]
With p=0.9, how many words in nucleus?

- A) 1
- B) 2
- C) 3
- D) 5

Answer 1: B

Low temperature ($T<1$) makes the distribution SHARPER = more confident = less random.

Example: [0.6, 0.2, 0.2] becomes [0.8, 0.1, 0.1]

Answer 2: C (3 words)

Cumulative sum:
0.7 (word 1)
0.85 (word 2)
0.95 (word 3) $\leftarrow$ exceeds 0.9!

Stop here, use 3 words.

You Can Use Multiple Techniques Together!

Common Combinations:

Temperature + Top-p

1. Apply temperature scaling
2. Filter with top-p
3. Sample from nucleus

Example: T=0.8, p=0.95

Beam + Sampling

Beam search, but sample within beam instead of greedy expansion

Gets diversity + good paths

Additional Tricks:

- **Repetition penalty:** Reduce prob of recently used words
- **Length normalization:** Don't favor short sequences
- **Min-p:** Absolute minimum threshold
- **Typical sampling:** Sample based on entropy

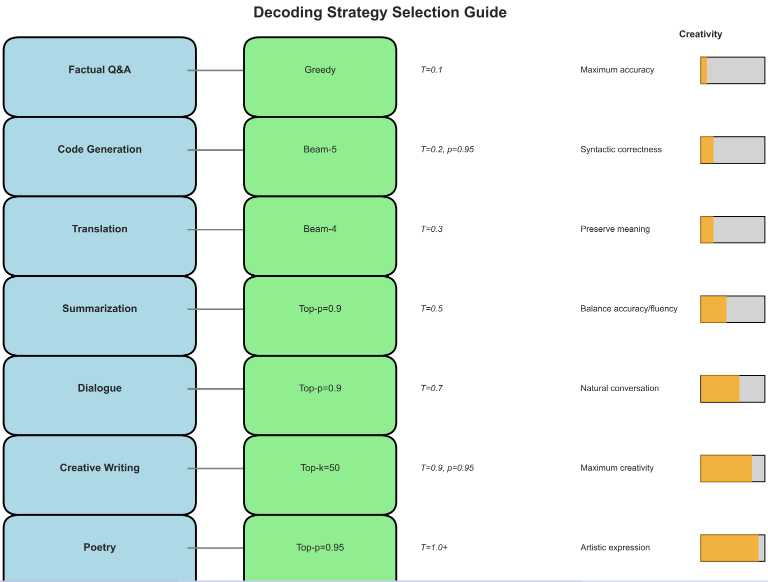
Real World: ChatGPT Settings

ChatGPT uses temperature + top-p:

Default: T=0.7, p=0.95
Creative mode: T=0.9
Precise mode: T=0.3

Evaluating Decoding Quality

Which Method for Which Task?



Example 1: Factual Question Answering

Task: Answer: "What is the capital of France?"

Priority: Accuracy > Creativity

Best Strategy:

Greedy or Beam-3

Temperature = 0.1-0.3

Results:

Method	Output
Greedy	"Paris"
T=0.8	"Paris, the city of lights"
T=1.5	"Lyon is also nice"

Why Greedy Wins:

- Only ONE correct answer
- Creativity adds errors
- Speed matters
- Consistency important

Real World: Search Engines

Google uses greedy-like decoding for autocomplete:

- Must be accurate
- Must be fast
- Consistency builds trust
- Users want predictability

Metrics:

Accuracy: 95% (greedy) vs 70% (T=1.5)

Example 2: Creative Story Writing

Task: Continue: “Once upon a time...”

Priority: Creativity > Accuracy

Best Strategy:

Top-k=50 or Top-p=0.95

Temperature = 0.9-1.2

Comparison:

Greedy:
“Once upon a time there was a beautiful princess...”

T=1.0, p=0.95:
“Once upon a time, beneath an ancient oak, a curious raven discovered...”

Why Sampling Wins:

- Many valid continuations
- Repetition is boring
- Readers want surprise
- No “ground truth”

Real World: NovelAI

AI writing assistants use:

- Temperature: 0.8-1.2
- Top-p: 0.9-0.95
- Repetition penalty: 1.1
- Users can adjust!

Human Preference:
75% prefer sampling over greedy for stories

Example 3: Code Generation

Task: Complete: `def factorial(n):`

Priority: Correctness + Some diversity

Best Strategy:

Beam search (size=5)

Temperature = 0.2-0.5

Why Beam:

- Syntax must be correct
- But multiple valid solutions
- Beam finds different approaches
- Pick best with tests

Results:

Greedy (always same):

```
if n == 0:
    return 1
return n * factorial(n-1)
```

Beam (multiple options):

```
# Option 1: Recursive
return 1 if n==0 else n*factorial(n-1)

# Option 2: Iterative
result = 1
for i in range(1, n+1):
    result *= i
return result
```

Metrics:

Pass rate: Beam-5 (85%) > Greedy (78%)

Practical Tips: Tuning Your Parameters

Start Conservative:

1. Begin with T=0.7, p=0.9
2. Generate 10 samples
3. Evaluate quality
4. Adjust based on problems

Problem: Too repetitive?

→ Increase T (try 1.0)
→ Increase p (try 0.95)
→ Add repetition penalty

Problem: Too random/nonsensical?

→ Decrease T (try 0.5)
→ Decrease p (try 0.85)
→ Try beam search

Grid Search Strategy:

T	p	Quality
0.5	0.9	High acc, boring
0.7	0.9	Balanced
1.0	0.9	Creative
0.7	0.95	More diverse
1.2	0.95	Very creative

Evaluation Metrics:

- **Distinct-n:** Unique n-grams (diversity)
- **Perplexity:** Model confidence
- **Repetition rate:** n-gram overlap
- **Human eval:** Ultimate test

Intuition: Rule of Thumb

Start at T=0.7, adjust by ± 0.2 until output quality feels right

Common Mistakes and How to Avoid Them

Mistake 1: Temperature Too High

T=2.0 → Gibberish
"The weather is purple dancing elephant"

Fix: T $\leq$ 1.5 for most tasks

Mistake 2: Using Greedy for Creativity

"Write a unique poem"
Greedy → Generic clichés

Fix: Use sampling for creative tasks

Mistake 3: No Repetition Control

"The cat sat on the cat sat on the cat..."

Fix: Add repetition penalty (1.1-1.3)

Mistake 4: Ignoring Task Type

Translation with T=1.5 → Wrong meaning
Code with T=1.2 → Syntax errors

Fix: Match strategy to task

Mistake 5: Too Large beam_size

beam=50 → Too slow, minimal gain

Fix: beam=3-5 usually sufficient

Mistake 6: Not Testing

Assuming defaults work for your use case

Fix: Always generate 10+ samples and evaluate

Best Practices: Your Decoding Checklist

1. **Match Strategy to Task:**
   - Factual $\rightarrow$ Greedy/Beam
   - Creative $\rightarrow$ Sampling
   - Code $\rightarrow$ Beam + low T
   - Dialogue $\rightarrow$ T=0.7-0.9
2. **Start Conservative:**
   - T=0.7, p=0.9
   - Gradually increase randomness
   - Stop when quality drops
3. **Always Use Top-p with Temperature:**
   - Prevents tail sampling
   - More robust than top-k
   - p=0.9 or 0.95 works well
4. **Control Repetition:**
   - Add repetition penalty
   - Monitor n-gram overlap
   - Adjust if >20% repetition
5. **Evaluate Properly:**
   - Generate 10+ samples
   - Check diversity (distinct-n)
   - Human evaluation critical
   - A/B test strategies
6. **Iterate:**
   - Decoding is an art + science
   - No universal best settings
   - Domain-specific tuning needed
   - User feedback matters

Decoding in Production Systems

ChatGPT:

- Base: T=0.7, p=0.95
- Creative mode: T=0.9
- Precise mode: T=0.3
- Repetition penalty: 1.1

Google Translate:

- Beam search (size=4)
- Length normalization
- Low temperature (T=0.3)
- Coverage penalty

GitHub Copilot:

- Beam search (size=5)
- Temperature=0.2
- Ranks by test pass rate
- Syntax validation

Jasper AI (Marketing):

- High temperature (T=1.0-1.2)
- Top-p=0.95
- Strong repetition penalty
- Multiple variations

Character.AI (Dialogue):

- Temperature=0.8
- Top-p=0.9
- Personality-specific tuning
- Context-aware adjustments

Real World: User Control

Many systems let users adjust:

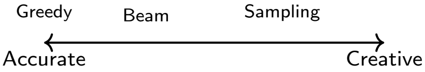
- Temperature slider
- "Creativity" dial
- Multiple output options

The Decoding Tradeoff

Core Concepts:

1. **Greedy:** Fast but boring
2. **Beam:** Better paths, still deterministic
3. **Sampling:** Creative but risky
4. **Temperature:** Control randomness
5. **Top-k/p:** Limit vocabulary

The Fundamental Tradeoff:



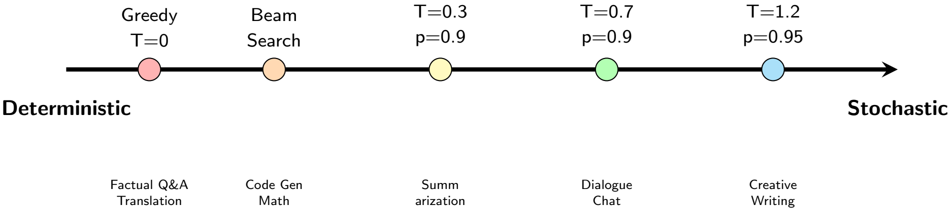
Practical Guidelines:

> **If accuracy critical:**
> Greedy or Beam + low T
>
> **If creativity needed:**
> Sampling + higher T
>
> **If unsure:**
> T=0.7, p=0.9 (good default)

Remember:

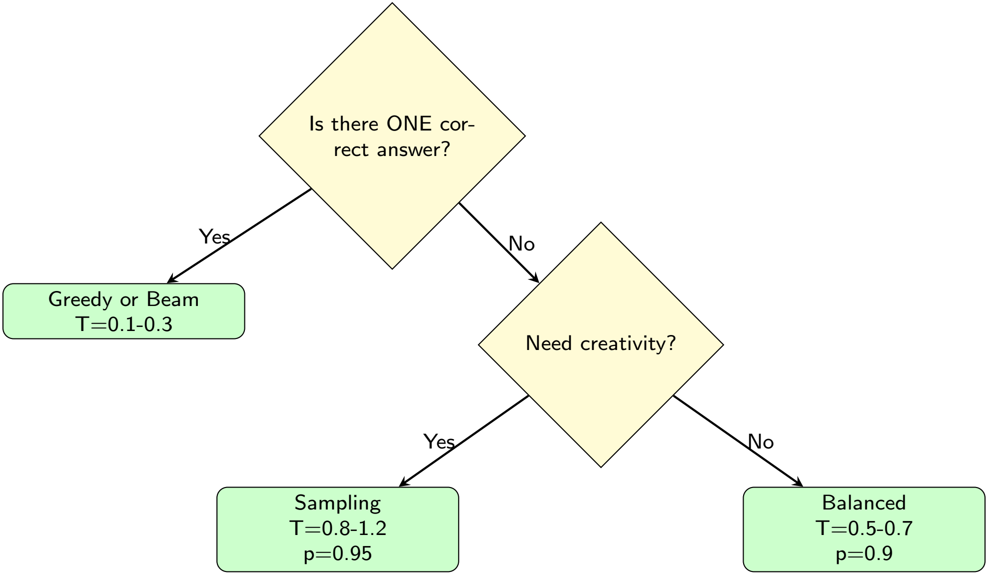
- No universal best method
- Task determines strategy
- Always test and iterate
- User feedback is gold

The Complete Decoding Spectrum



Your Task Determines Your Position on This Spectrum

Quick Decision Guide



Next Week: Fine-Tuning & Prompt Engineering

What We Learned:

- How to decode probabilities
- Trade-off: accuracy vs creativity
- Multiple strategies available
- Task determines best approach

Questions to Ponder:

1. Can you combine beam + sampling?
2. How to auto-tune parameters?
3. What about constrained decoding?

Week 10 Preview:

- Fine-tuning pre-trained models
- Prompt engineering techniques
- Few-shot learning
- Adapting models to your task

Connection:
Decoding controls HOW the model generates text.
Fine-tuning controls WHAT the model knows.
Together: powerful customization!

Lab: Implement all decoding strategies and compare!

Appendix A: Mathematical Formulations

Temperature Scaling:

$$P_T(w_i \mid w_{<i}) = \frac{\exp(z_i/T)}{\sum_{j=1}^{V} \exp(z_j/T)} \tag{1}$$

where z_i = logit for word i, T = temperature, V = vocabulary size

Top-k Sampling:

$$V_k = \{w_1, w_2, \ldots, w_k\} \text{ where } P(w_i) \geq P(w_j) \text{ for } i \leq k < j \tag{2}$$

Sample from renormalized distribution over V_k only

Top-p (Nucleus) Sampling:

$$V_p = \min\left\{ V' : \sum_{w \in V'} P(w) \geq p \right\} \tag{3}$$

Beam Search Scoring:

$$\text{score}(w_{1:t}) = \sum_{i=1}^{t} \log P(w_i \mid w_{<i}) \tag{4}$$

With length normalization:

Appendix B: Implementation Pseudocode

Temperature Sampling:

```
def sample_with_temperature(logits, T):
    # Scale logits by temperature
    scaled = logits / T
    # Apply softmax
    probs = softmax(scaled)
    # Sample from distribution
    return sample(probs)
```

Top-p Sampling:

```
def top_p_sampling(logits, p):
    # Sort probabilities descending
    probs = softmax(logits)
    sorted_probs, indices = sort(probs, descending=True)
    # Find cutoff
    cumsum = cumulative_sum(sorted_probs)
    cutoff = argmax(cumsum >= p) + 1
    # Keep only nucleus
    nucleus_probs = sorted_probs[:cutoff]
    nucleus_indices = indices[:cutoff]
    # Renormalize and sample
    nucleus_probs = nucleus_probs / sum(nucleus_probs)
    sampled_idx = sample(nucleus_probs)
    return nucleus_indices[sampled_idx]
```

Translation Task (WMT14 EN-DE):

Method	BLEU	Speed	Diversity	Repetition
Greedy	26.5	100%	0.21	15%
Beam-4	**28.2**	25%	0.24	12%
T=0.5, p=0.9	26.8	80%	0.35	8%
T=1.0, p=0.9	25.1	80%	0.52	5%

Story Generation (WritingPrompts):

Method	Coherence	Creativity	Human Pref	Distinct-2
Greedy	4.2/5	2.1/5	15%	0.18
Beam-5	4.0/5	2.5/5	20%	0.22
T=0.8, p=0.9	3.8/5	3.9/5	35%	0.51
T=1.0, p=0.95	3.5/5	**4.3/5**	**40%**	**0.62**

Key Observations:

- Beam wins on translation (objective metrics)
- Sampling wins on creative writing (human preference)
- Trade-off between coherence and diversity is real
- No single best method across tasks