# *Inference 1:* Decoding and Generation Algorithms

11-711 Spring 2025

# Good news!

We have a great new model *M*!

7 billion parameters!

Pretrained on trillions of tokens of text!

# So what's in the box?

A model defines a *conditional probability distribution*

$$P(Y|X) = \prod_{j=1}^{J} P(y_j \mid X, y_1, \ldots, y_{j-1})$$

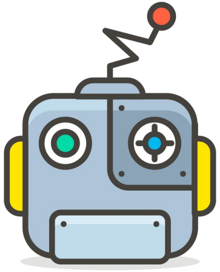
# A model defines a *conditional probability distribution*

| Input *X* | Output *Y* | Task |
|---|---|---|
| English text | Japanese | Translation |
| Question | Answer | Question-answering |
| Document | Short description | Summarization |
| Utterance | Response | Response generation |
| Chess game state | Next chess move | Game-playing |
| Math problem | Answer | Math reasoning |

# (modern) LMs are *locally normalized*

Monotonically non-increasing probability scores

The U.S. president in 2024 was

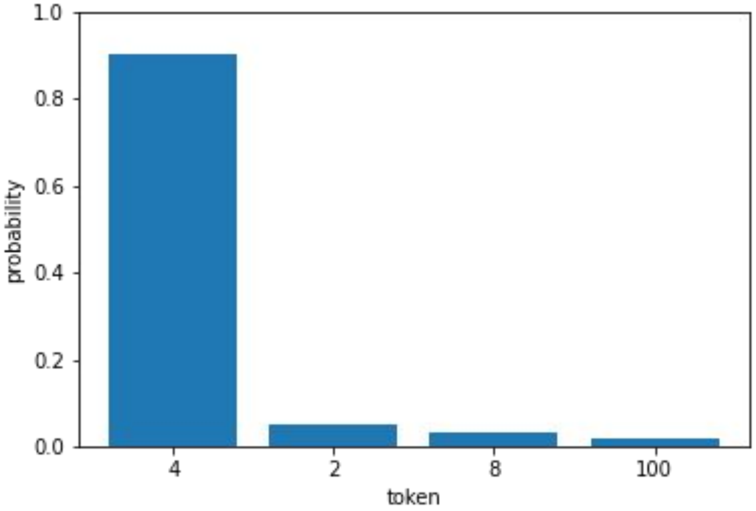


At the sequence level, the first output is clearly better; but if it starts with very low probability tokens, it can never have high overall probability

easy/fast to train with local normalization, but harder to do inference with global constraints

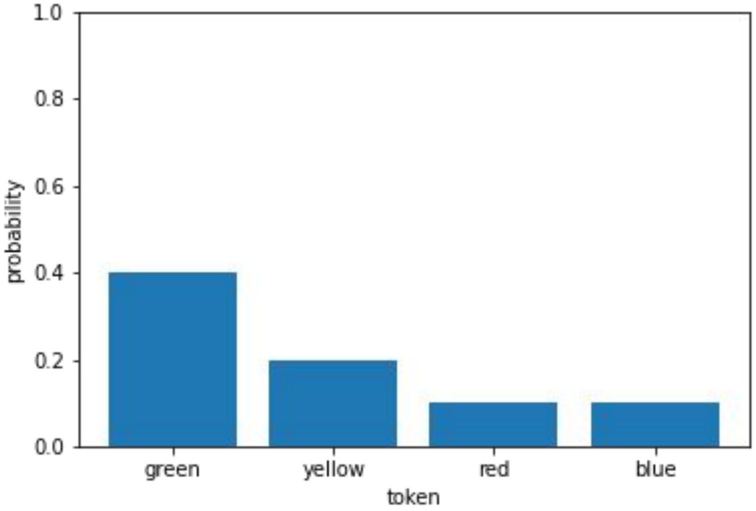
# Probability distributions: confidence

M(“2 + 2 = ”):




**4** (high confidence)

M(“Sean’s favorite color is ”):




**green** (low confidence)

# Calibration (quick reminder)

A model is **well-calibrated** if the confidence score is well-correlated with the probability of correctness

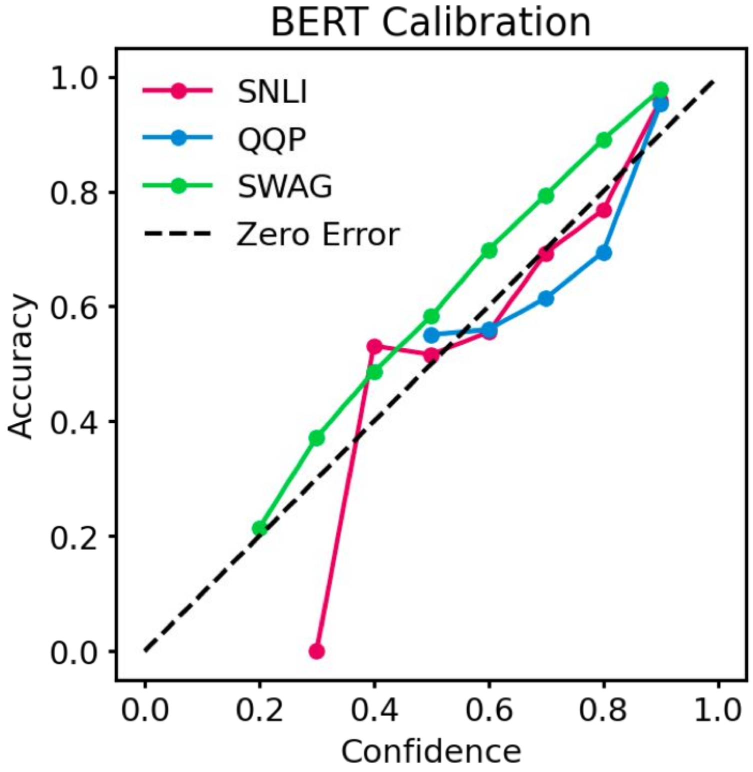
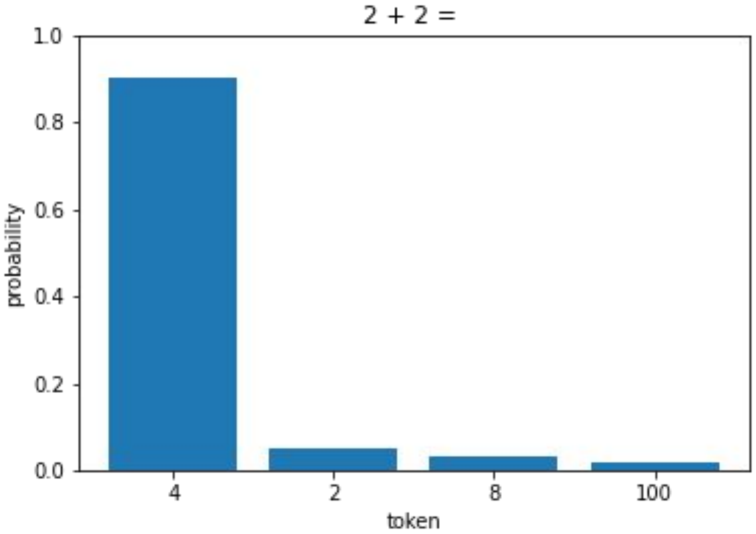



*Figure from Desai & Durrett (2020)*

# Probability distributions: hallucination

Models generally assign non-zero probability to some incorrect outputs




This is true *even if all pretraining data is factual*!

Calibrated Language Models Must Hallucinate

Adam Tauman Kalai
Microsoft Research

Santosh S. Vempala
Georgia Tech

December 5, 2023

*Reference:* [*Kalai & Vempala, 2023*]

# How do we get outputs from this model?

$$P(Y|X) = \prod_{j=1}^{J} P(y_j \mid X, y_1, \ldots, y_{j-1})$$

We know:

- The model's distribution of likelihood over all vocabulary tokens *V*, for the next time step, given the input and previous generations

We want:

- a "good" output

Previous: models as distributions

up next:

# decoding as optimization

# Mode-seeking decoding methods

Given our inputs (evidence) and the model's parameters (prior), what's the *single most likely* output?

$$\hat{Y} = \underset{Y}{\operatorname{argmax}}\ P(Y|X)$$

(this is the mode of the distribution over outputs!)

# Greedy decoding

Idea: choose the single most likely token at each step

$$y_j = \texttt{argmax} \;\; P(y_j | X, y_1, \ldots, y_{j-1})$$

**Exactly** what we want for a single-token output!

What about longer sequences? Doesn't always yield the highest-probability output :(

# Beam search

Idea: maintain a few options, so we don’t miss a high-probability completion “hidden” behind a lower-probability prefix

Breadth-first search: explore many options for each decoding step before generating candidates for the next step

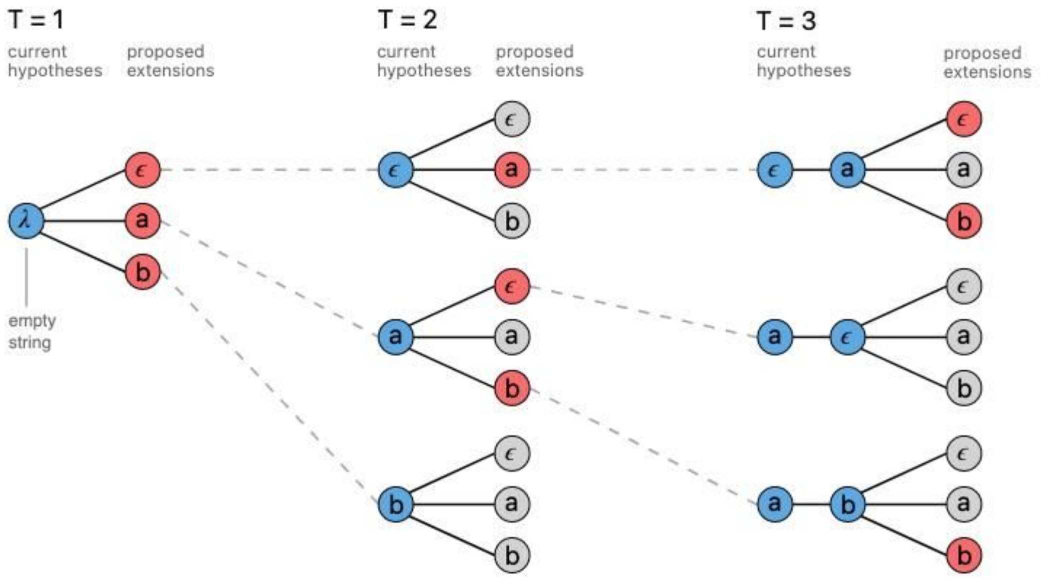



*Figure from the [PyTorch blog on fast decoding]*

# What does this look like in huggingface?

Greedy decoding:

```
model.generate(do_sample=False, num_beams = 1)
```

Beam search:

```
model.generate(do_sample=False, num_beams = <n>)
```

# Is the highest-probability output best?

Outputs with *low probability* tend to be worse than those with *high probability*

| Probability | Output |
| --- | --- |
| 0.3 | The cat sat down. |
| 0.001 | The cat grew wings. |

But when you're *just* comparing the top outputs... it's less clear

| Probability | Output |
| --- | --- |
| 0.3 | The cat sat down. |
| 0.25 | The cat ran away. |

# Wait: is the highest-probability output best?

What if we have multiple ways to say the same thing? Probability is **split** between them

# Is the highest-probability output best?

6 outputs:

| **Probability** | **Output** |
|---|---|
| 0.3 | The cat sat down. |
| 0.25 | The cat ran away. |
| 0.2 | The cat sprinted off. |
| 0.149 | The cat got out of there. |
| 0.1 | The cat is very small. |
| 0.001 | The cat grew wings. |

The single most probable output is that the cat sat down…

But 60% of the probability mass says something meaning “the cat left”!

The probability of this is **split** over multiple similar generations

# Issues with MAP: length

In early models: the mode given any prefix was often <EOS>

Length is a confounder for both quality and probability

- Annotators for preference data prefer long outputs
- With conditional probability distributions, longer outputs are usually lower probability

Solution: length “penalty”

# Issues with MAP: repetition

We generated some text and then the sequence repeated

and then the sequence repeated

and then the sequence repeated

Solution?

- Train a better model!
- Repetition penalty: discount the scores of previously-generated tokens (Keskar & McCann et al, 2019)

# Issues with MAP: Atypicality

If you have a coin with a 60% chance of yielding tails, and you flip it 100 times…

The *single most likely output:* 100 tails

A *typical* output: slightly more tails than heads

# Issues with MAP: Curse of Beam Search

What is better, decoding with beam width 5 or beam width 500?

Very large beam widths can *decrease* performance on downstream metrics– despite finding higher-probability sequences

# Improving diversity: diverse and stochastic beam search

Idea: try to do more exploration *during* beam search

**Diverse beam search:** modify the **scoring** when pruning beams to avoid choosing overly similar beams

**Stochastic beam search:** modify the **next token selection** to sample instead of using the top greedy decodings

# What does this look like in huggingface?

Diverse beam search

```
model.generate(do_sample=False, num_beams = n, num_beam_groups = m)
```

Stochastic beam search:

```
model.generate(do_sample=True, num_beams = n)
```

# Locally typical decoding

Information theory approach

Previous: decoding as optimization

up next:

# sampling from LMs

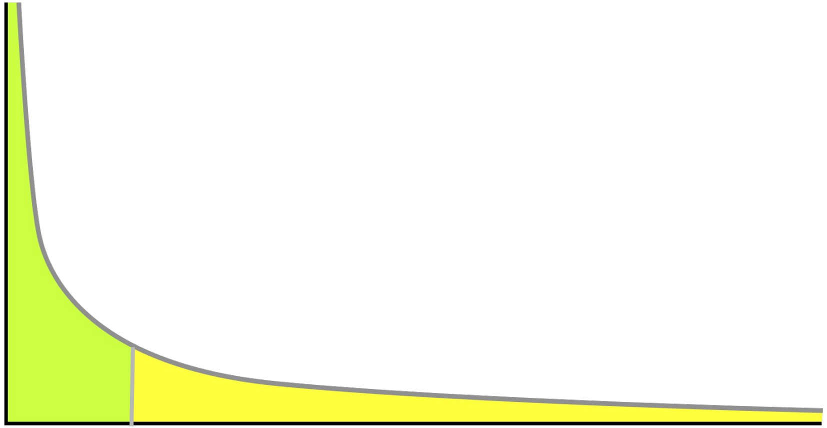
# Ancestral Sampling

$$y_j \sim P(y_j | X, y_1, \ldots, y_{j-1})$$

- Exactly samples from model distribution!
- So we’re done… right?

# Issues with ancestral sampling: long tail

Llama has 32,000 vocabulary tokens!



Even if each individual token in the long tail has very little probability…. these small probabilities add up

Figure from [Wikipedia]

# What if we just ignore the long tail?

**Top-k sampling:** only sample from the most probable <k> next tokens

E.g., for k=6:

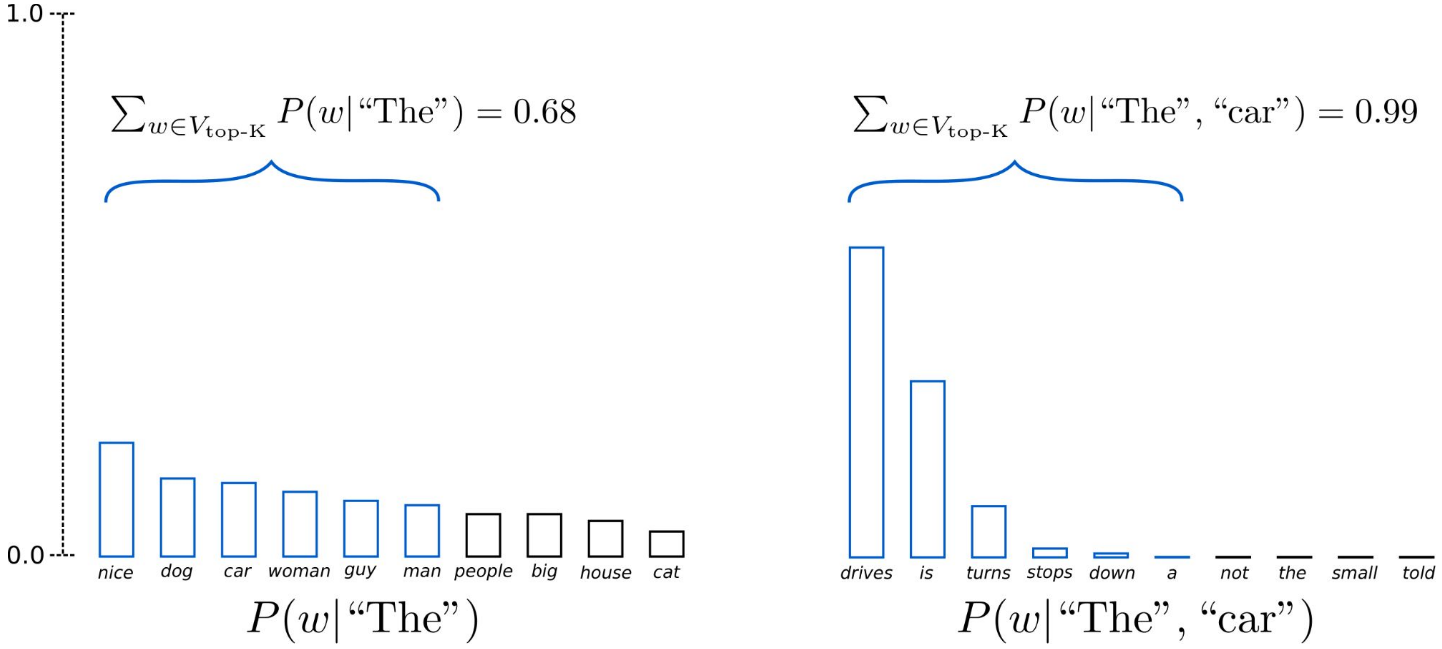



*Figure from the [HuggingFace blog on text generation]*

# What if we just ignore the long tail?

**Top-p (nucleus) sampling:** only sample from the top <p> probability mass

E.g., for p=0.94:

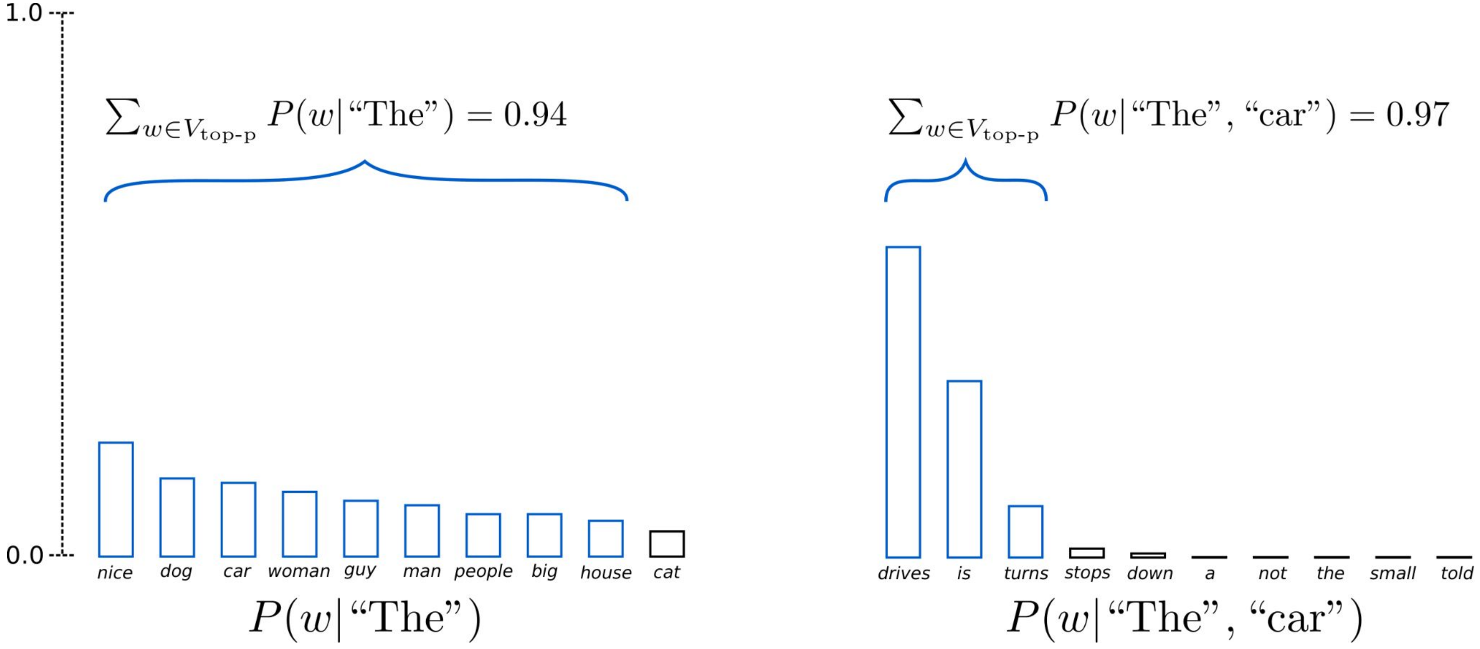



*Figure from the [HuggingFace blog on text generation]*

# What if we just ignore the long tail?

**Epsilon sampling:** only sample tokens with probability of at least ϵ

E.g., for ϵ=0.05:

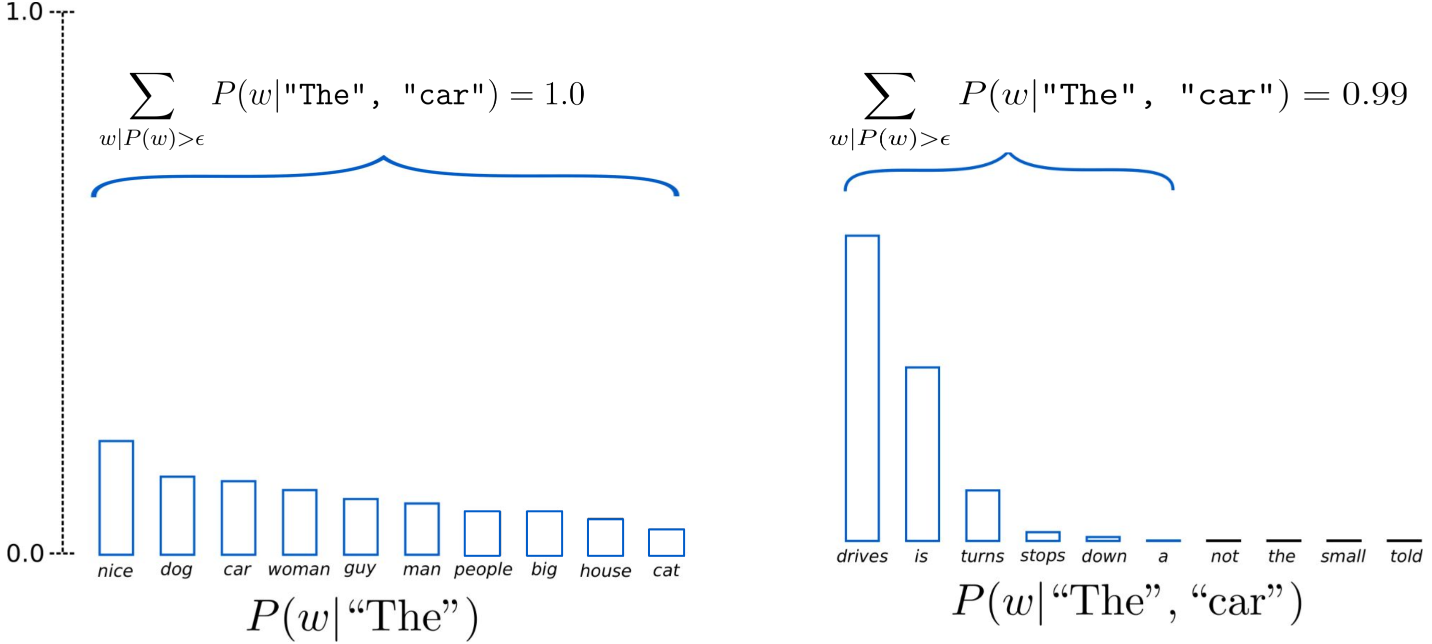



*Figure modified from the [HuggingFace blog on text generation]*

# Basis-aware threshold sampling

Idea: not all tokens are in the long-tail for the same reason

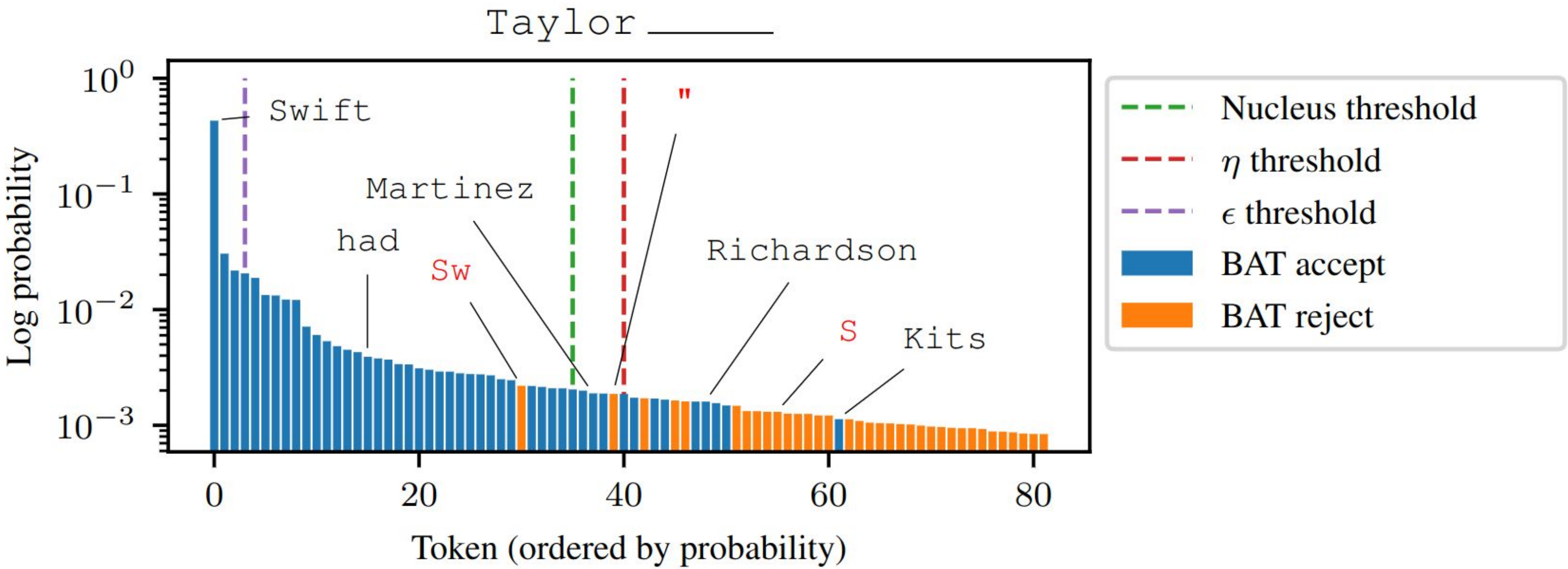
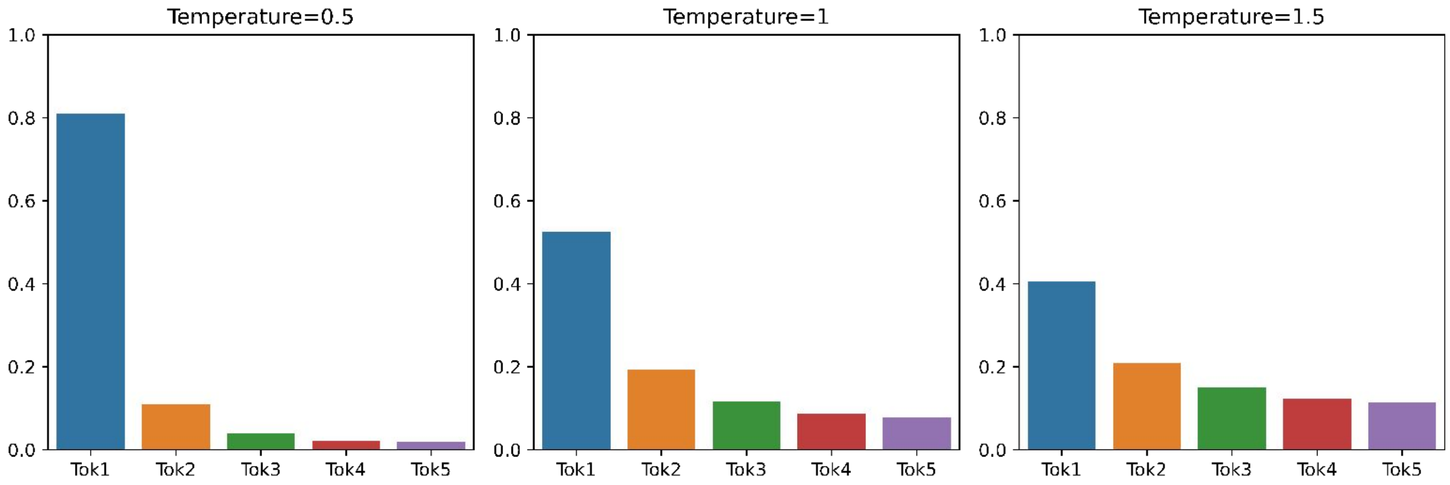



*Figure from [Finlayson et al (2023)]*

# Distribution temperature

Idea: manipulate the distribution to have higher (or lower) probability on the top few tokens



# What does this look like in huggingface?

Ancestral sampling

```
model.generate(do_sample=True, num_beams = 1)
```

Top-k sampling:

```
model.generate(do_sample=True, num_beams = 1, top_k = k)
```

Nucleus sampling:

```
model.generate(do_sample=True, num_beams = 1, top_p = p)
```

# What does this look like in huggingface?

Epsilon sampling

```
model.generate(do_sample=True, num_beams = 1, epsilon_cutoff=e)
```

Modifying temperature

```
model.generate(do_sample=True, num_beams=1, temperature = 0.8)
```

# Microstat?

Add if there’s time left in the talk

# Contrastive decoding

Smaller models make different mistakes– can we learn from these to improve our models?

Choose outputs that the "expert" finds much more likely than the "amateur"

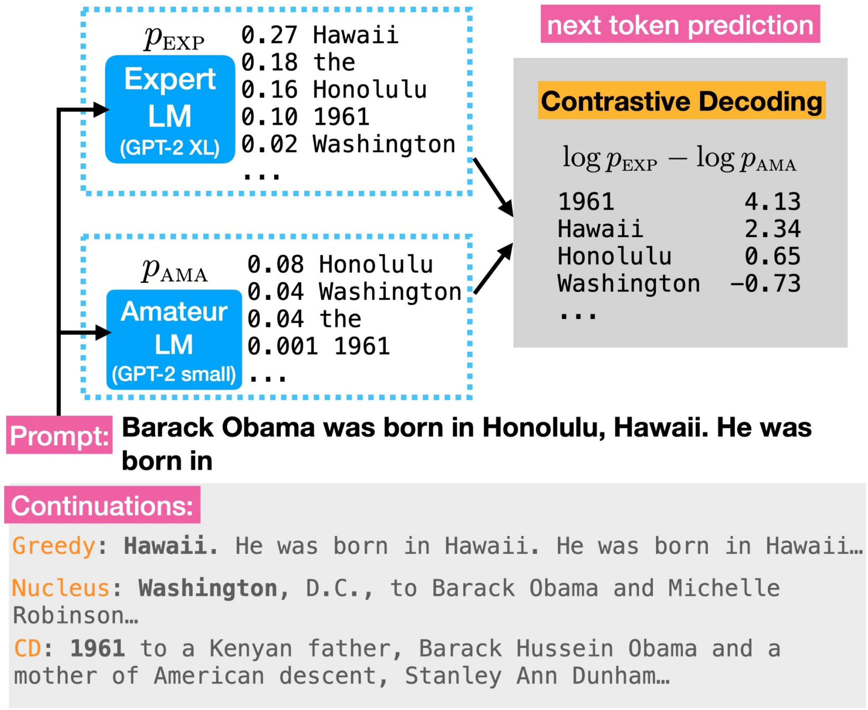



*Figure from [Li et al (2023)]*

# What does this look like in huggingface?

Contrastive decoding

```
sampling_method(model1.forward(seq) - model2.forward(seq))
```

*! not the same as contrastive search*

Previous: sampling from LMs

up next:

# constrained generation

# Templamatic constraints

We’d like the model to output valid JSON, according to some schema we’ve developed

But even good models can struggle at this task….

# Templamatic constraints

Format the following information using the JSON schema:

*Taylor Swift was born December 13, 1989*

| Key | Type |
|---|---|
| **name** | string |
| **birth year** | int |

Slide credit: Matt Finlayson

# Templamatic constraints

Idea: represent the schema as a state machine

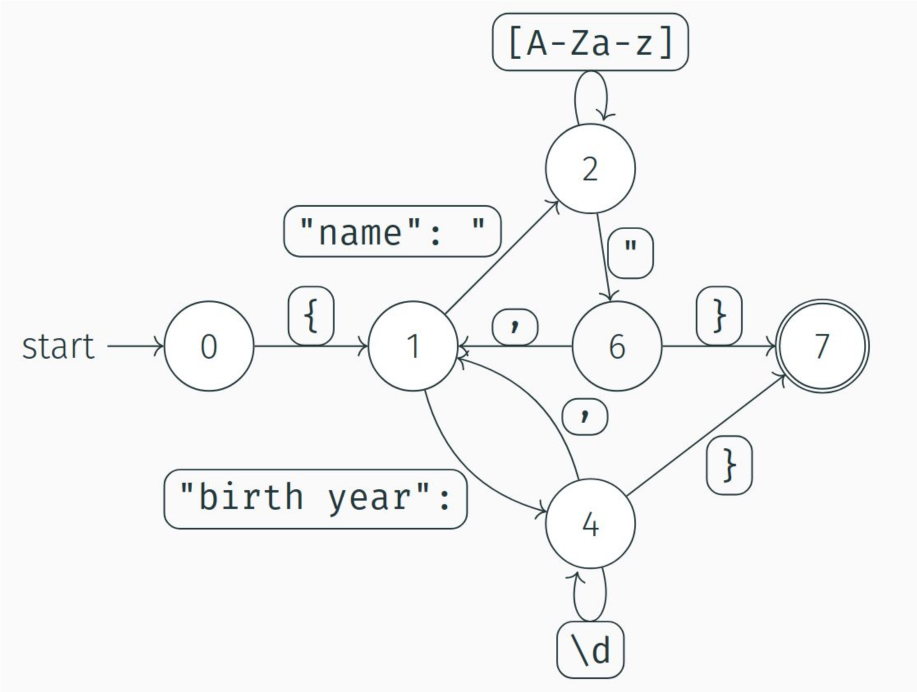
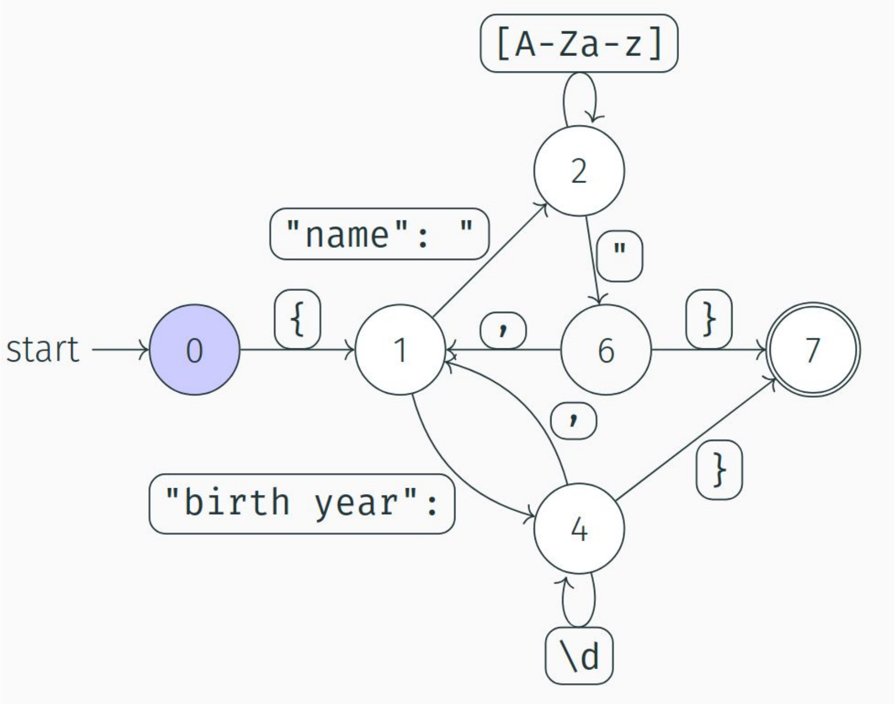



Figure credit: Matt Finlayson

# Templamatic constraints




| Token | Prob. |
|---|---|
| \n | ~~0.36~~ |
| " | ~~0.16~~ |
| { | 0.026 |
| https | ~~0.025~~ |
| ... | ... |

Figure credit: Matt Finlayson

# Unnatural token boundaries

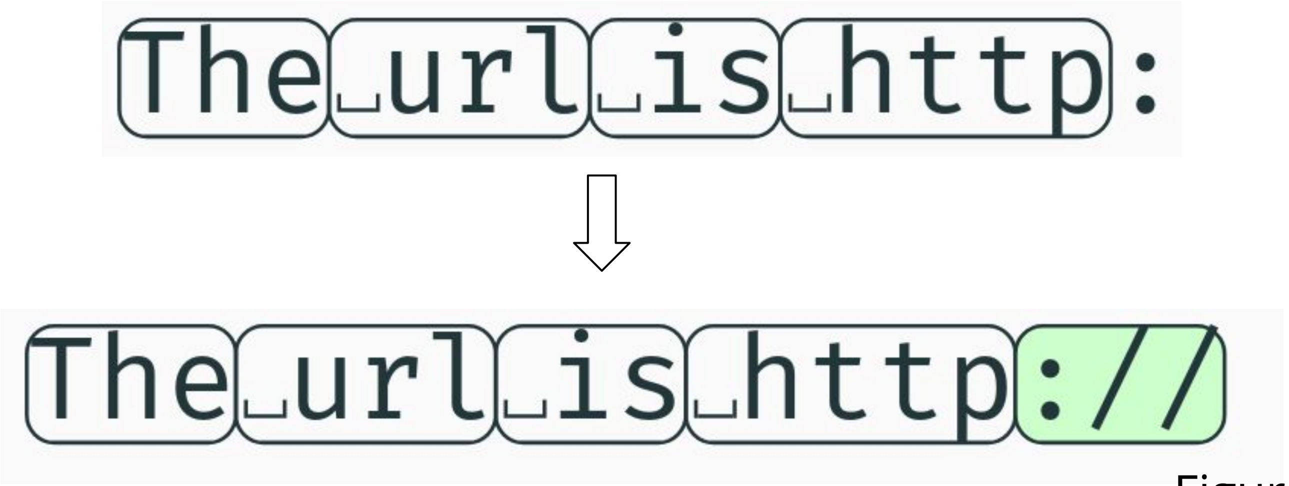
Problem with templatic generation: unnatural token boundaries


vs


Figure credit: Matt Finlayson

# Token healing

We “know” what needs to come next, so roll back a token or two of generation and require that the next token contains the same string



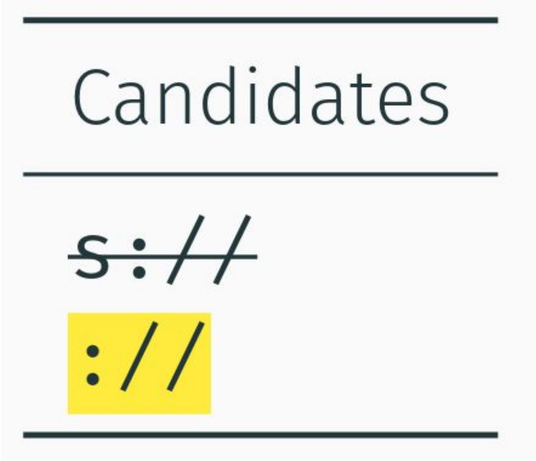




Figure credit: Matt Finlayson

# What does this look like in huggingface?

Templatic constraints with automata

```
[write a LogitsProcessor]
```

Token healing

```
model.generate(token_healing=True)
```

# Semantic constraints

M("Describe a few hobbies I could try to stay in shape.")

I don't want to try climbing!

How do we prevent the model from suggesting climbing?

# Putting instructions in the input isn't enough

What are a few hobbies I could try to stay in shape? I don't want to try climbing.

View other drafts

There are tons of fun hobbies you can try to stay in shape without scaling any cliffs! Here are a few ideas based on different preferences:

**For the outdoor enthusiast:**

- **Hiking or backpacking:** Immerse yourself in nature and get a great workout on scenic trails. Choose beginner-friendly paths at first and gradually increase your difficulty as you build fitness.
- **Kayaking or canoeing:** Enjoy a peaceful paddle on lakes or rivers while getting your upper body and core engaged. Can be social with friends or enjoyed solo.
- **Cycling:** Explore your city, countryside, or trails on two wheels. Choose a road bike for speed, a mountain bike for off-road adventures, or a cruiser for leisurely rides.
- **Rock climbing (indoor):** Okay, so technically we're not going high, but indoor rock climbing offers a thrilling workout challenging your strength, coordination, and problem-solving skills.

# Constrained decoding: logit manipulation

What if we set $P(y_j = \text{"climbing"} \mid X, y_1, \ldots, y_{j-1})$ to be 0?

✔ Easy to implement: just add a big negative to the logit before the softmax!

✘ Bad if there are a lot of synonyms

✘ Bad if the tokens we restrict could be used in “allowed” ways

✘ Bad if we generate other related terms before the restricted term

# Constrained decoding: sample-then-rank (or reject)

Generate a set of sequences S

```
for s_i in S:
  if s_i.is_about_climbing():
    discard(s_i)
```

- Easier to check if the full sequence violates the constraint
- Expensive (i.e. slow), might even need to re-generate

# Constrained decoding: FUDGE (Yang & Klein, 2021)

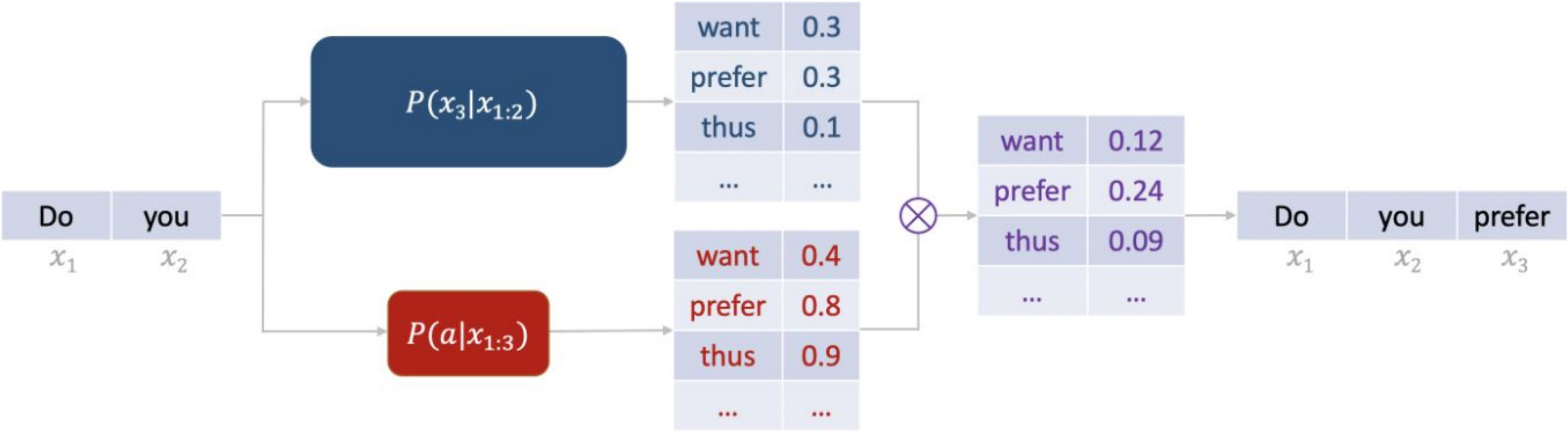



*Figure from Yang & Klein (2021)*

# Constrained decoding via… RLHF?

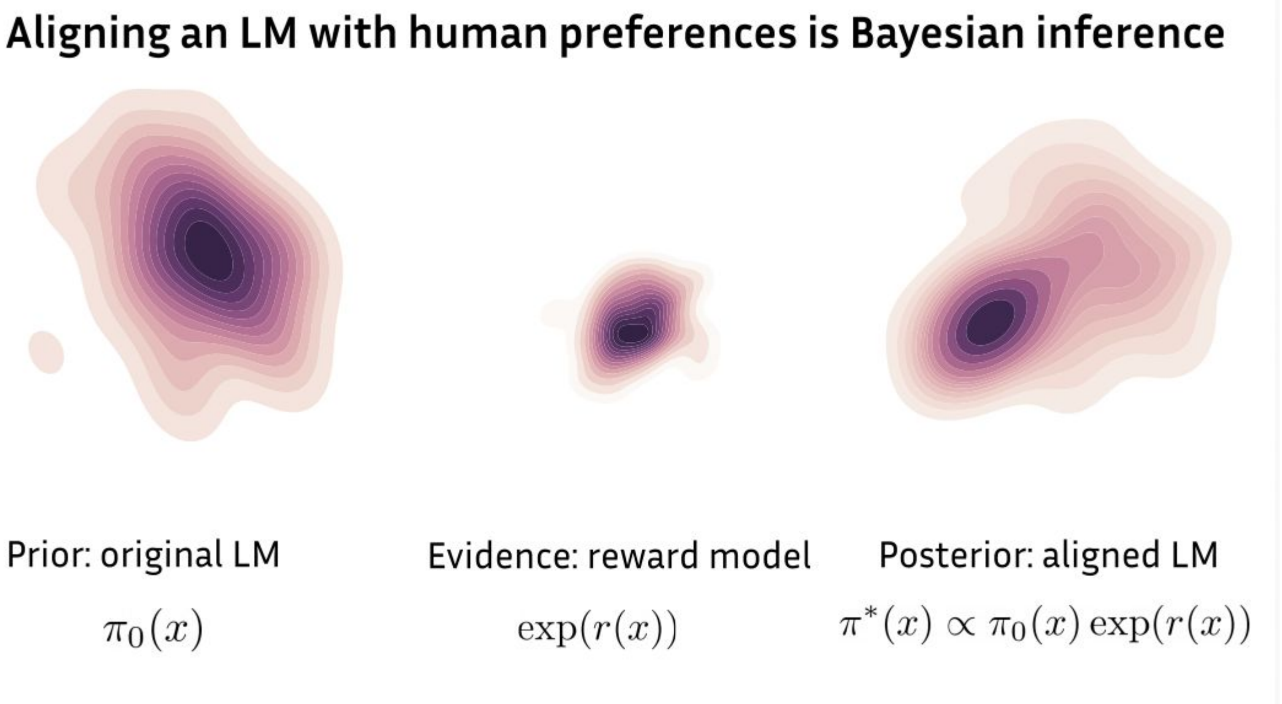
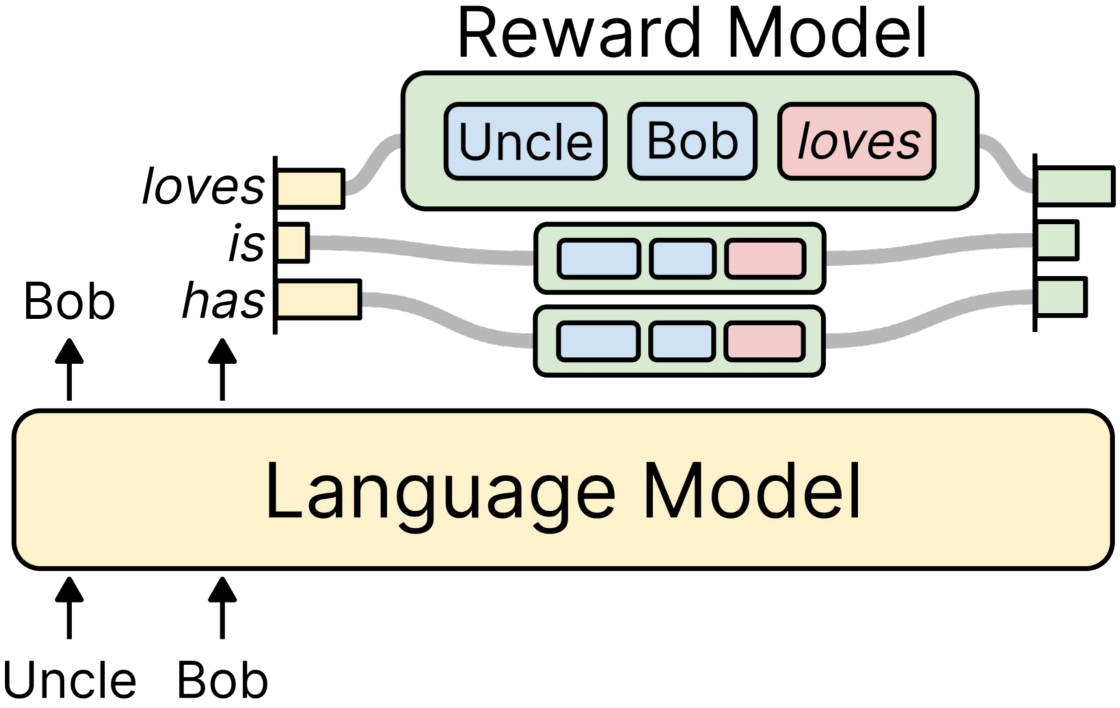



*Figure from Korbak et al (2022)*

# Reward-augmented decoding




Modify probabilities by factoring in the estimated final reward of each sequence

*Figure from Deng & Raffel (2023)*

# What does this look like in huggingface?

For most more complicated methods:

```
[write a LogitsProcessor or use model.forward() and write your
own decoding loop!]
```

# Summary: two levels of decoding

The model provides a distribution P(y | X)

1. **At each decoding step:** choose a function **f(**P(y | X)**)** to manipulate the next-token distribution
2. **Over the full decoding process:** choose a function **g(**s**)** to choose between (full or partial) sequences generated from f(P(y | X))

Not covered here: how do we make these fast?

# Takeaways: decoding methods

**You can use decoding methods to control features of the output**

- Match certain constraints
- Factor in a reward function or data source
- You can do more expensive decoding to compensate for a worse model… up to a point

**Different methods have tradeoffs in quality, diversity, and inference speed**

- Sampling is fast and diverse but can be lower-quality
- More restricted sampling and MAP methods are higher-quality but less diverse
- Adding external scorers can be high quality but slow

**Your responsibility to make design decisions doesn't stop when the model is trained!**
**Letting your libraries pick "sensible defaults" can leave performance on the table.**

Previous: constrained generation

up next:

# Human-in-the-loop decoding

# Human-in-the-loop decoding: interleaved text

Choose when to insert model-generated text versus human continuation

Optionally, edit model-generated text before continuing

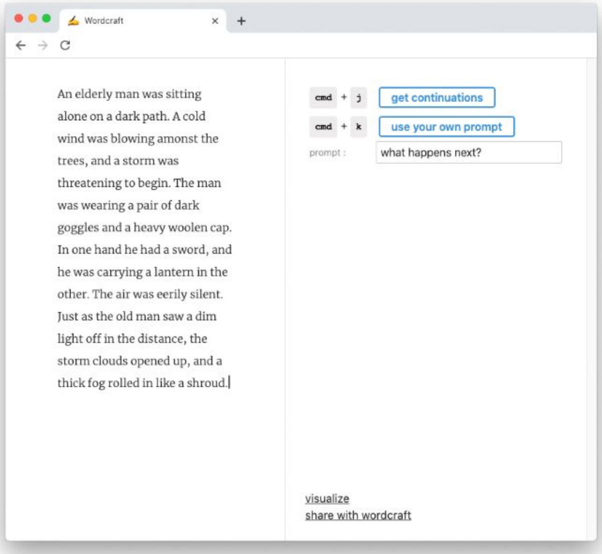



*Figure from Yuan et al (2022)*

# Human-in-the-loop decoding: fine-grained replacement

User chooses the point to intervene, adds additional constraints (e.g. “more descriptive”, “four words”)

This can be accomplished with

- input manipulation
- modeling changes
- decoding changes

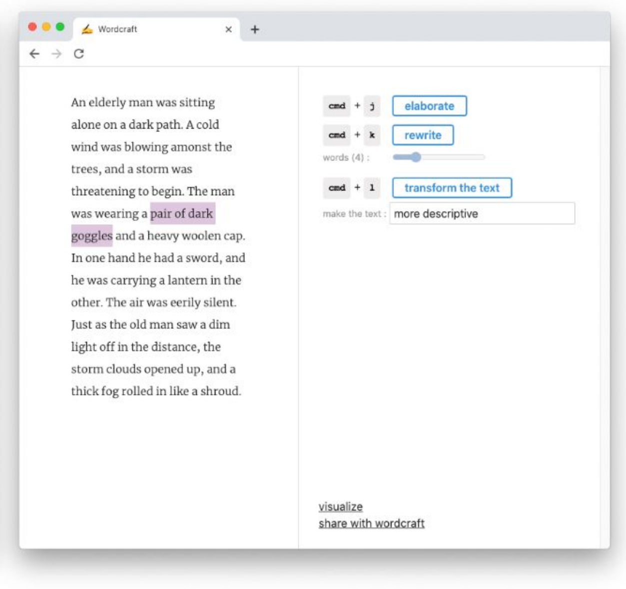
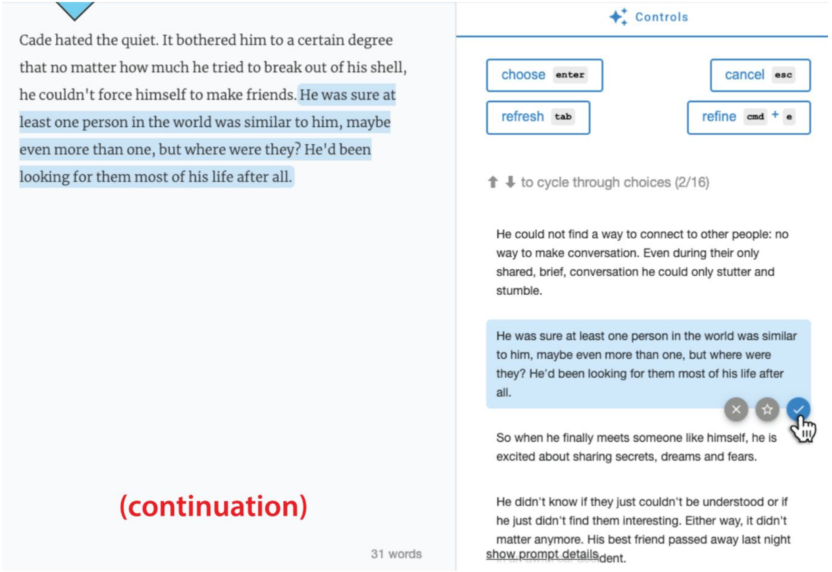


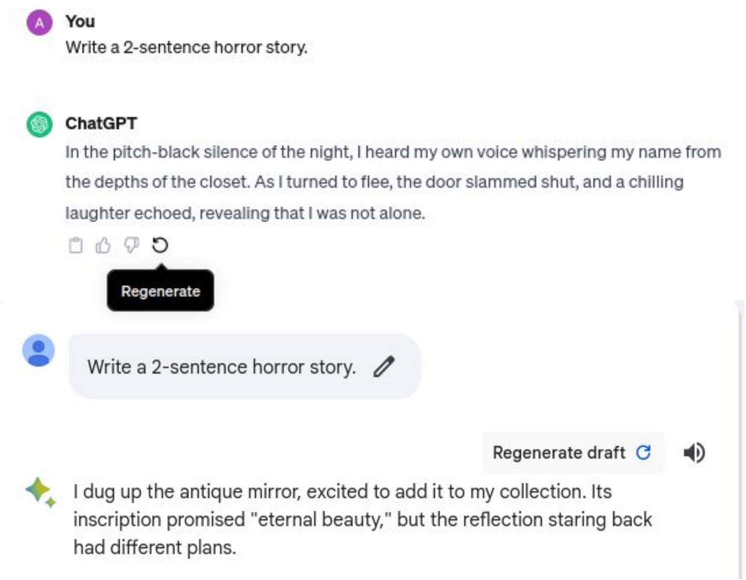

*Figure from Yuan et al (2022)*

# Human-in-the-loop decoding: choosing outputs

## Provide multiple options…




## or the option to regenerate




*Left figure from [Yuan et al (2022)]*

# Model-in-the-loop decoding: Tree of Thought

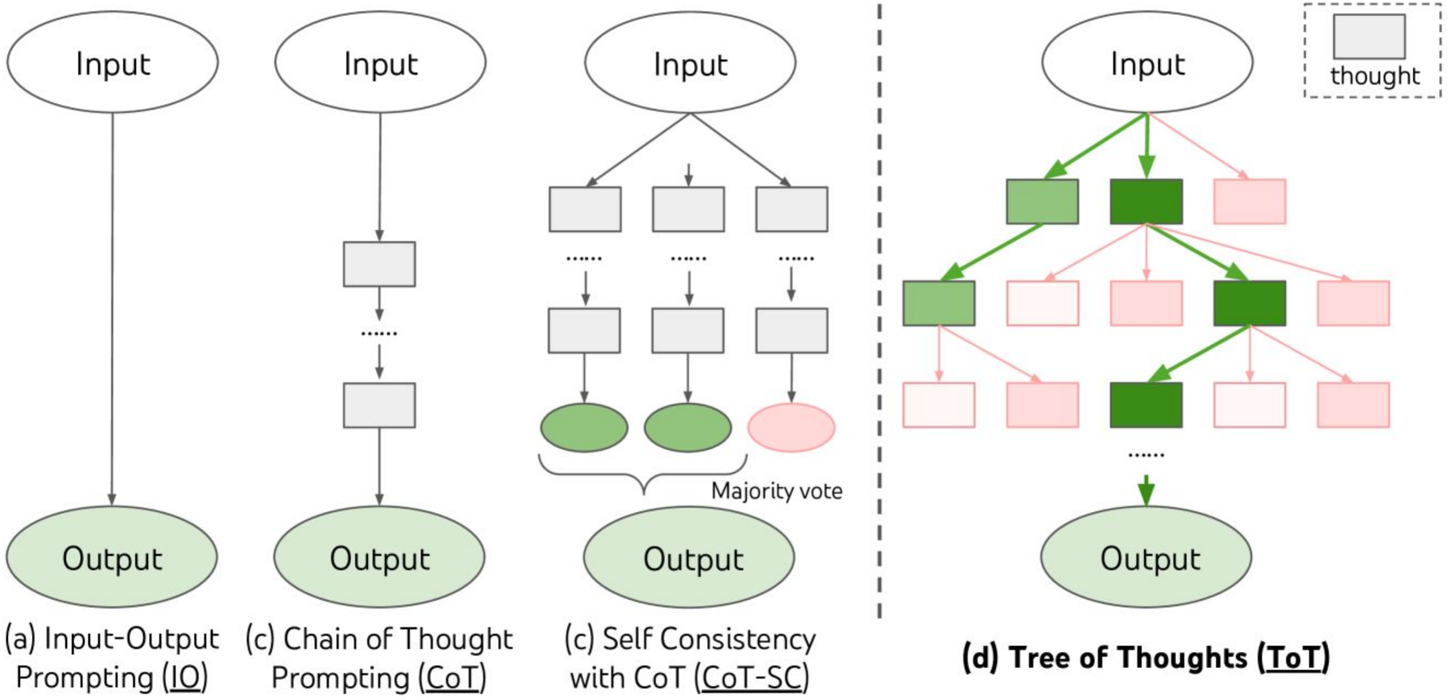


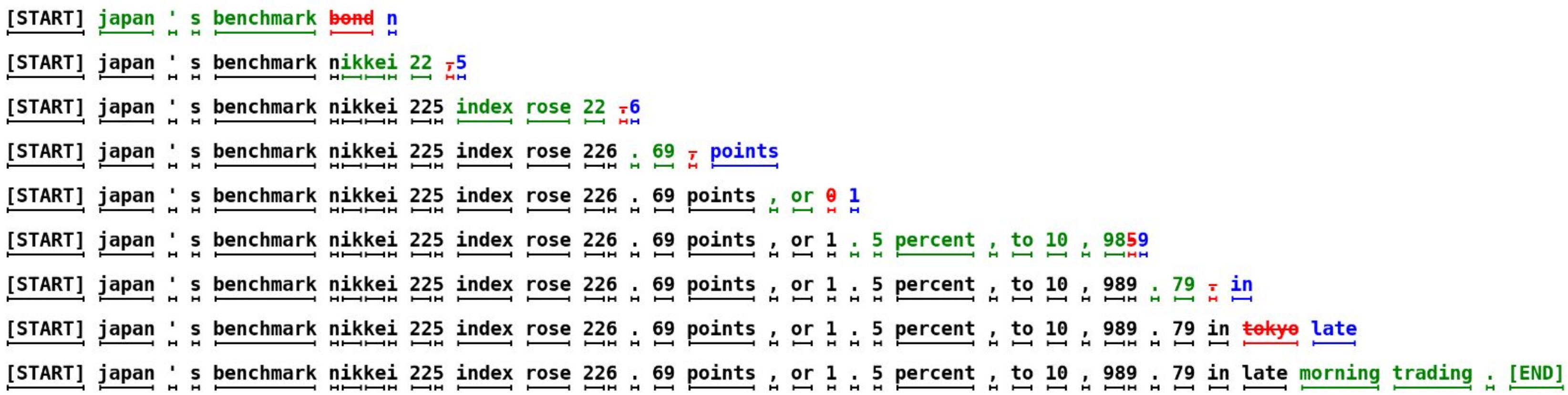

*Figure from Yao et al (2023)*

Previous: human-in-the-loop

up next:

# practical considerations

# Practical considerations: speed (speculative decoding)




Propose candidates with small model, accept/reject candidates with larger model

*Figure from Leviathan et al (2022)*

# Practical considerations: speed (attention sinks)

How do we keep generating quickly when we have more and more context to condition on?

Sliding windows: performance drops quickly

Alternative: attn sinks

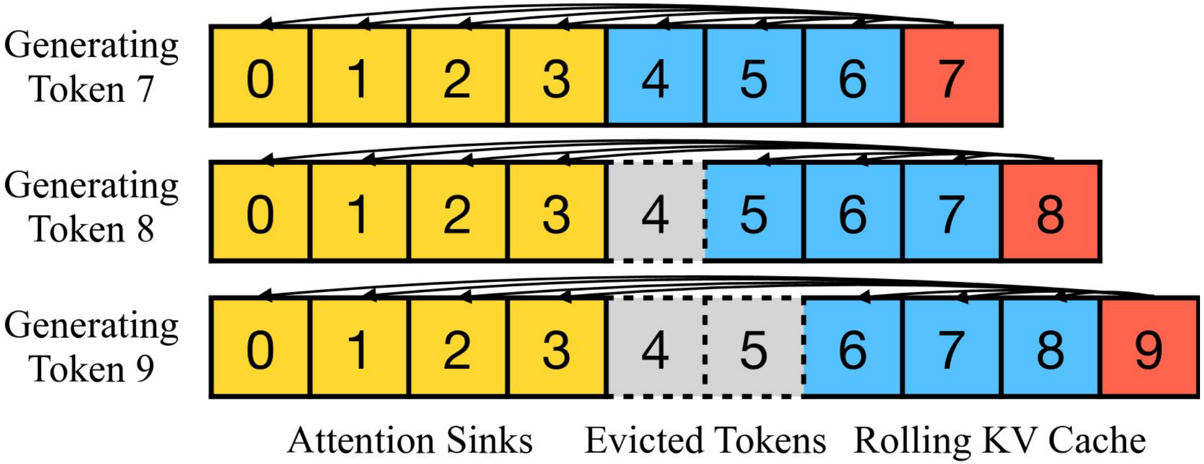



*Figure from Xiao et al (2023)*

# Libraries for decoding (and fast inference)

vLLM

Outlines 〰️

🕺 disco

+ Many methods are implemented in HuggingFace, fairseq2, jax, etc

# Summary: two levels of decoding

The model provides a distribution P(y | X)

1. **At each decoding step:** choose a function **f(**P(y | X)**)** to manipulate the next-token distribution
2. **Over the full decoding process:** choose a function **g(**s**)** to choose between (full or partial) sequences generated from f(P(y | X))

# Takeaways: decoding methods

**You can use decoding methods to control features of the output**

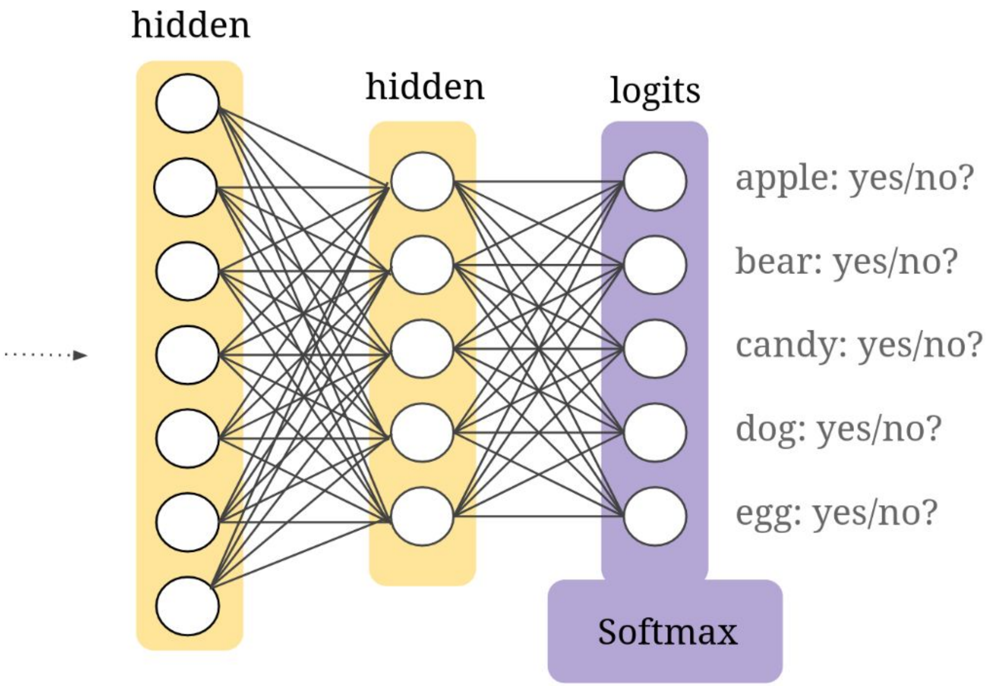
- Match certain constraints
- Factor in a reward function or data source
- You can do more expensive decoding to compensate for a worse model… up to a point

**Different methods have tradeoffs in quality, diversity, and inference speed**

- Sampling is fast and diverse but can be lower-quality
- More restricted sampling and MAP methods are higher-quality but less diverse
- MBR is high quality but slow

**Your responsibility to make design decisions doesn't stop when the model is trained! Letting your libraries pick "sensible defaults" can leave performance on the table.**

# Softmax bottleneck




Softmax of the last layer's output (logits) to get a probability distribution over next tokens

This causes a **softmax bottleneck**– the model is very expressive, but softmax effectively creates a lower-rank output (see Yang, Dai et al (2018))

*Figure from the Google ML course materials*

# Issues with mode-seeking search

Mode-seeking search

# Constrained decoding: A* search

We don't want to just find the highest-probability ("best") path, we want the "best" path that satisfies some conditions

A* and A*-esque algorithms:

$$f(n) = \boxed{g(n)} + \boxed{h(n)}$$

The probability up to token *n*

Heuristic estimation of how likely we are to satisfy constraints with this prefix

# Practical considerations: text detection

Features of generated text vary by decoding method

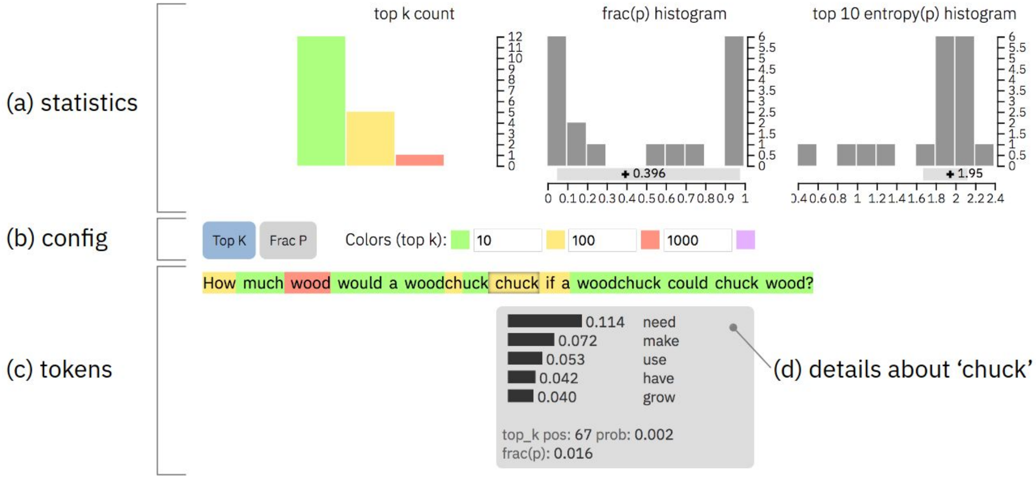



*Figure from Gehrmann et al (2019)*