Lecture 11:
Natural Language Generation (cont.)

Instructor: Xiang Ren

USC CSCI 444 NLP

2026 Spring

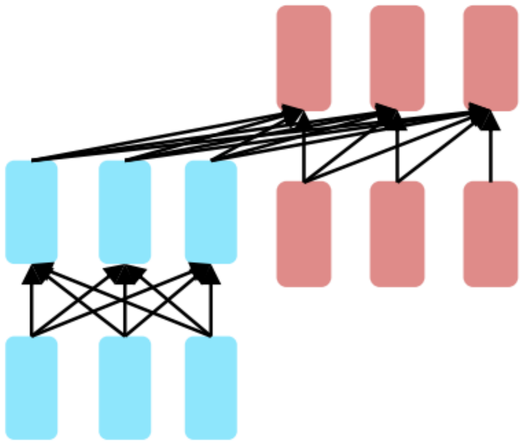
Slides mostly adapted from Dan Jurafsky, some from Mohit Iyyer

Recap: NLG

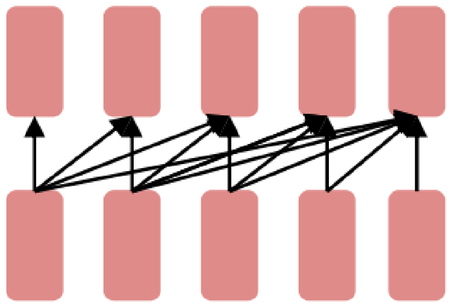
Broad Spectrum of NLG Tasks

Less Open-Ended — More Open-Ended

- Machine Translation
- Summarization
- Task-driven Dialog
- Chitchat Dialog
- Story Generation



Encoder-Decoders



Decoders

Language Generation: Inference/Decoding

- At inference time, our decoding algorithm defines a function to select a token from this distribution:

Inference / Decoding Algorithm

$$\hat{y}_t = g(P(y_t|y_{<t}))$$

- The "obvious" decoding algorithm is to ***greedily*** choose the highest probability next token according to the model at each time step

$$g = \text{arg}max$$

$$\hat{y}_t = \text{arg}\underset{w \in V}{max}(P(y_t = w|y_{<t}))$$

Greedy Decoding: Issues

- Greedy decoding has no wiggle room for errors!
  - Input: the green witch arrived
    - Output: llego
    - Output: llego la
    - Output: llego la verde
- How to fix this?
  - Need a lookahead strategy / longer-term planning

Exhaustive Search Decoding

- Ideally, we want to find a (length T) translation y that maximizes

$$P(y|x) = P(y_1|x)\,P(y_2|y_1,x)\,P(y_3|y_1,y_2,x)\ldots,P(y_T|y_1,\ldots,y_{T-1},x)$$
$$= \prod_{t=1}^{T} P(y_t|y_1,\ldots,y_{t-1},x)$$

- We could try computing all possible sequences y
  - This means that on each step t of the decoder, we're tracking V^t possible partial translations, where V is vocab size
  - This $O(V^T)$ complexity is far too expensive!

Beam Search Decoding

- Core idea: On each step of decoder, keep track of the k most probable partial translations (which we call hypotheses)
  - k is the beam size (in practice around 5 to 10, in NMT)
- A hypothesis has a score which is its log probability:

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

  - Scores are all negative, and higher score is better
  - We search for high-scoring hypotheses, tracking top k on each step
- Beam search is not guaranteed to find optimal solution
- But much more efficient than exhaustive search!

Beam Search Decoding: Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

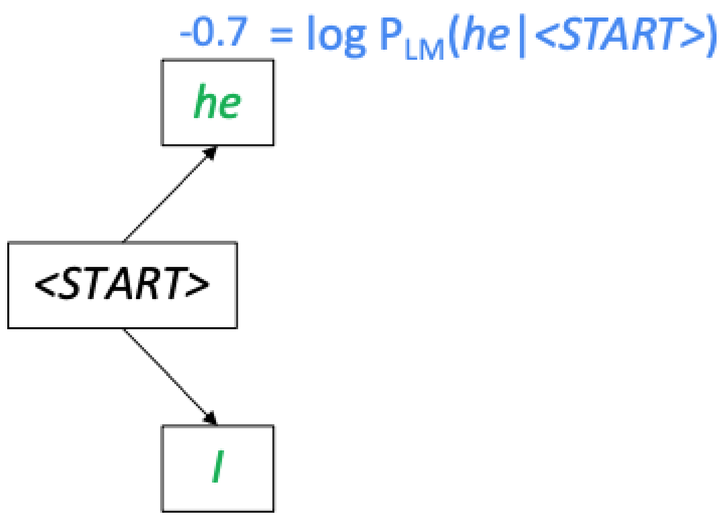
<START>

Calculate prob dist of next word

Slide credit: Chris Manning

Beam Search Decoding: Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

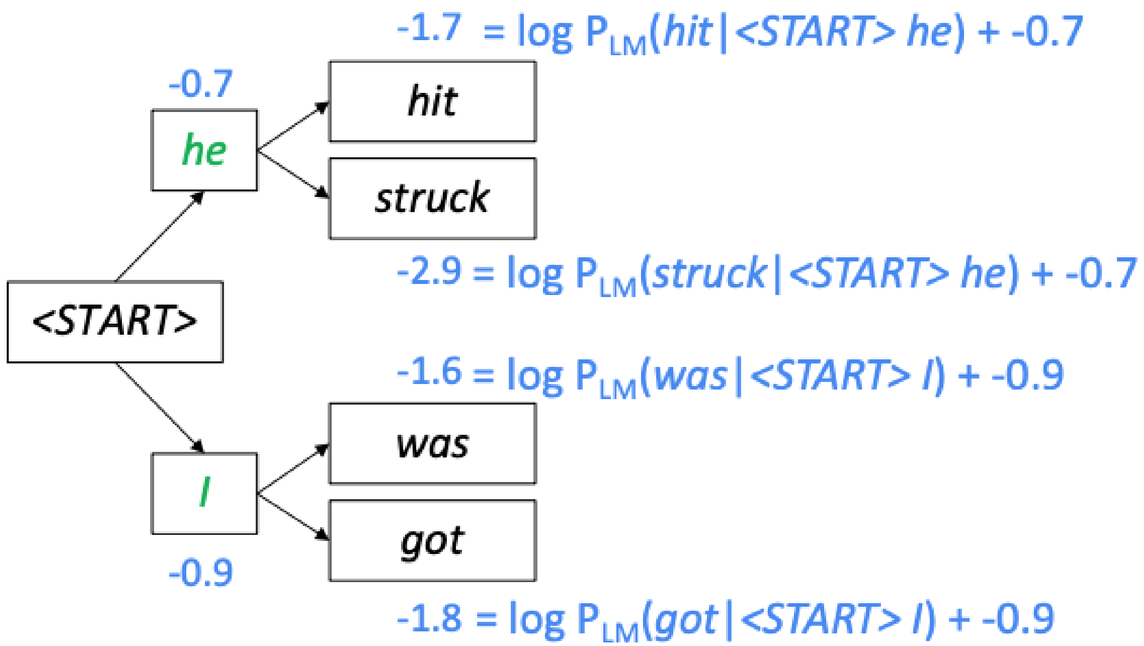


Take top *k* words and compute scores

Slide credit: Chris Manning

Beam Search Decoding: Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

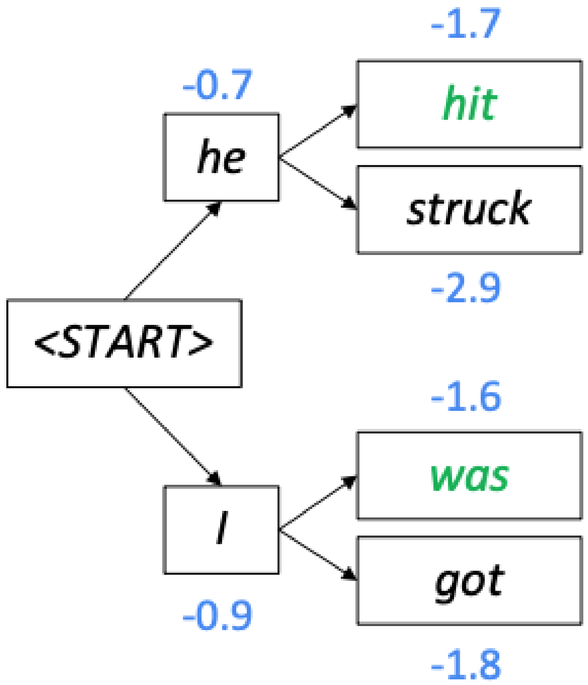


For each of the *k* hypotheses, find top *k* next words and calculate scores

Slide credit: Chris Manning

Beam Search Decoding: Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

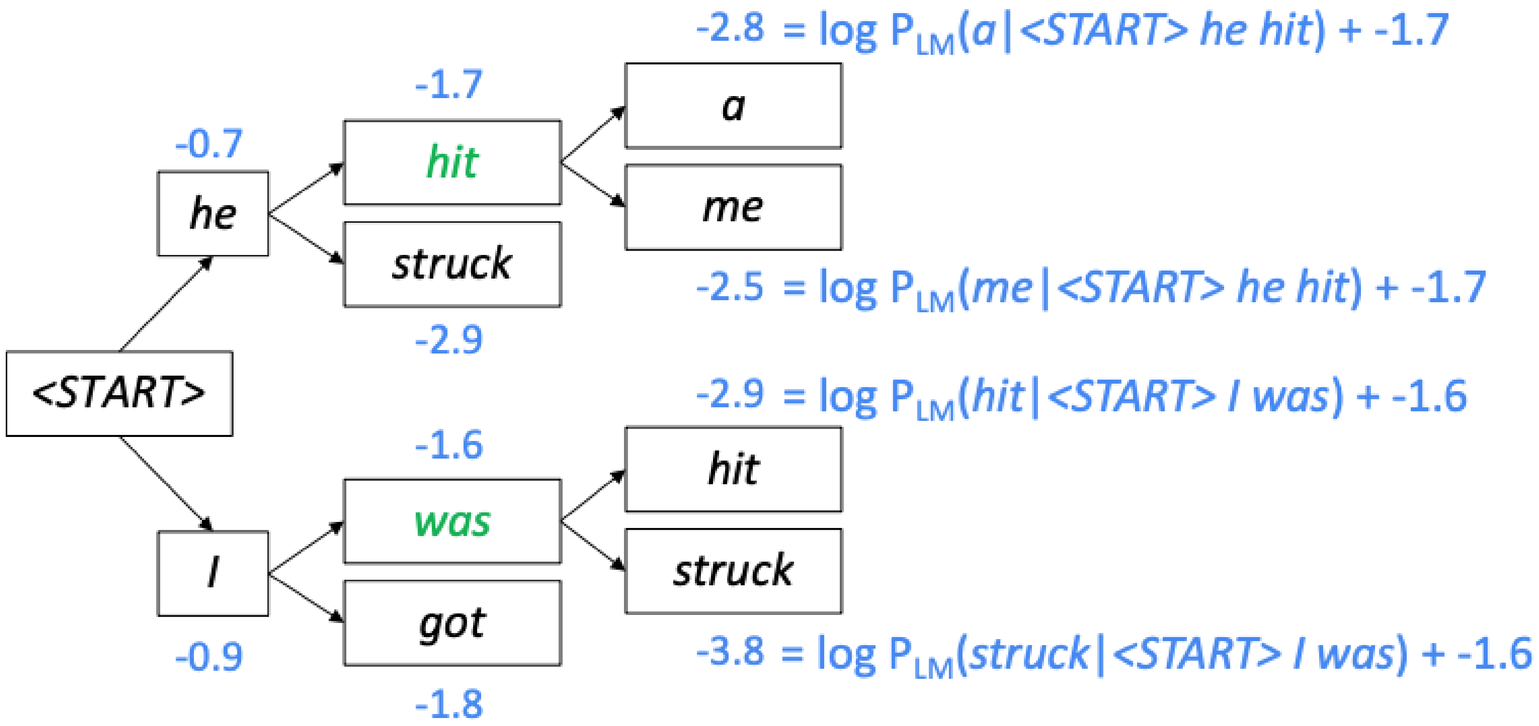


Of these k^2 hypotheses, just keep k with highest scores

Slide credit: Chris Manning

Beam Search Decoding: Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

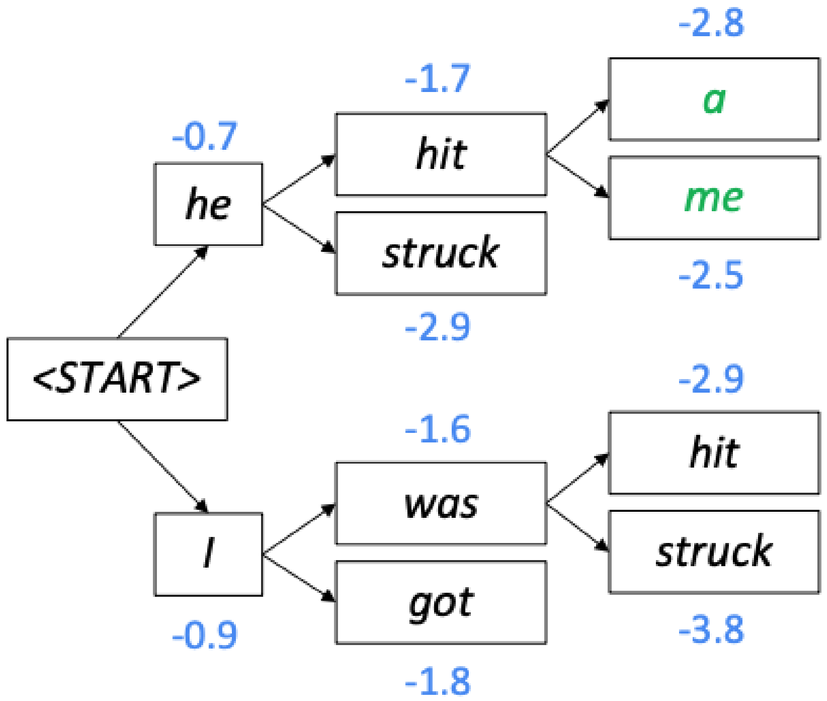


For each of the *k* hypotheses, find top *k* next words and calculate scores

de credit: Chris Manning

Beam Search Decoding: Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

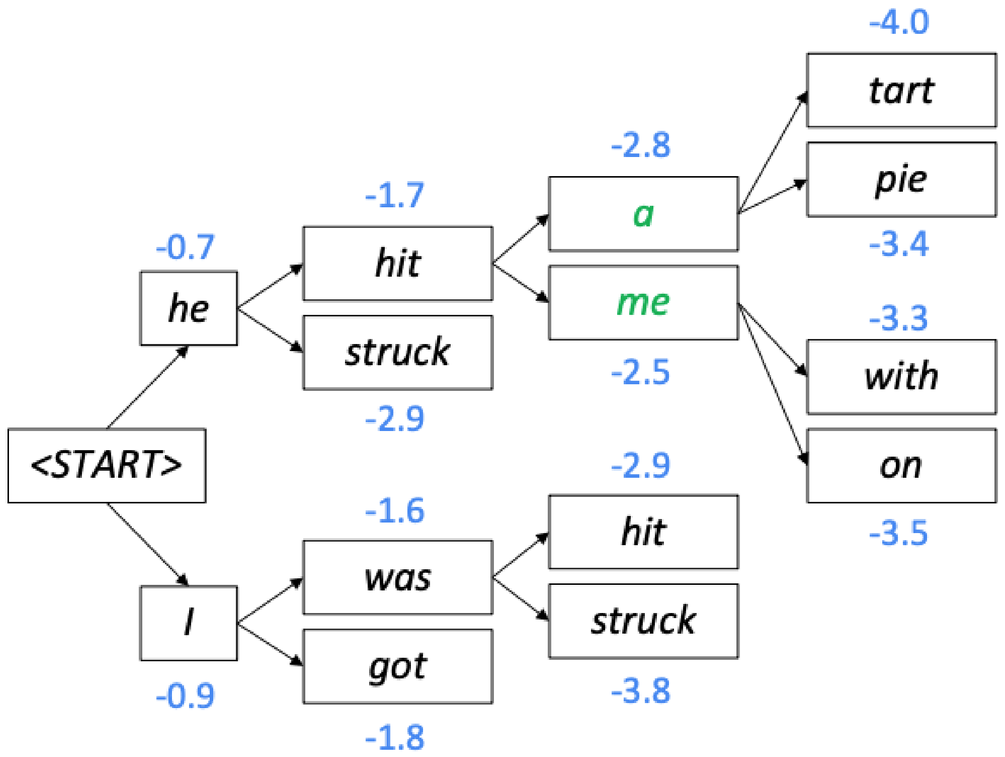


Of these k^2 hypotheses, just keep k with highest scores

Slide credit: Chris Manning

Beam Search Decoding: Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

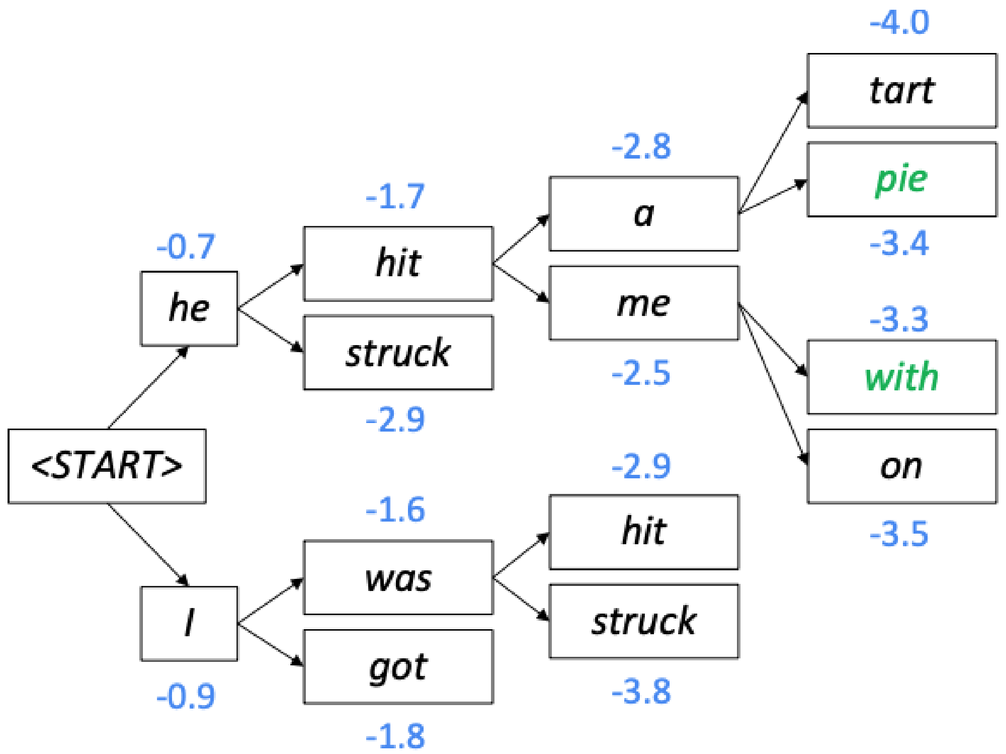


For each of the *k* hypotheses, find top *k* next words and calculate scores

Slide credit: Chris Manning

Beam Search Decoding: Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

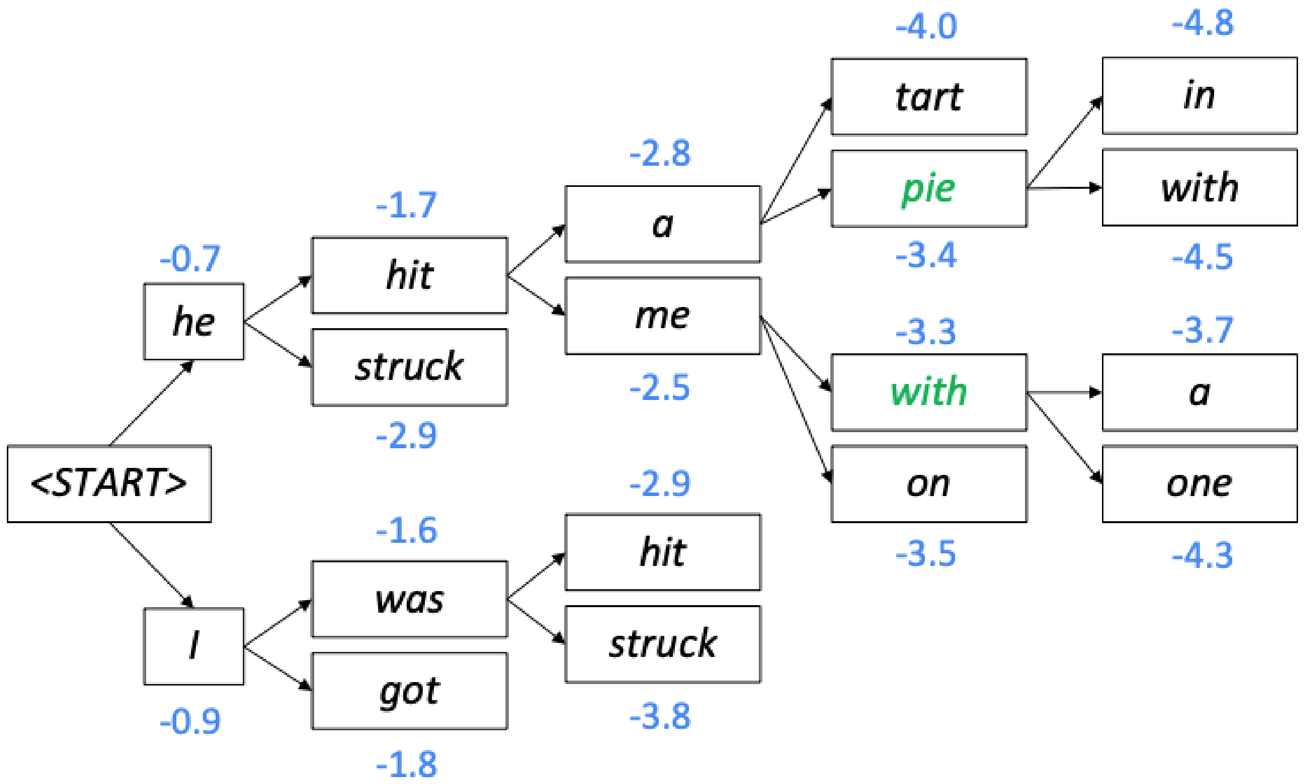


Of these k^2 hypotheses, just keep k with highest scores

Slide credit: Chris Manning

Beam Search Decoding: Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

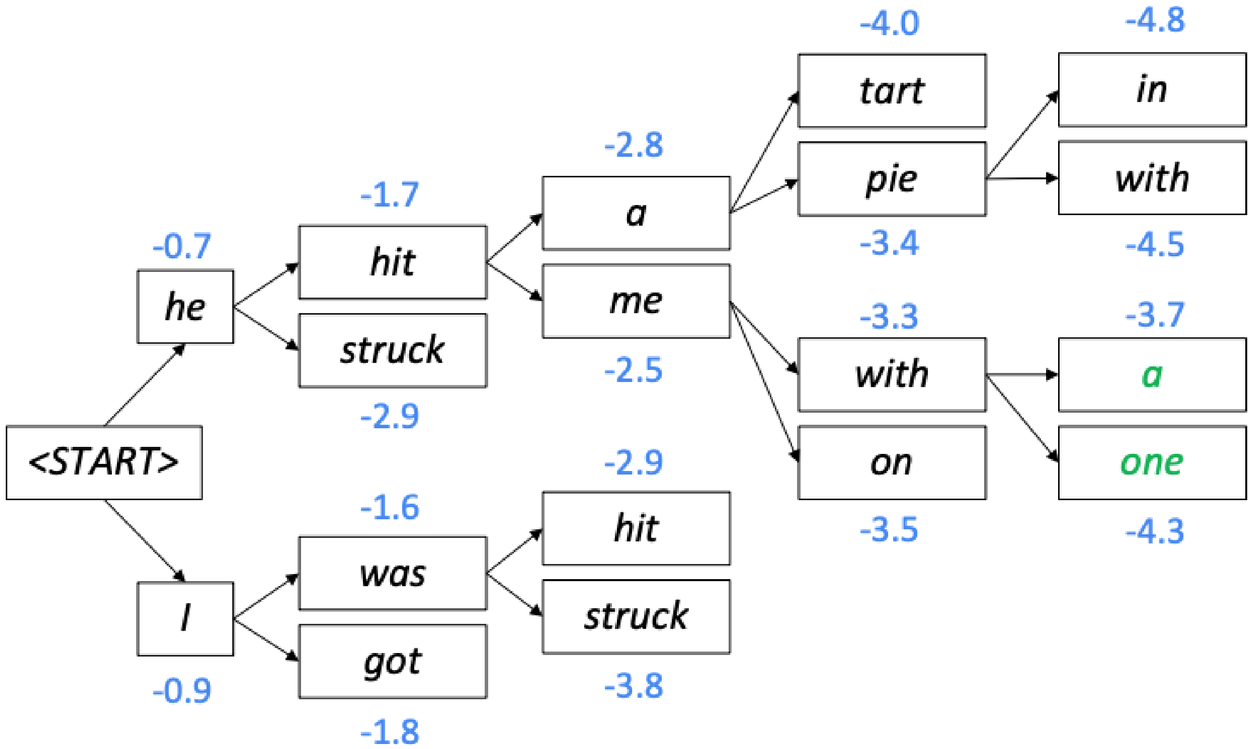


For each of the *k* hypotheses, find top *k* next words and calculate scores

Slide credit: Chris Manning

Beam Search Decoding: Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

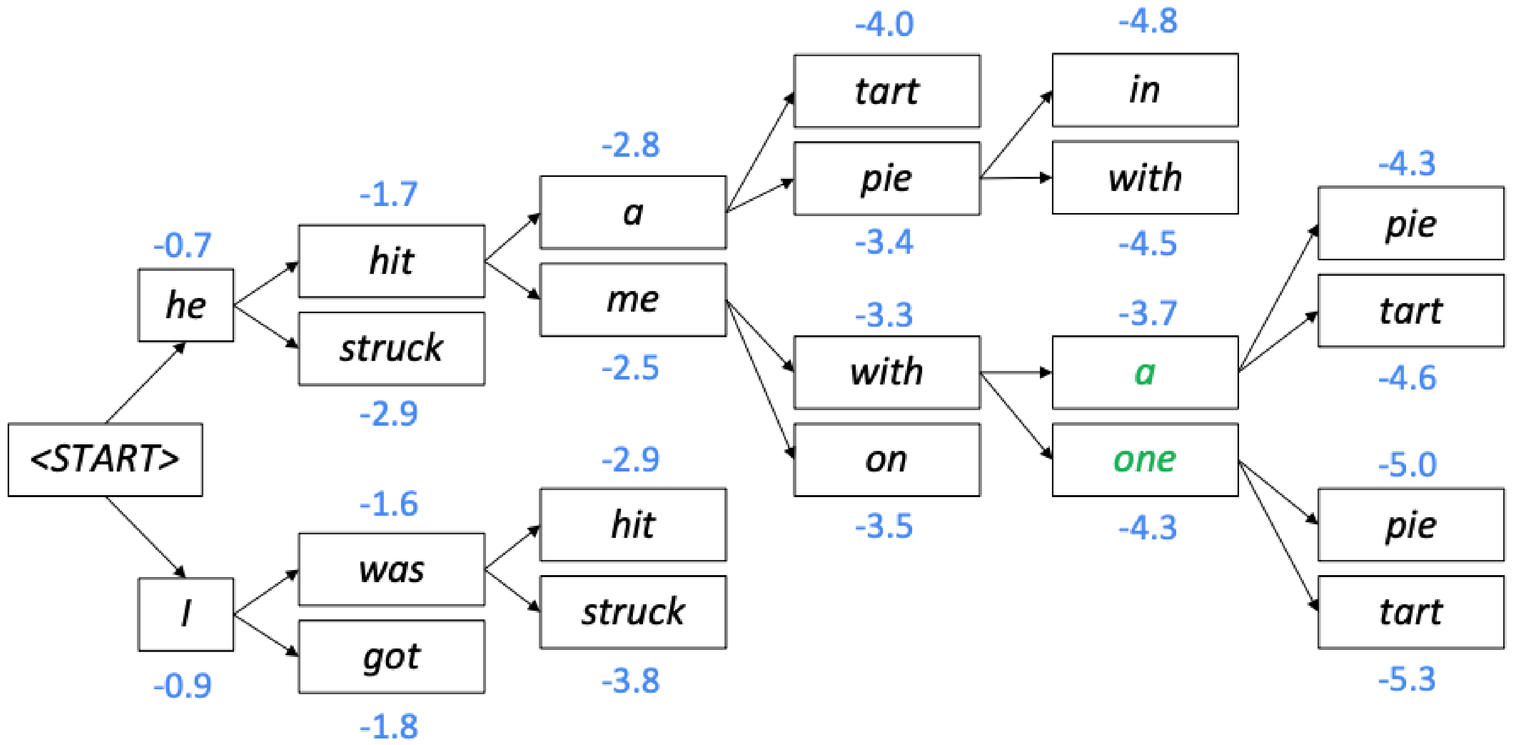


Of these k^2 hypotheses, just keep k with highest scores

Slide credit: Chris Manning

Beam Search Decoding: Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

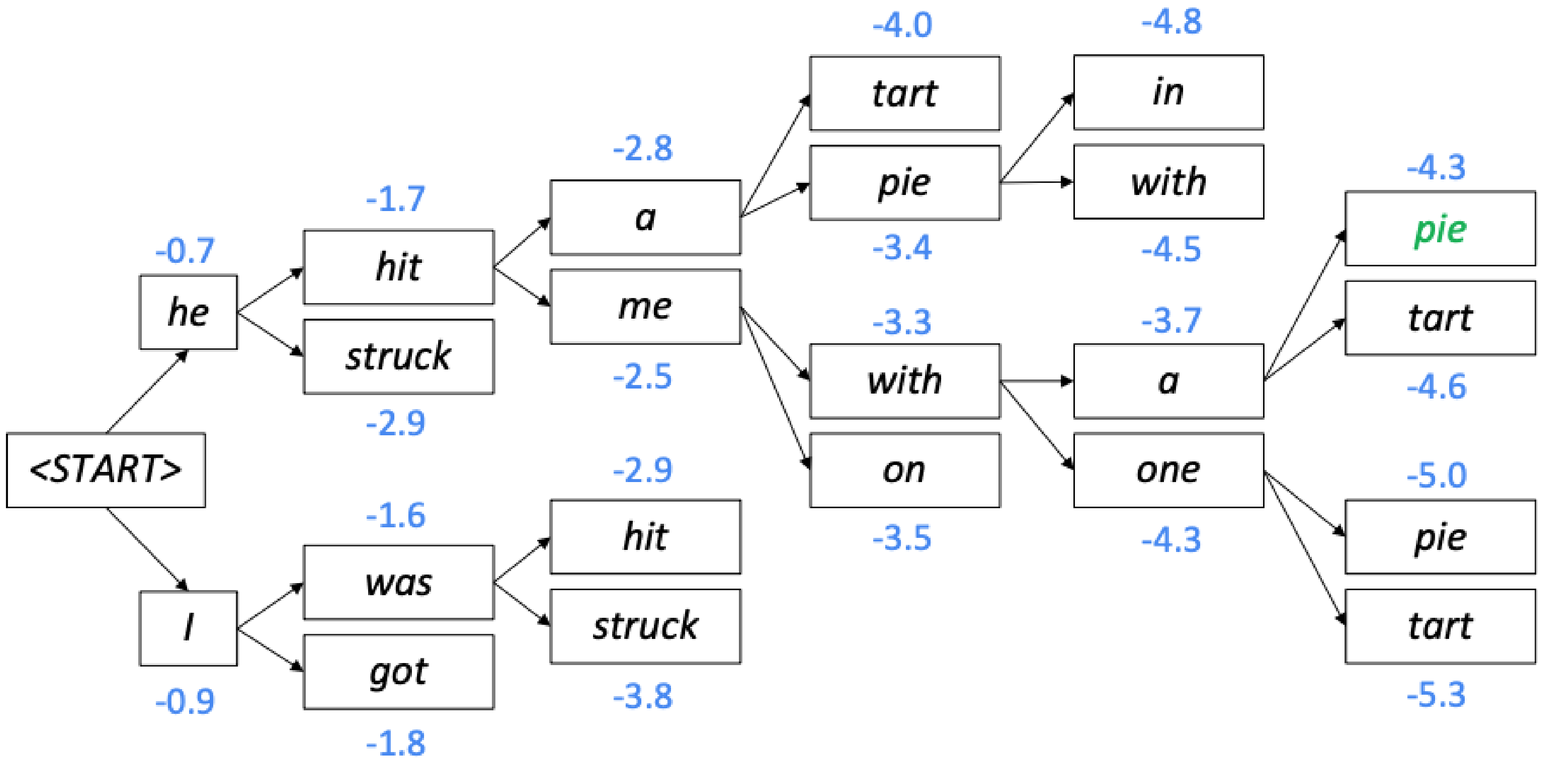


For each of the *k* hypotheses, find top *k* next words and calculate scores

Slide credit: Chris Manning

Beam Search Decoding: Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

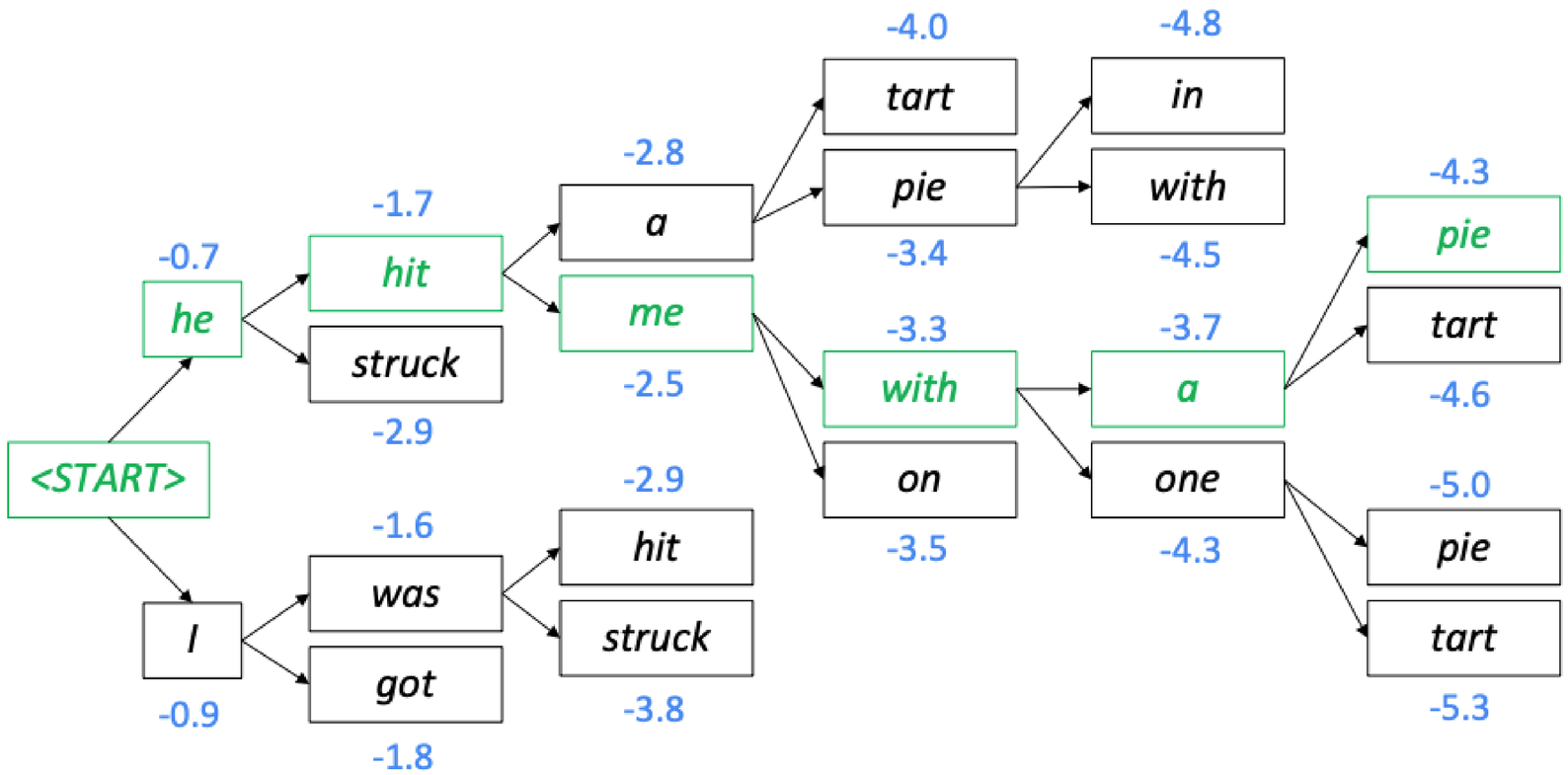


This is the top-scoring hypothesis!

Slide credit: Chris Manning

Beam Search Decoding: Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Backtrack to obtain the full hypothesis

Slide credit: Chris Manning

Beam Search Decoding: Stopping Criterion

- Greedy Decoding is done until the model produces an </s> token
  - For e.g. <s> he hit me with a pie </s>
- In Beam Search Decoding, different hypotheses may produce </s> tokens at different time steps
  - When a hypothesis produces </s>, that hypothesis is complete.
  - Place it aside and continue exploring other hypotheses via beam search.
- Usually we continue beam search until:
  - We reach time step T (where T is some pre-defined cutoff), or
  - We have at least n completed hypotheses (where n is pre-defined cutoff)

Beam Search Decoding: Parting Thoughts

- We have our list of completed hypotheses. Now how to select top one?
- Each hypothesis $y_1, \ldots, y_t$ on our list has a score

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

- Problem with this: longer hypotheses have lower score

- Fix: Normalize by length. Use this to select top one instead

$$\frac{1}{t} \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

But this is expensive!

Issues with Maximization-Based Decoding

- Either greedy or beam search
- Beam search can be more effective with large beam width, but also more expensive
- Another key issue:

Generation can be bland or repetitive (also called degenerate)

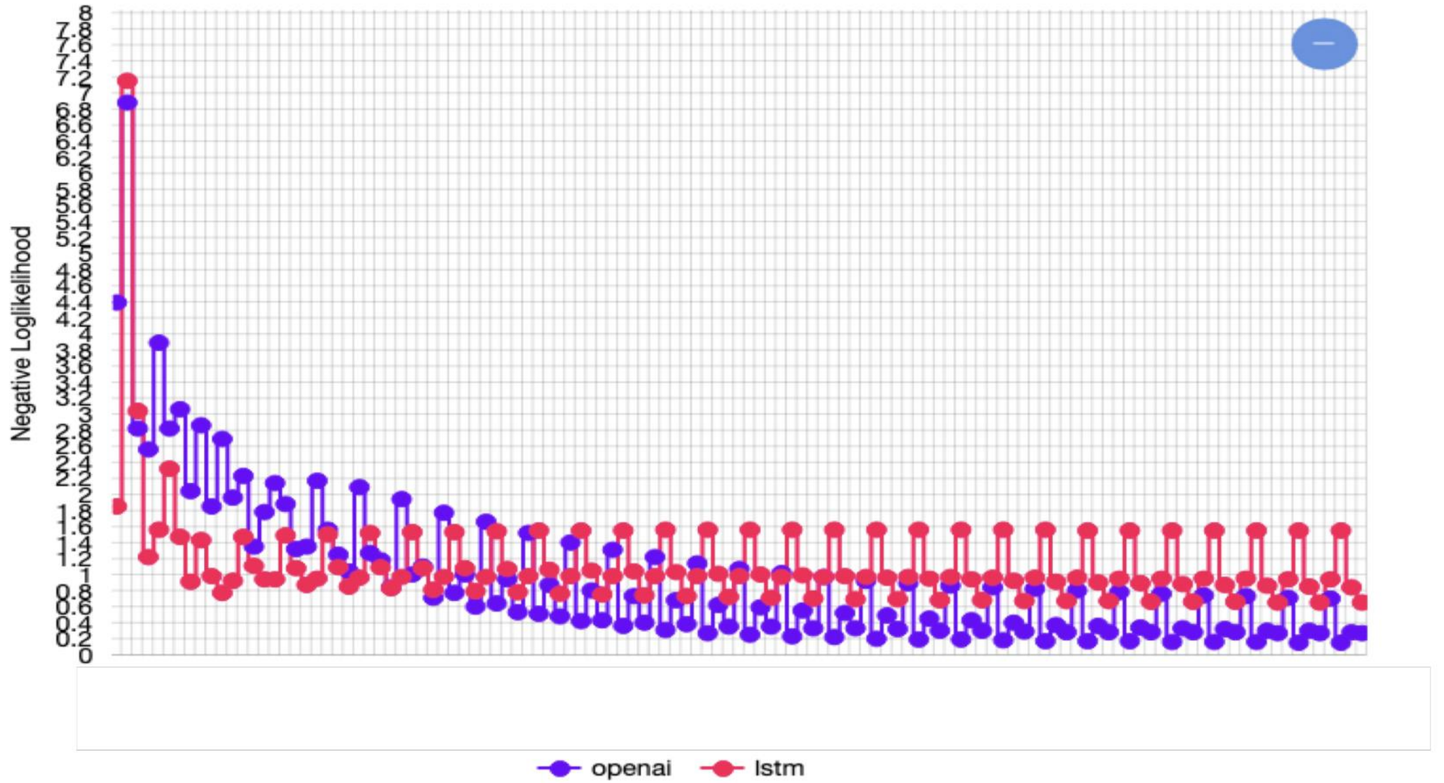
Context: In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

Continuation: The study, published in the Proceedings of the National Academy of Sciences of the United States of America (PNAS), was conducted by researchers from the **Universidad Nacional Autónoma de México (UNAM)** and **the Universidad Nacional Autónoma de México (UNAM/Universidad Nacional Autónoma de México/ Universidad Nacional Autónoma de México/ Universidad Nacional Autónoma de México/ Universidad Nacional Autónoma de México…**

Holtzmann et al., 2020

Degenerate Outputs

I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired.

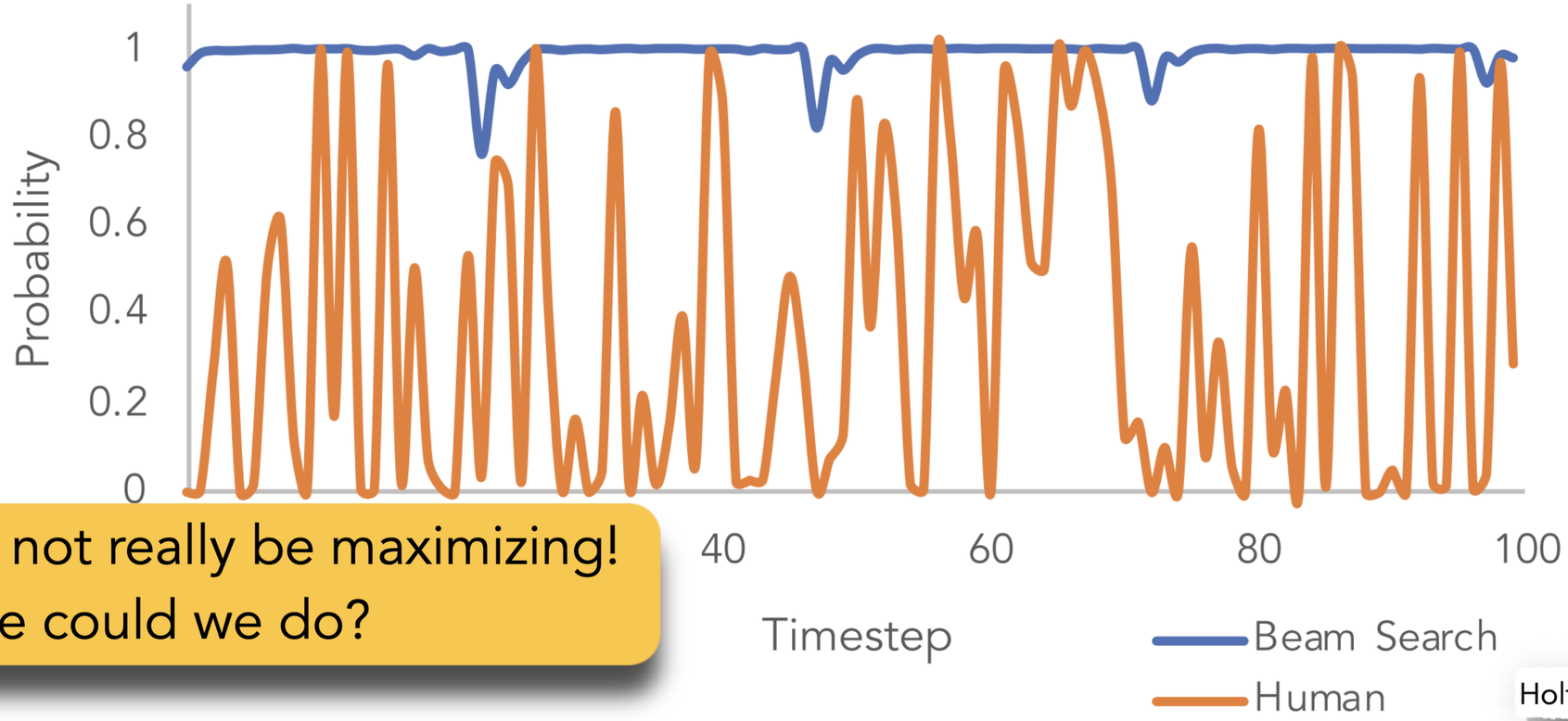


Holtzmann et al., 2020

Scale doesn't solve this problem: even a 175 billion parameter LM still repeats when we decode for the most likely string.

Why does repetition happen?

- Probability amplification due to maximization based decoding
- Generation fails to match the uncertainty distribution for human written text

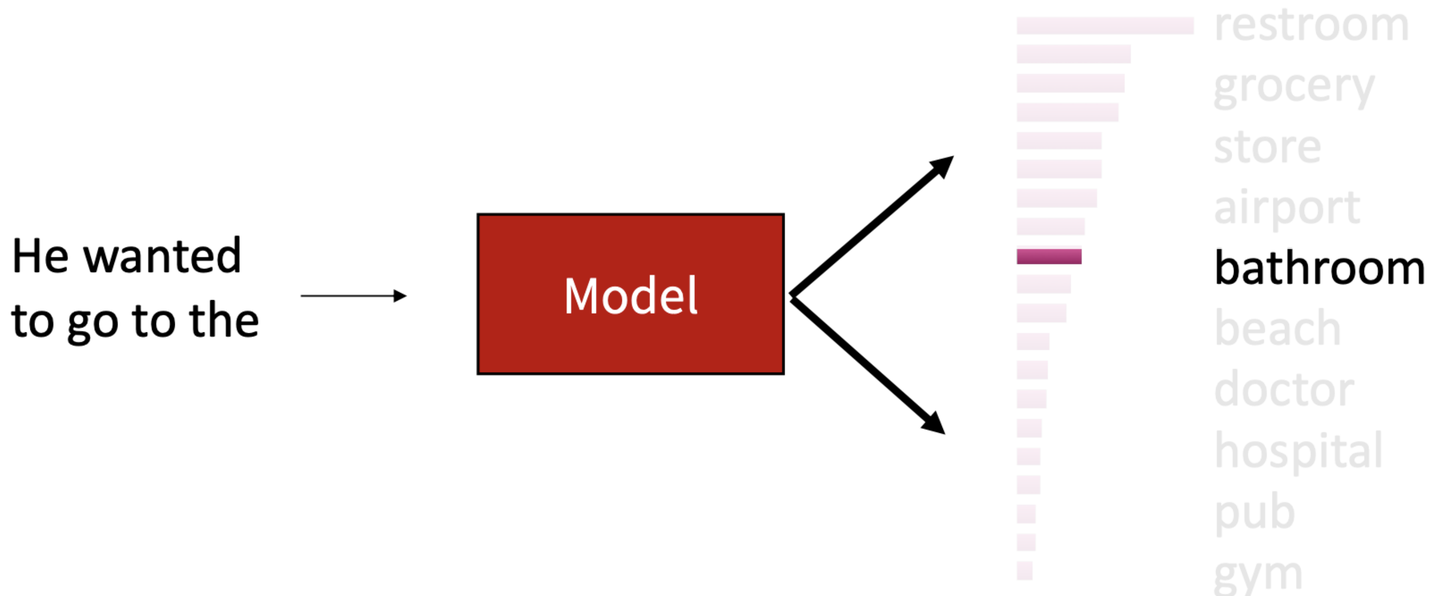


Holtzmann et al., 2020

Perhaps we should not really be maximizing!
What else could we do?

Solution: Don't Maximize, Pick a Sample

- Sample a token from the distribution of tokens.
- But this is not a random sample, it is a sample for the learned model distribution
  - Respects the probabilities, without going just for the maximum probability option
  - Many good options which are not the maximum probability!



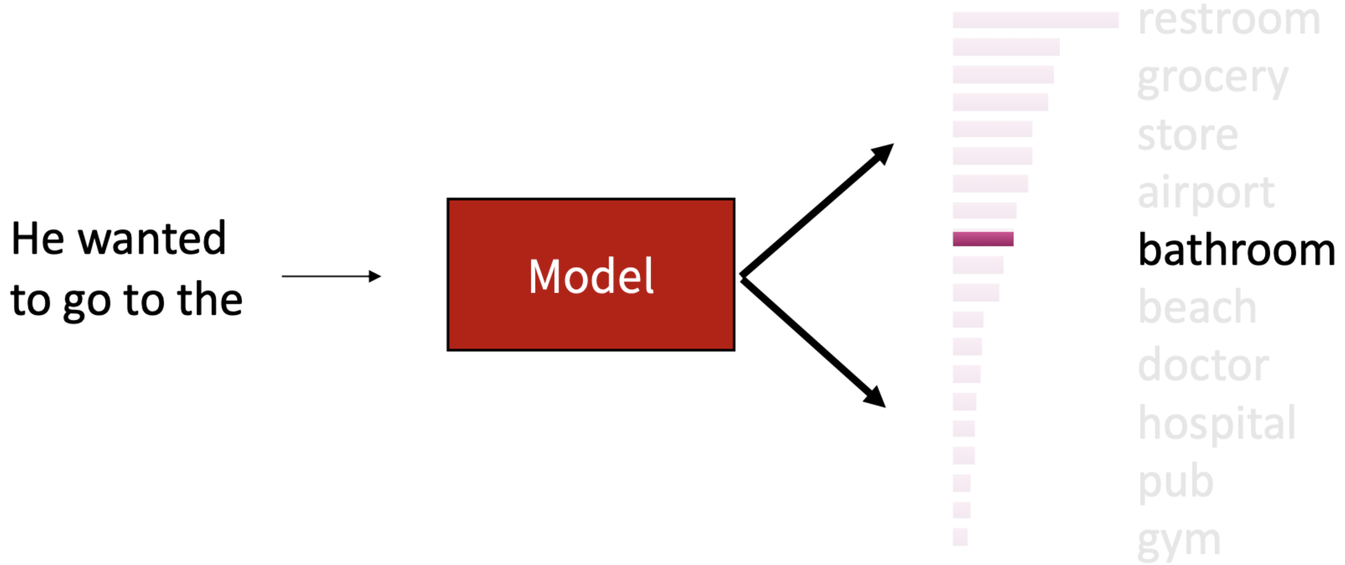
Modern Generation: Sampling

Pure Sampling

- Sample directly from P_t

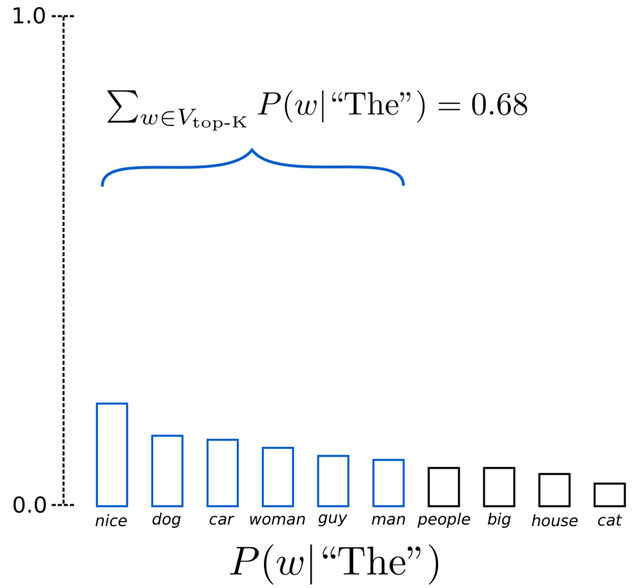
$$y_t \sim P_t(w) = \frac{\exp(S_w)}{\sum_{v \in V} \exp(S_v)}$$

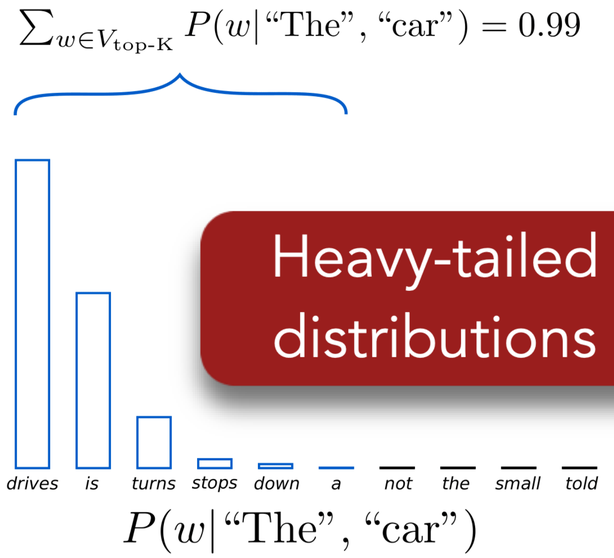
- Still has access to the entire vocabulary
- But if the model distributions are of low quality, generations will be of low quality as well
- Often results in ill-formed generations
  - No guarantee of fluency



Top-K Sampling

- Problem: Pure sampling makes every token in the vocabulary an option
  - Even if most of the probability mass in the distribution is over a limited set of options, the tail of the distribution could be very long and in aggregate have considerable mass
  - Many tokens are probably really wrong in the current context. Yet, we give them individually a tiny chance to be selected.
  - But because there are many of them, we still give them as a group a high chance to be selected.
- Solution: Top-K sampling
  - Only sample from the top K tokens in the probability distribution





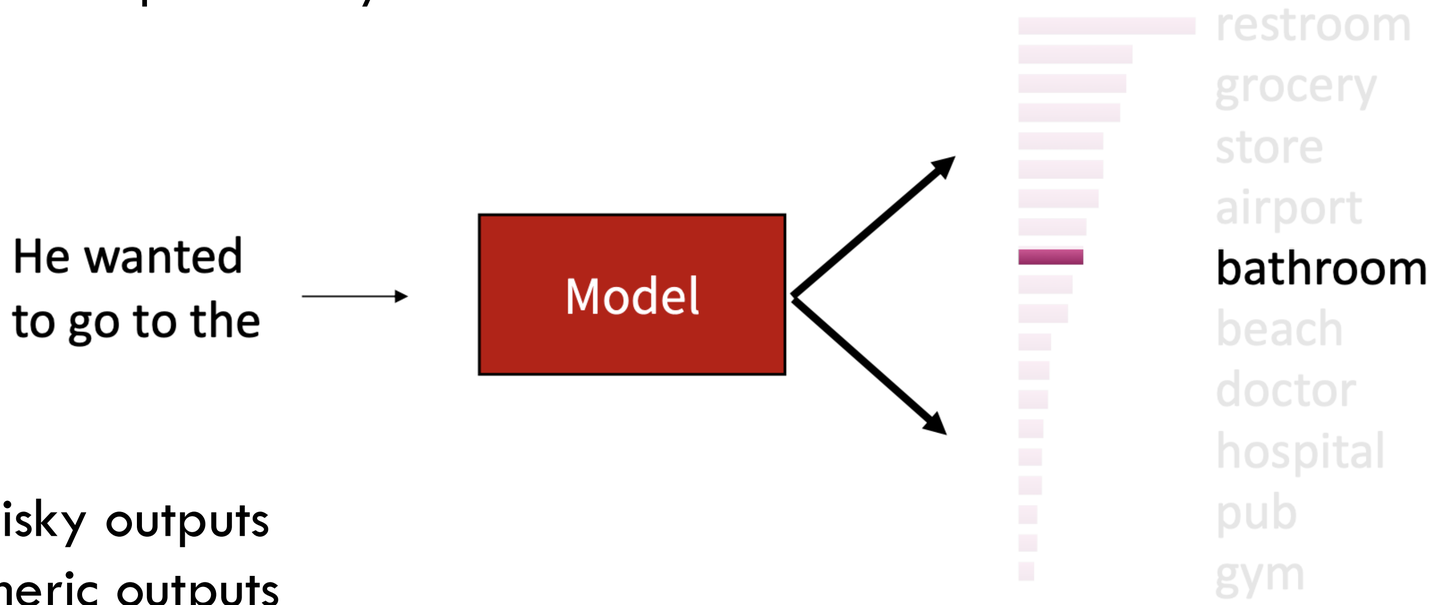
Heavy-tailed distributions

Fan et al., ACL 2018; Holtzman et al., ACL 2018

Image Source: Huggingface

Top-K Sampling: Value of K

- Solution: Top-K sampling
  - Only sample from the top K tokens in the probability distribution
  - Common values are $K = 50$



- Increase K yields more diverse, but risky outputs
- Decrease K yields more safe but generic outputs

Top-K Sampling: Issues

Top-K sampling can cut off too quickly

Top-K sampling can also cut off too slowly!

We can do better than having one-size-fits-all: a fixed K for all contexts

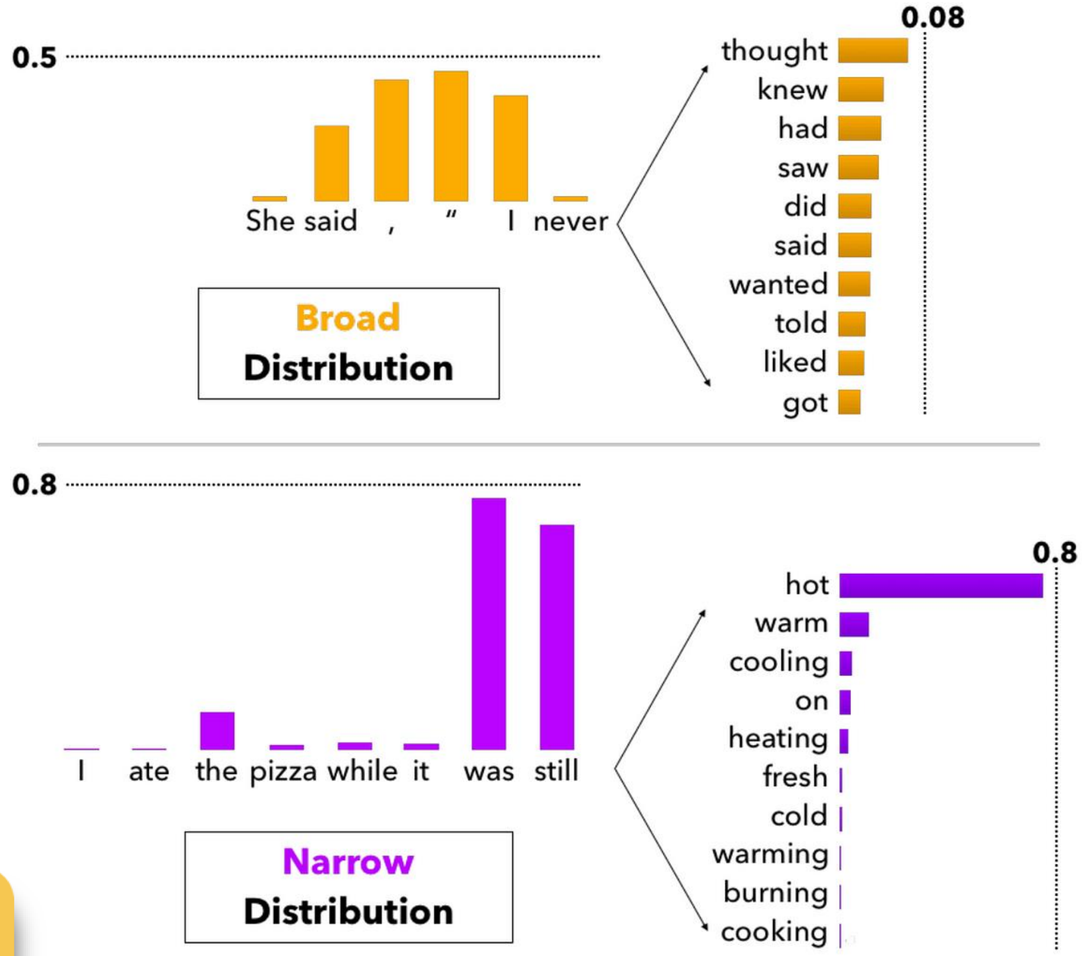


Image Source: Holtzmann et al., 2019

Modern Decoding: Nucleus Sampling

- Problem: The probability distributions we sample from are dynamic
  - When the distribution P_t is flatter, a limited K removes many viable options
  - When the distribution P_t is peakier, a high K allows for too many options to have a chance of being selected

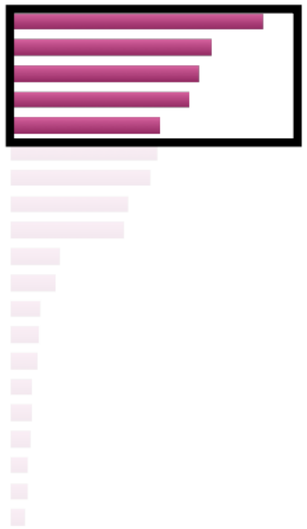
- Solution: Nucleus Sampling / Top-P sampling
  - Sample from all tokens in the top P cumulative probability mass (i.e., where mass is concentrated)
  - Varies K depending on the uniformity of P_t

Holtzman et al., ICLR 2020

Nucleus (Top-P) Sampling

- Solution: Top-P sampling
  - Sample from all tokens in the top P cumulative probability mass (i.e., where mass is concentrated)
  - Varies K depending on the uniformity of P_t

$P_t^1(y_t = w \mid \{y\}_{<t})$



$P_t^2(y_t = w \mid \{y\}_{<t})$

$P_t^3(y_t = w \mid \{y\}_{<t})$

Holtzman et al., ICLR 2020

Temperature Scaling

$$P(y_t = w) = \frac{\exp(S_w)}{\sum_{v \in V} \exp(S_v)}$$

- Recall: On timestep t, the model computes a prob distribution P_t by applying the softmax function to a vector of scores $s \in \mathbb{R}^{|V|}$

- We can apply a temperature hyperparameter τ to the softmax to rebalance P_t

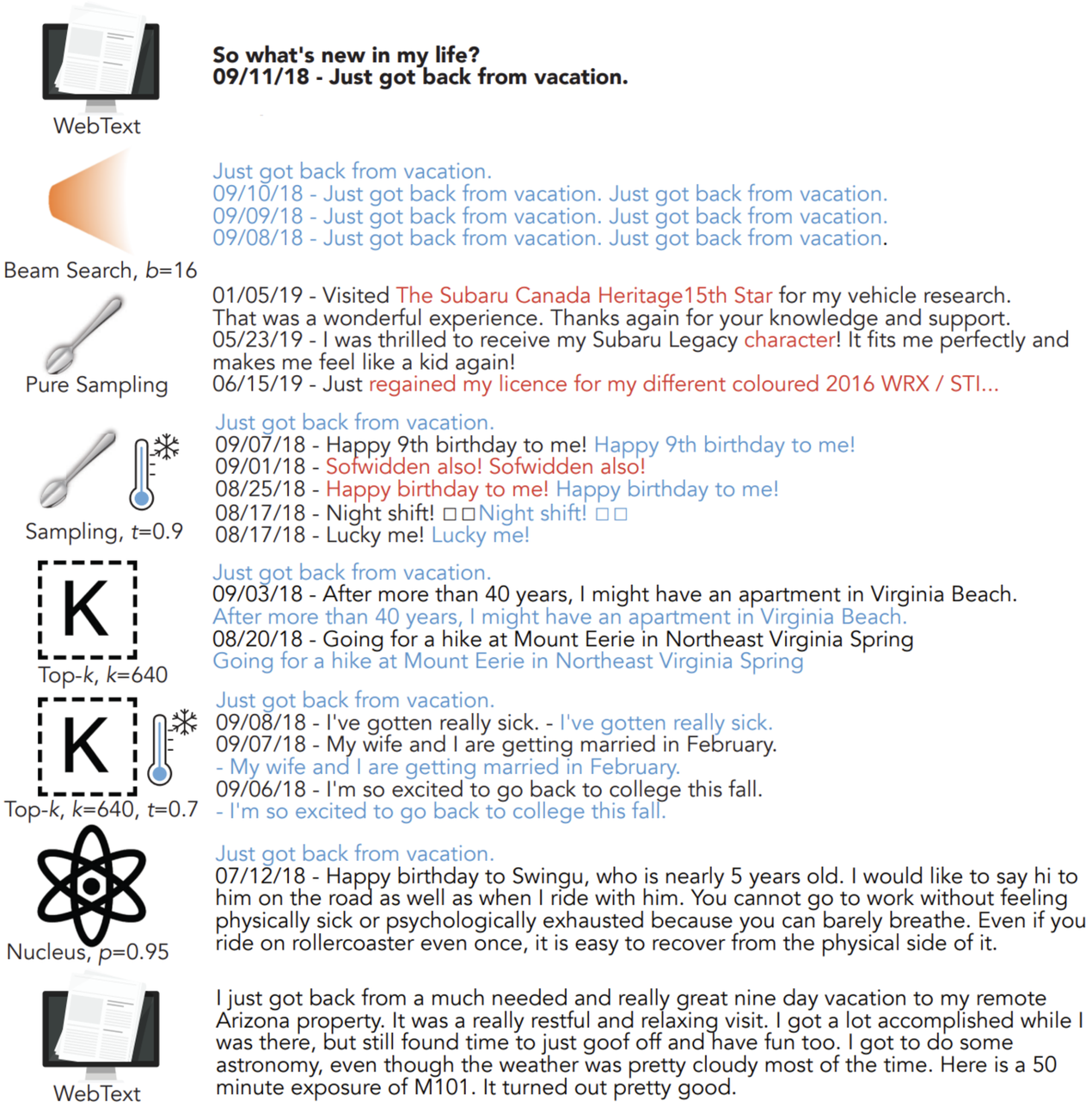
$$P(y_t = w) = \frac{\exp(S_w/\tau)}{\sum_{v \in V} \exp(S_v/\tau)}$$

- Raise the temperature $\tau > 1$: P_t becomes more uniform
  - More diverse output (probability is spread around vocab)
- Lower the temperature $\tau < 1$: P_t becomes more spiky
  - Less diverse output (probability is concentrated on top words)

Temperature is a hyperparameter for decoding: It can be tuned for both beam search and sampling.

Comparing different decoding algorithms

- Generate text to continue a given context
  - Open-ended generation
- Same decoding algorithms are also useful for close-ended generation tasks



Holtzman et al., ICLR 2020

Modern Decoding: Takeaways

- Natural language distributions are very peaky but the softmax function assigns probabilities to all tokens in the vocabulary
- Hence we need approaches to truncate / modify the softmax distribution
  - Pure, Top-k, Top-p (Nucleus), Temperature

Post-training of LLM

Three Stages of Training LLMs

Stage 1

- Pre-training on large corpus of text
- Produces a Base Language Model
- Continued Pre-training for domain adaptation (optional; sometimes called mid-training, stage 1.5)

Stage 2

- Post-training for Task Adaptation
- Seq2Seq Instruction Tuning (Supervised Finetuning, different meaning than BERT-style classification tasks)

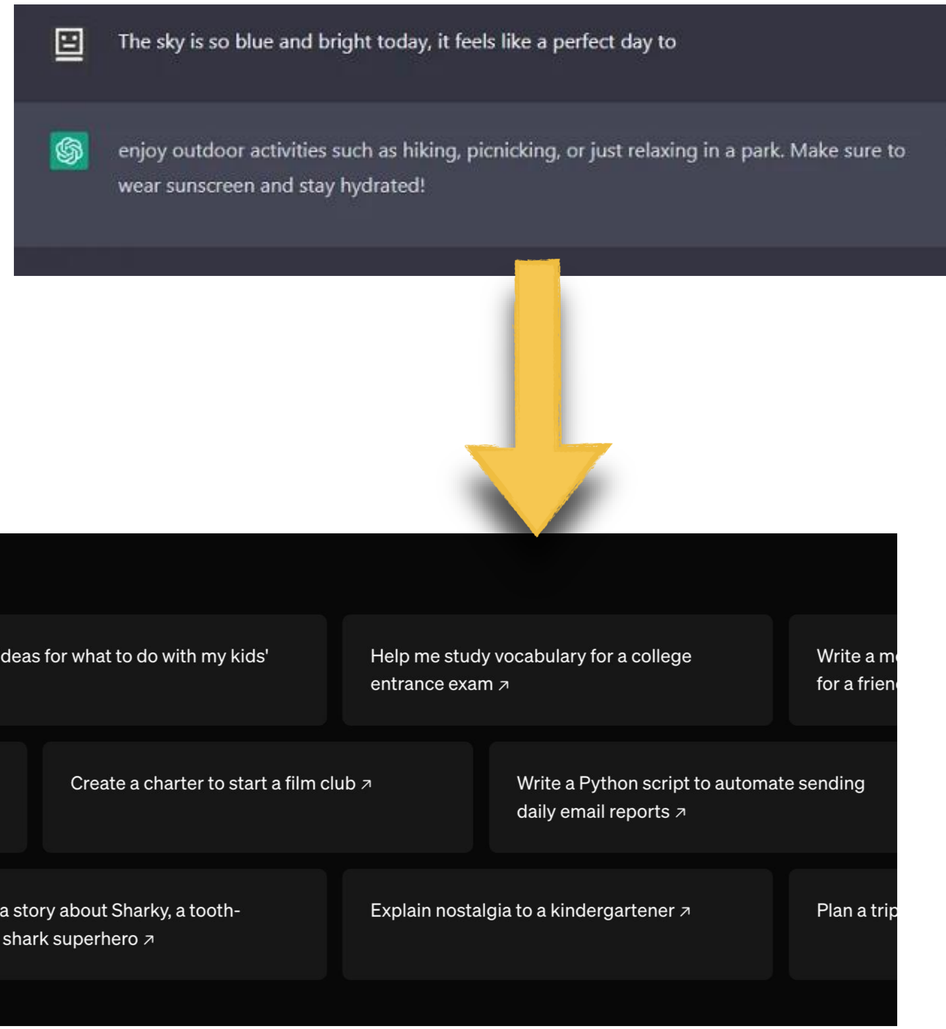
Stage 3

- Post-training for Preference Alignment
  - Either Reinforcement Learning with Human Feedback : Train a supervised classifier (reward model) on human demonstrations to provide feedback to LM. Reinforcement learning to maximize rewards given by reward model
  - Or, train with a preferred and a dispreferred generation (more popular now)

Inference Prompting with Instructions and Demonstrations (also called examples, shots)

Pre-training and Post-training

- The term “training” is no longer specific enough :)
- Pre-training: Self-supervised, standard next token prediction
- Post-training: Supervision (sequence to sequence)
  - Instruction-Tuning: Supervision is not necessarily via labels, but sequence pairs. Labels in standard NLP benchmarks can be converted into sequence pairs
  - Preference-Tuning: Collects human judgments / preferences as rewards, or a pair of preferred / dispreferred generations

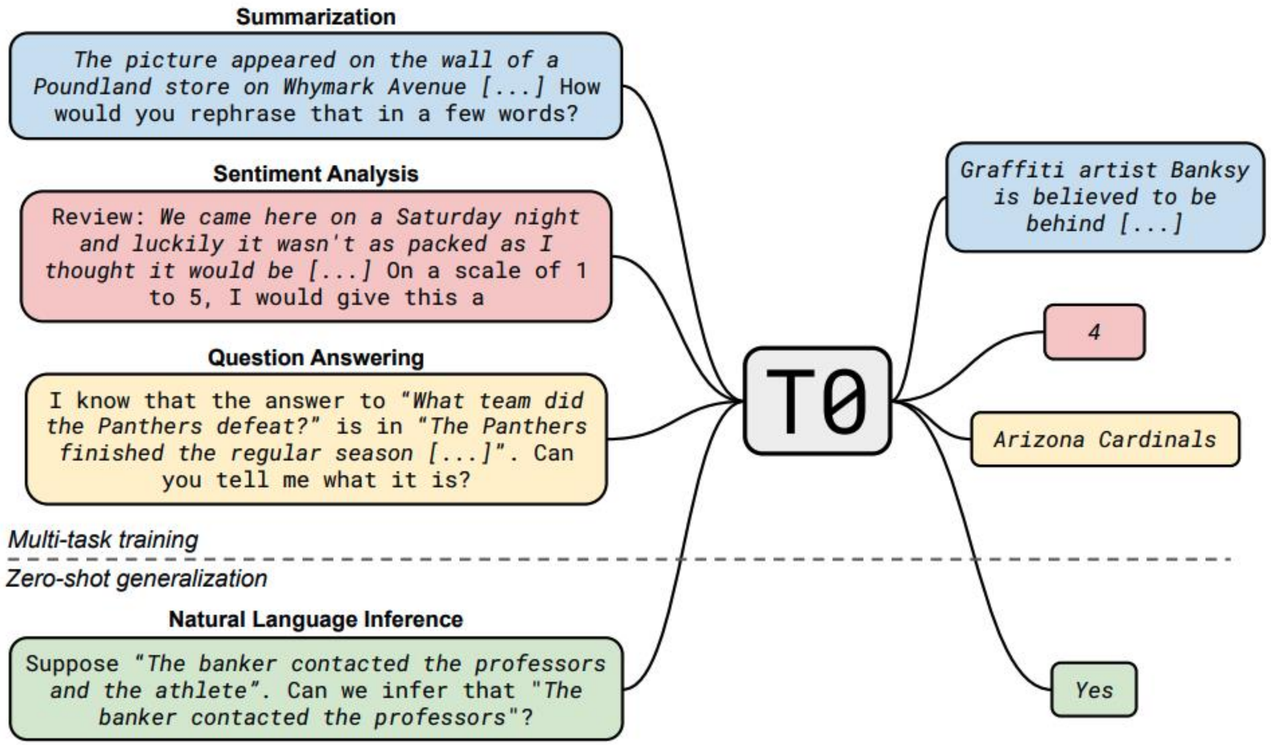


Post-training converts next-word completion models into **world models (agents, assistants)** with many capabilities!

Instruction-Tuning LLMs for Task Adaptation

Instruction Tuning

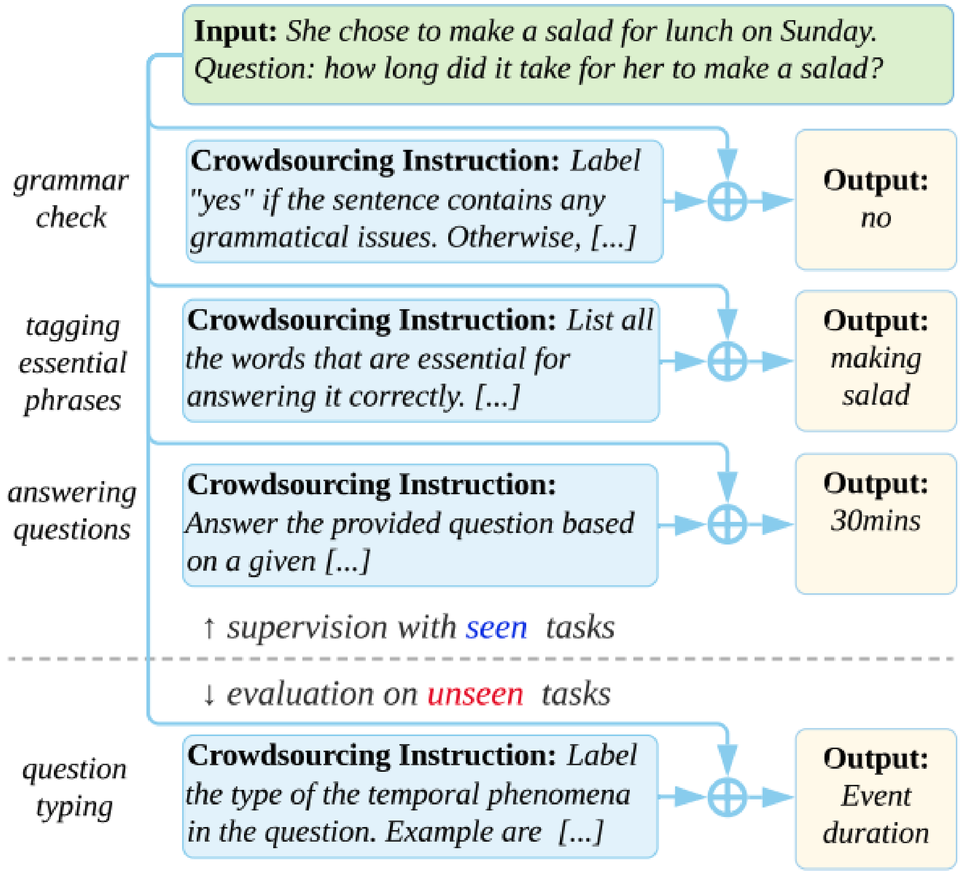
- Pretraining:
  - Train a model to continue a given context
- Instruction Tuning:
  - Train a model to follow varied instructions
  - Needed because the vast majority of pretraining is done on data which are not in the form of instructions
  - Fine-tuned (using the next-token-prediction objective) on a dataset of instructions together with correct responses



"Multitask Prompted Training Enables Zero-Shot Task Generalization" (Sahn et al., 2022)

Instruction Tuning and Task Generalization

- During instruction tuning, the model learns to follow instructions of given tasks
- At test time, it generalizes to follow instructions on unseen tasks!



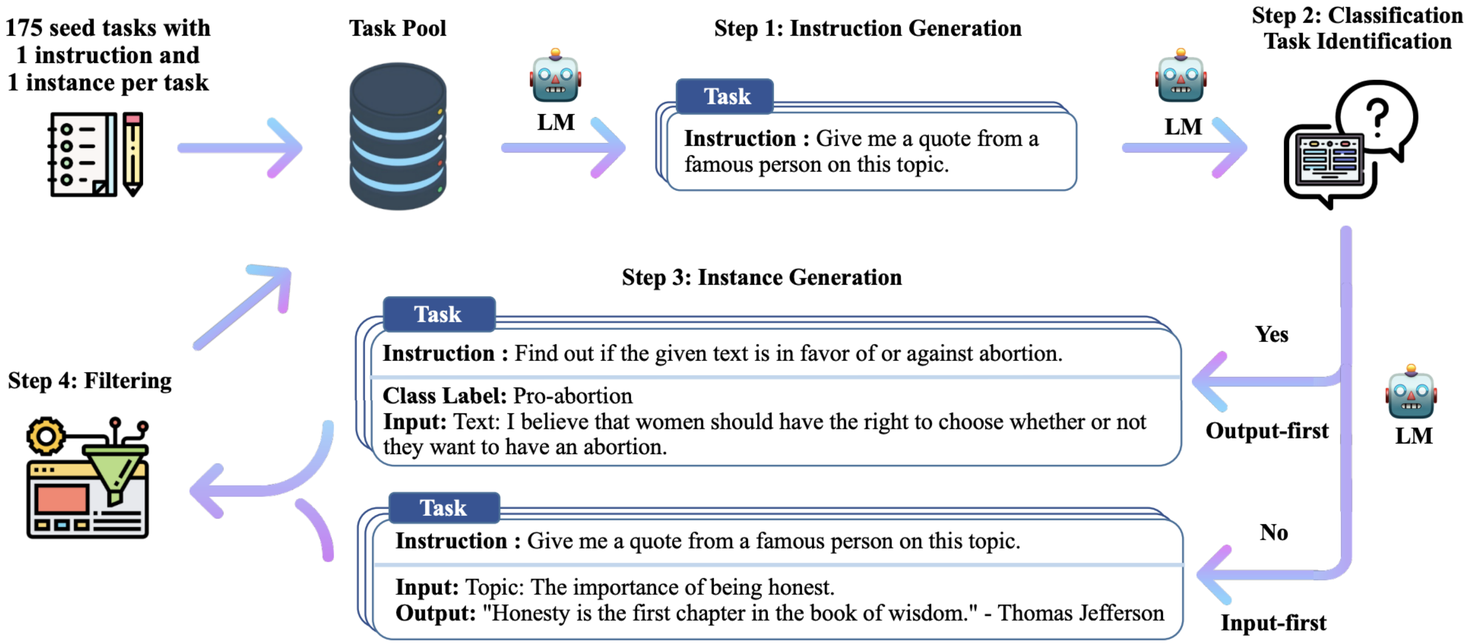
"Cross-Task Generalization via Natural Language Crowdsourcing Instructions" (Mishra et al., 2022)

Instruction Tuning Data

More data (instructions) → better model

Resource →	SUP-NATINST (this work)	NATINST (Mishra et al., 2022b)	CROSSFIT (Ye et al., 2021)	PROMPTSOURCE (Bach et al., 2022)	FLAN (Wei et al., 2022)	INSTRUCTGPT (Ouyang et al., 2022)
Has task instructions?	✓	✓	✗	✓	✓	✓
Has negative examples?	✓	✓	✗	✗	✗	✗
Has non-English tasks?	✓	✗	✗	✗	✓	✓
Is public?	✓	✓	✓	✓	✓	✗
Number of tasks	1616	61	269	176	62	–
Number of instructions	1616	61	–	2052	620	14378
Number of annotated tasks types	76	6	13	13*	12	10
Avg. task definition length (words)	56.6	134.4	–	24.8	8.2	–

Super-NaturalInstructions. (Wang et al., 2022)



Diverse data (instructions) → better model

"Self-Instruct: Aligning Language Models with Self-Generated Instructions" (Wang et al., 2023)

Instruction Tuning Data

- Instruction tuning datasets are often created by repurposing standard NLP datasets for tasks like question answering or machine translation
- Often synthesized!
  - Prompting existing LLMs
- More variety in the instruction templates → better models!

		Model Details				Data Collection & Training Details			
Release	Collection	Model	Base	Size	Public?	Prompt Types	Tasks in Flan	# Exs	Methods
2020 05	UnifiedQA	UnifiedQA	RoBerta	110-340M	P	ZS	46 / 46	750k	
2021 04	CrossFit	BART-CrossFit	BART	140M	NP	FS	115 / 159	71.M	
2021 04	Natural Inst v1.0	Gen. BART	BART	140M	NP	ZS / FS	61 / 61	620k	+ Detailed k-shot Prompts
2021 09	Flan 2021	Flan-LaMDA	LaMDA	137B	NP	ZS / FS	62 / 62	4.4M	+ Template Variety
2021 10	P3	T0, T0+, T0++	T5-LM	3-11B	P	ZS	62 / 62	12M	+ Template Variety + Input Inversion
2021 10	MetaICL	MetaICL	GPT-2	770M	P	FS	100 / 142	3.5M	+ Input Inversion + Noisy Channel Opt
2021 11	ExMix	ExT5	T5	220M-11B	NP	ZS	72 / 107	500k	+ With Pretraining
2022 04	Super-Natural Inst.	*Tk*-Instruct	T5-LM, mT5	11-13B	P	ZS / FS	1556 / 1613	5M	+ Detailed k-shot Prompts + Multilingual
2022 10	GLM	GLM-130B	GLM	130B	P	FS	65 / 77	12M	+ With Pretraining + Bilingual (en, zh-cn)
2022 11	xP3	BLOOMz, mT0	BLOOM, mT5	13-176B	P	ZS	53 / 71	81M	+ Massively Multilingual
2022 12	Unnatural Inst.†	T5-LM-Unnat. Inst.	T5-LM	11B	NP	ZS	~20 / 117	64k	+ Synthetic Data
2022 12	Self-Instruct†	GPT-3 Self Inst.	GPT-3	175B	NP	ZS	Unknown	82k	+ Synthetic Data + Knowledge Distillation
2022 12	OPT-IML Bench†	OPT-IML	OPT	30-175B	P	ZS + FS, CoT	~2067 / 2207	18M	+ Template Variety + Input Inversion + Multilingual
2022 10	Flan 2022 (ours)	Flan-T5, Flan-PaLM	T5-LM, PaLM	10M-540B	P NP	ZS + FS, CoT	1836	15M	+ Template Variety + Input Inversion + Multilingual

"The Flan Collection: Designing Data and Methods for Effective Instruction Tuning" (Longpre et al., 2023)

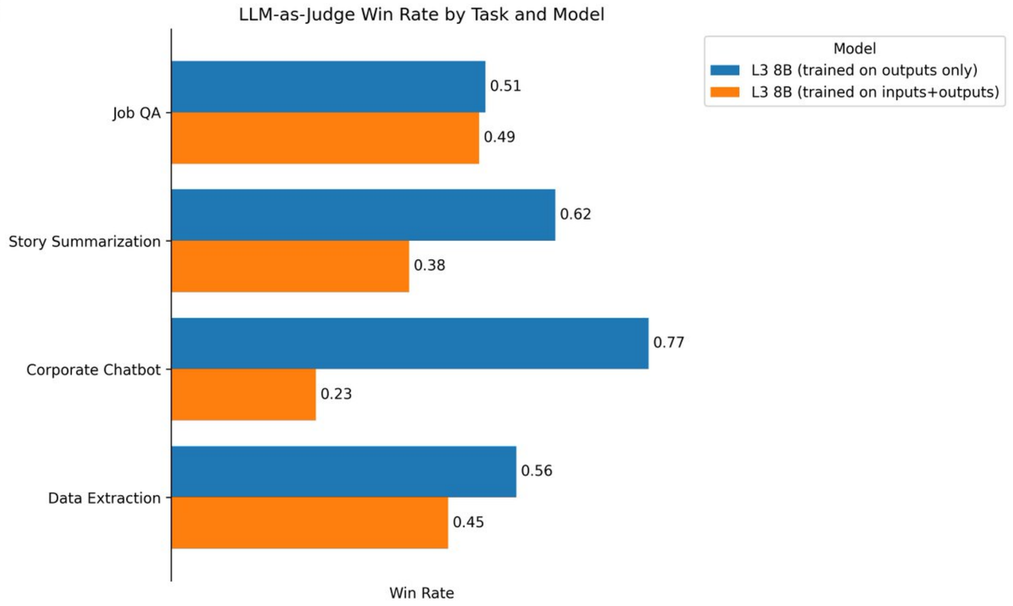
Instruction Tuning: Masking Instructions

- We're still using decoder-only models (the same as we used in pre-training)
- How to update these models for an encoder-decoder like behavior?
- The instruction itself is masked, so the model does not generate instructions

In the loss function, mask out the tokens corresponding to prompt

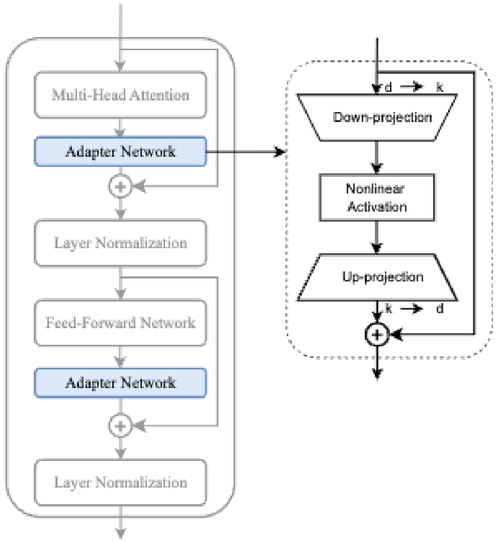
```
Below is an instruction that describes a task. Write a
response that appropriately completes the request.

### Instruction:
Rewrite the following sentence using passive voice.

### Input:
The team achieved great results.

### Response:
Great results were achieved by the team.
```

Calculate the loss only over the tokens corresponding to the response text

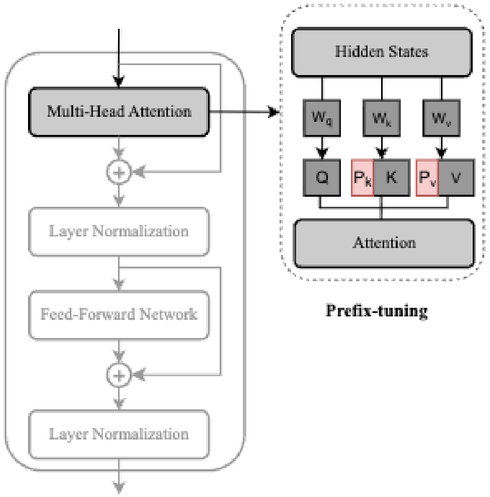


An illustration of input masking where the highlighted text is still fed to the LLM, but it is not used when calculating the loss during training.

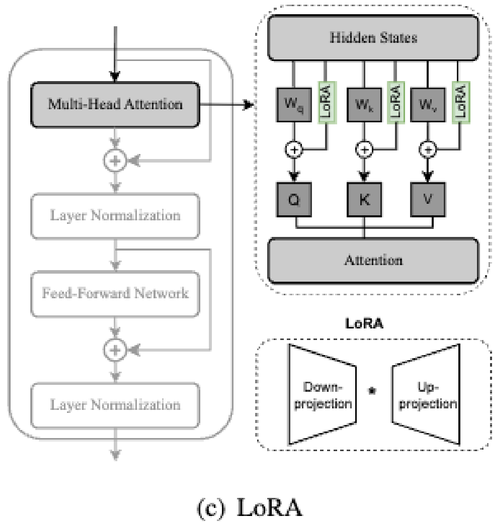
Parameter-Efficient Fine-tuning



(a) Sequential Adapter



(b) Prefix-tuning

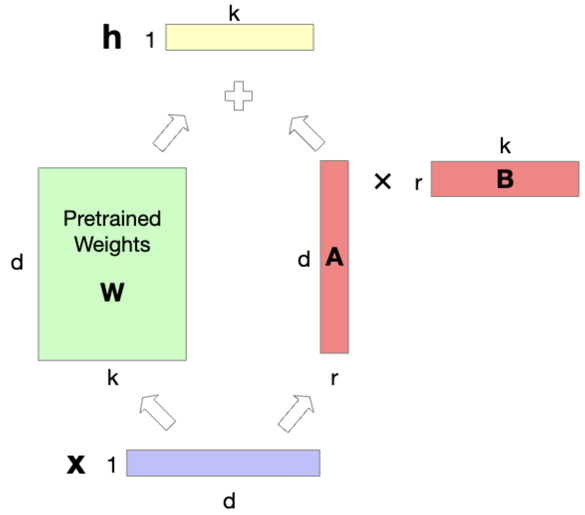


(c) LoRA

- Fine-tuning can be very difficult and expensive with LLMs
  - enormous numbers of parameters to train; think 70B parameters, and their gradients!!
  - each pass of batch gradient descent has to backpropagate through many many huge layers.
- Alternative: allow a model to be finetuned without changing all the parameters.
  - parameter-efficient fine tuning or PEFT,
  - efficiently select a subset of parameters to update when finetuning and keep the rest frozen
- Examples: Adapters, Prefix-Tuning, LoRA or Low Rank Adaptation

LoRA: Low-Rank Adaptation

- Instead of updating weight matrices in attention layers during finetuning, Low-Rank Adaptation eases this by updating a low-rank approximation of the matrix : matrices which are far smaller
- LoRA freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture
- Gradient updates are reparametrized as $\mathbf{W}_0 + \Delta\mathbf{W} = \mathbf{W}_0 + \mathbf{B} \cdot \mathbf{A}$
  - Where $\mathbf{B}$ and $\mathbf{A}$ are low-rank matrices, the only matrices to be updated
- LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times for GPT-3 175B
- Usually comes at a (small) cost to performance



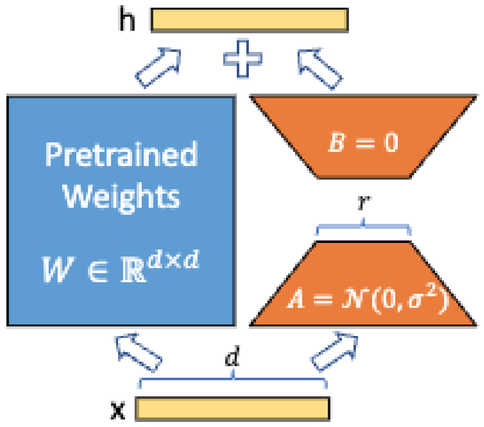


Figure 1: Our reparametrization. We only train A and B.

LoRA. Hu et al., 2021

Interacting with LLMs: Prompting

Interfacing with Large Language Models

- Once trained, language models can be very powerful
  - The power only increases with scale
  - Most tasks in NLP can be formatted as sequence completion tasks
  - How to interface with a language model to extract relevant information?
- Prompting (or In-Context / Few-Shot Learning): the ability to do many tasks with no gradient updates and no / a few examples, by simply:
  - Specifying the right sequence prediction problem
  - You can get interesting zero-shot behavior if you're creative enough with how you specify your task!

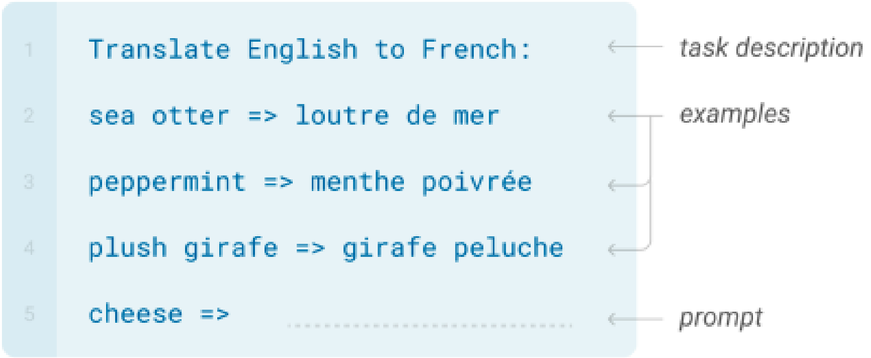
Basic Prompt Templates

Summarization	{input} ; `tldr;`
Translation	{input} ; `translate to French:`
Sentiment	{input}; `Overall, it was`
Fine-Grained-Sentiment	{input}; `What aspects were important in this review?`

Prompting

- Interface to a language model: prompts in natural language
- Very large language models seem to perform some kind of "learning" without gradient updates!!! "Learn"simply from examples you provide within their contexts
  - Sometimes called in-context learning
  - Misnomer: no learning (parameter update) actually happens during prompting
- Can be zero shot (without examples) or few-shot (with a few examples)
  - Typically <10
  - With powerful LLMs, 0-shot approaches are all you need!

Zero-Shot



Few-Shot

"Language Models are Few-Shot Learners" (Brown et al., 2020)

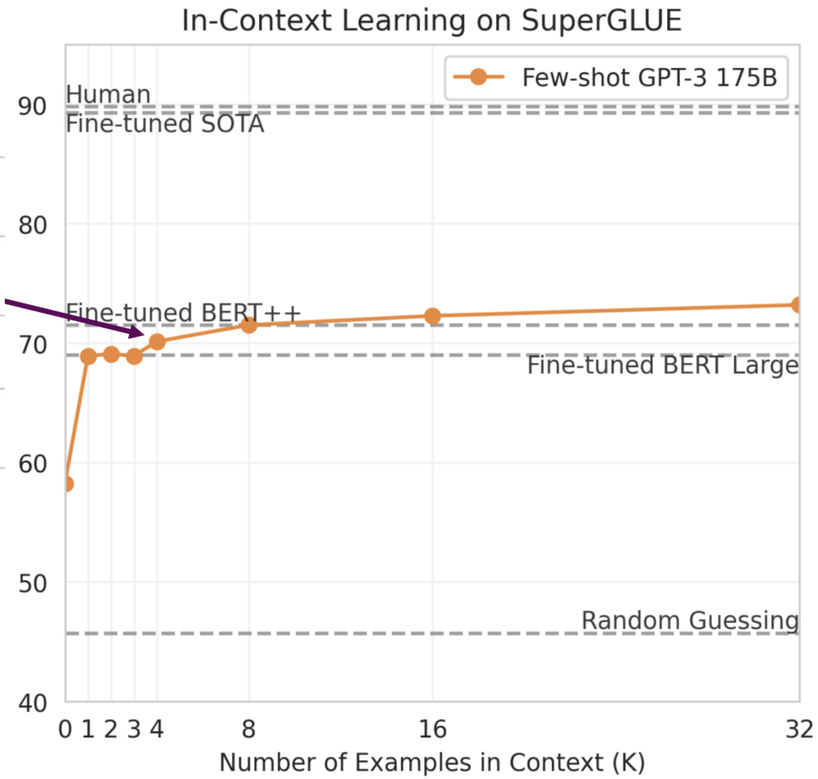
Prompting: Success

- Much more flexible than older formulation of pre-training encoder-only models and fine-tuning to specific classification tasks (the BERT paradigm)
- Now, pre-train and instruction tune one large model and prompt it to do a variety of tasks!
- Much much more generalizability!

Emergent few-shot learning

Few-shot

```
1  Translate English to French:
2  sea otter => loutre de mer
3  peppermint => menthe poivrée
4  plush girafe => girafe peluche
5  cheese =>
```



[Brown et al., 2020]

Why does prompting work so well?

- Induction heads
- Discovered by looking at mini language models with only 1-2 attention heads
- If the model sees the pattern AB … A in an input sequence, it predicts that B will follow, instantiating the pattern completion rule AB… A→ B
- Perhaps a generalized fuzzy version of this pattern completion rule, implementing a rule like A*B* … A→ B*, where A* ≈ A and B* ≈ B (by ≈ we mean some form of semantically similarity), might be responsible for in-context learning

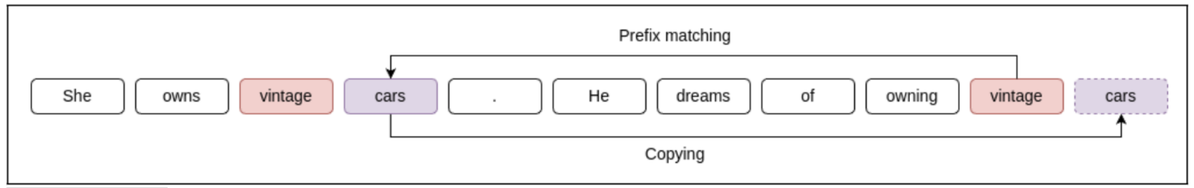
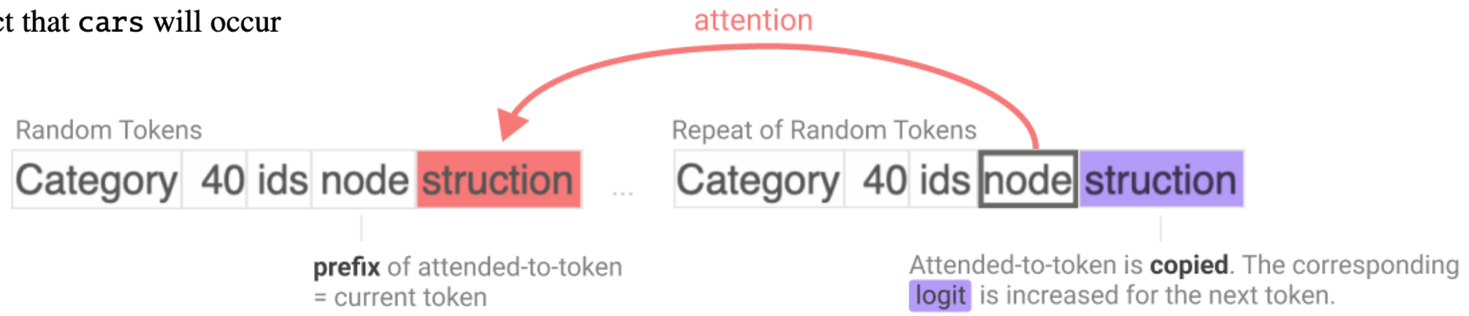


Figure 12.3 An induction head looking at `vintage` uses the *prefix matching* mechanism to find a prior instance of `vintage`, and the *copying* mechanism to predict that `cars` will occur again. Figure from Crosbie and Shutova (2022).



"In-context Learning and Induction Heads" (Olsson et al., 2022)