LLM Inference

Zdeněk Kasner

21 March 2024

Charles University
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Lesson plan

1. 💬 **Discussion** (10 min)
2. 🦙 **(L)LM zoo** (5 min)
3. 🔍 **Transformer inference** visualization (20 min)
4. 💻 **Decoding** algorithms (30 min)
5. 🤗 **Text generation** hands-on (20 min)

Discussion

Discussion time!

1. How many **trained language models** do you estimate to be publicly available?

2. What is the difference between **inference**, **generation**, and **decoding**?

LLM Zoo

Warm-up: identify what is **not** a language model

BART

BERT

BigBird

ERNIE

HOMER

Optimus

MARGE

Megatron

Warm-up: identify what is **not** a language model

BART | BERT | BigBird | ERNIE

not yet

HOMER | Optimus | MARGE | Megatron

images with public license

(Although...)

source: https://arxiv.org/abs/2402.11746

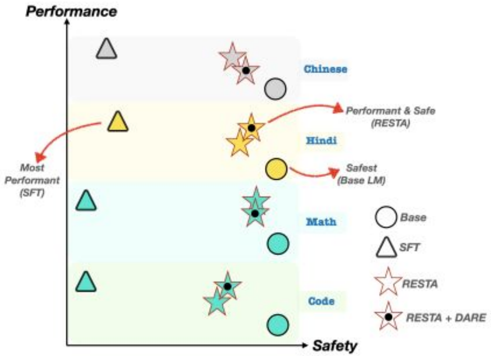
Language Models are Homer Simpson! Safety Re-Alignment of Fine-tuned Language Models through Task Arithmetic

Rishabh Bhardwaj[1], **Do Duc Anh**[2], **Soujanya Poria**[1]

[1] Singapore University of Technology and Design, [2] Nanyang Technological University

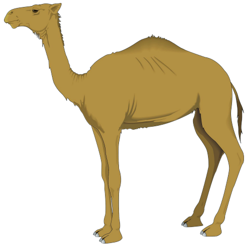
Abstract

Aligned language models face a significant limitation as their fine-tuning often results in compromised safety. To tackle this, we propose a simple method RESTA that performs LLM safety realignment. RESTA stands for **RE**storing **S**afety through **T**ask **A**rithmetic. At its core, it involves a simple arithmetic addition of a safety vector to the weights of the compromised model. We demonstrate the effectiveness of RESTA in both parameter-efficient and full fine-tuning, covering a wide range of downstream tasks, in-



Warm-up: identify what is **not** a language model

Alpaca

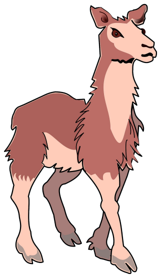


Camel

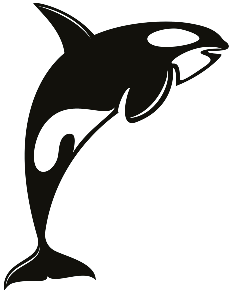
Falcon

Flamingo

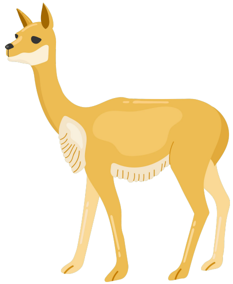
Koala



Llama



Orca

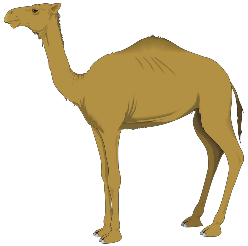


Vicuna

images from https://creazilla.com/

Warm-up: identify what is **not** a language model

Alpaca

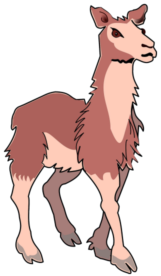


Camel

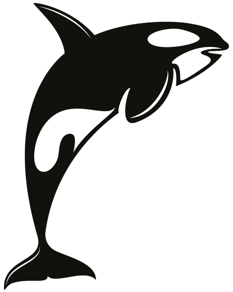
Falcon

Flamingo

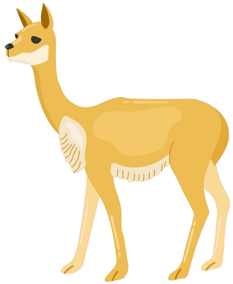
Koala



Llama



Orca



Vicuna

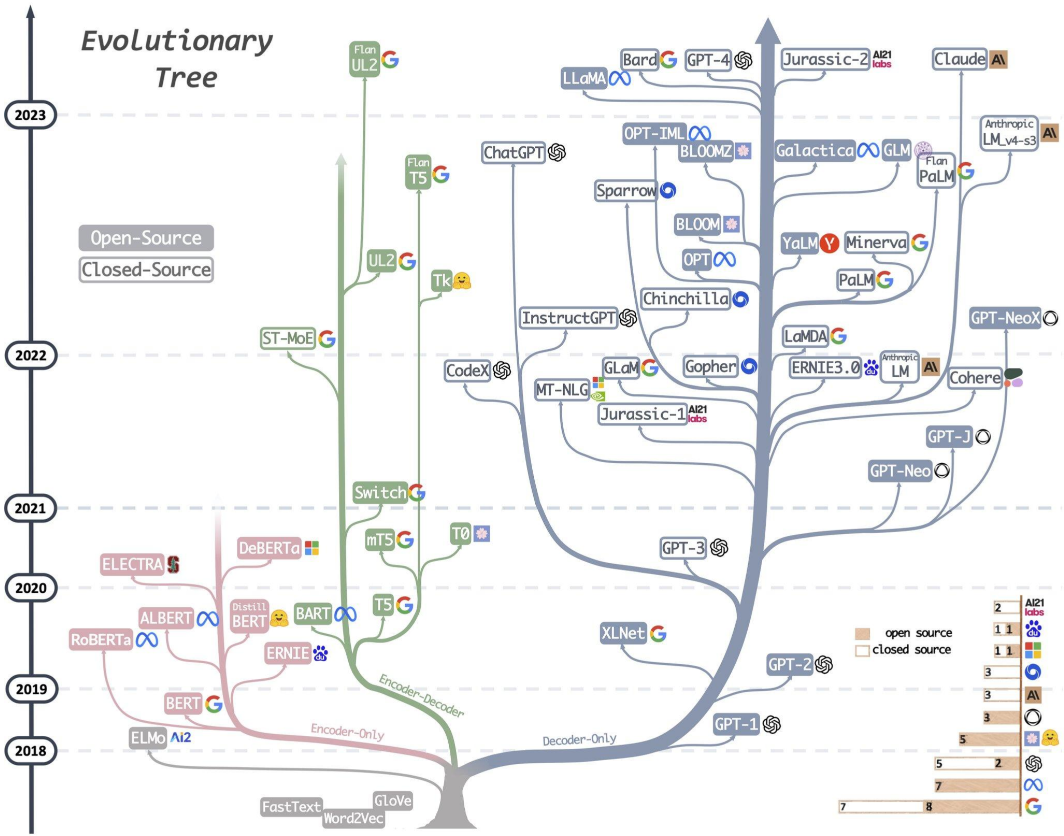
images from https://creazilla.com/

Some numbers

- **555,743 models** uploaded on **HuggingFace** (2024/03/18)
  - includes finetuned / scaled variants of the same base model
- **123 models** on **AlpacaEval Leaderboard**
  - mostly instruction-tuned models
- **73 models** in the **LMSYS Chatbot Arena**
  - mostly models finetuned for chat
- **1 ~~model~~ platform** ~~to rule them all~~ everyone knows (ChatGPT)
- See also https://github.com/Hannibal046/Awesome-LLM.

LLM evolutionary tree

source: https://arxiv.org/abs/2304.13712



Transformer Inference

Inference vs. generation vs. decoding

Inference

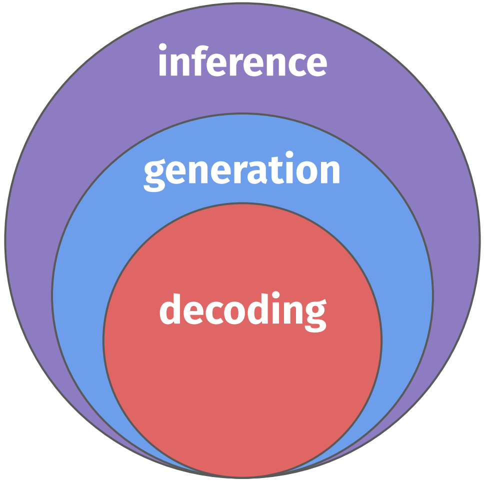
The concept of using a trained model for **making predictions** on new data (for classification, sequence tagging, text generation, ...).

Generation

The process of using a trained model for **producing a sequence of tokens**.

Decoding

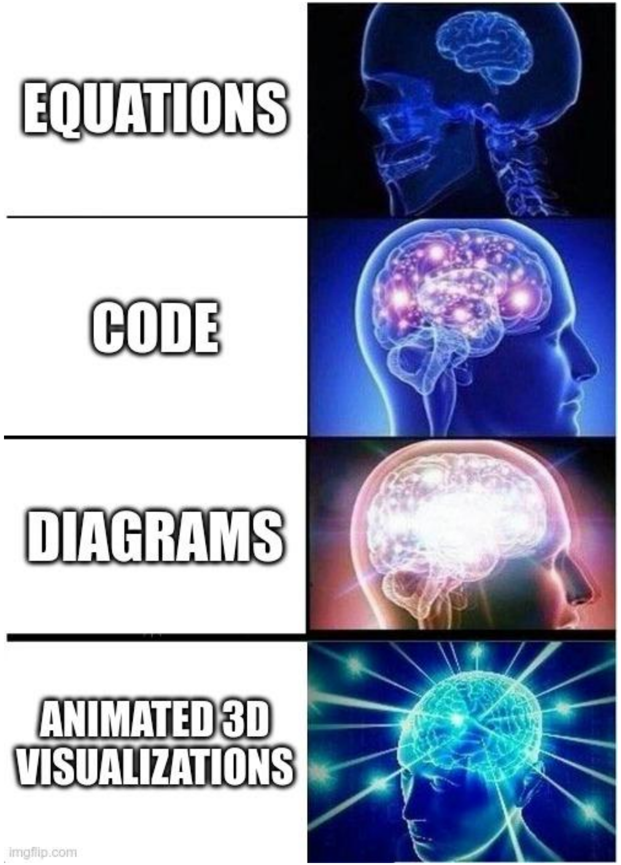
The algorithm of **turning the model's internal representation** into a **sequence of tokens**.



Transformer inference

https://bbycroft.net/llm

https://github.com/bbycroft/llm-viz



Text Generation

Autoregressive decoding

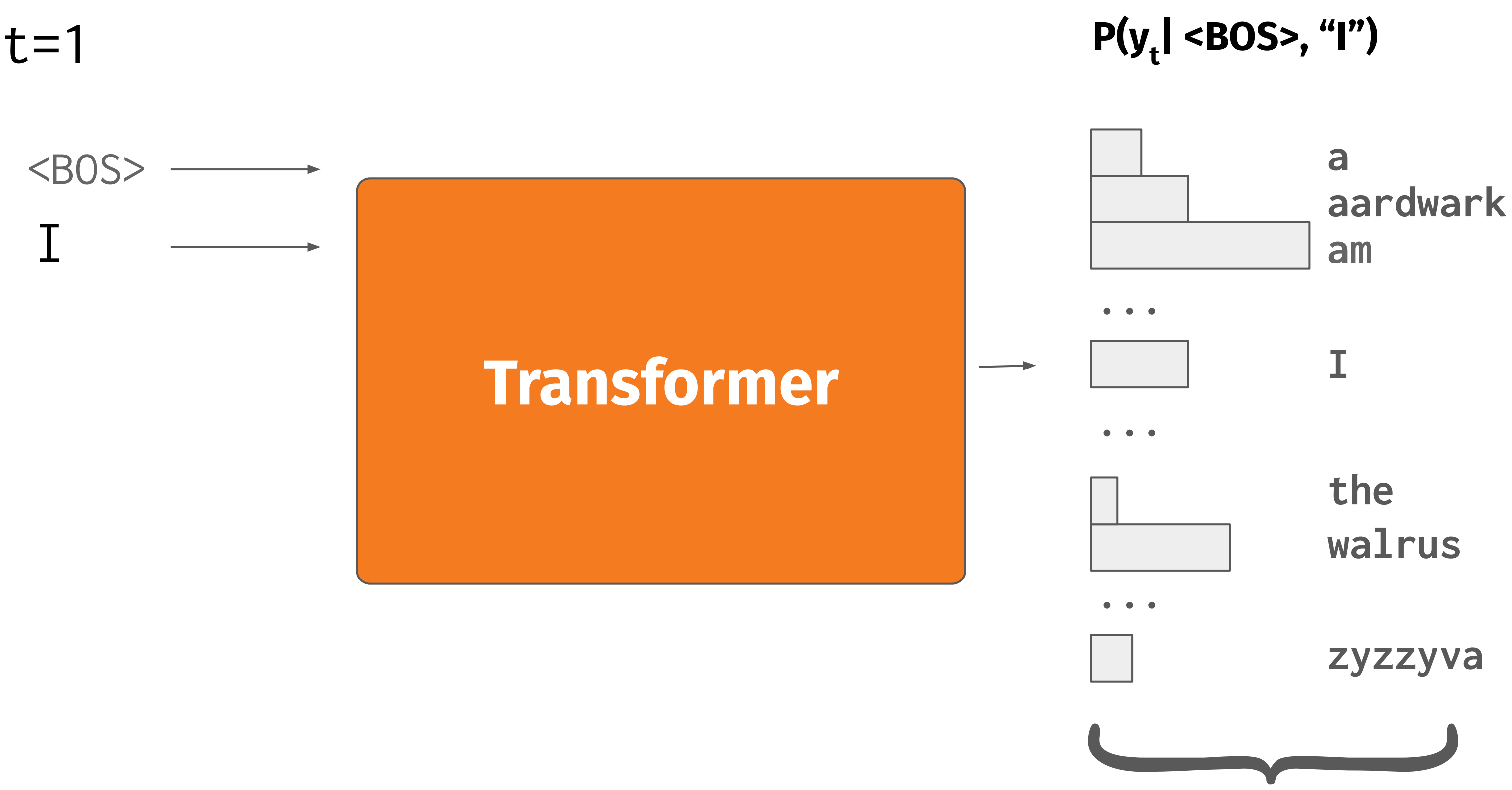
Where are we:

- **Task:** Generating a **sequence of tokens**.
- **Tool:** A language model (LM) giving us a **probability distribution** over the vocabulary for a given prefix.
- **Method:** Feed the sequence prefix in the LM → Select the next token → Append the token to the prefix → Repeat.

(how?)

= *Autoregressive decoding*

Autoregressive decoding

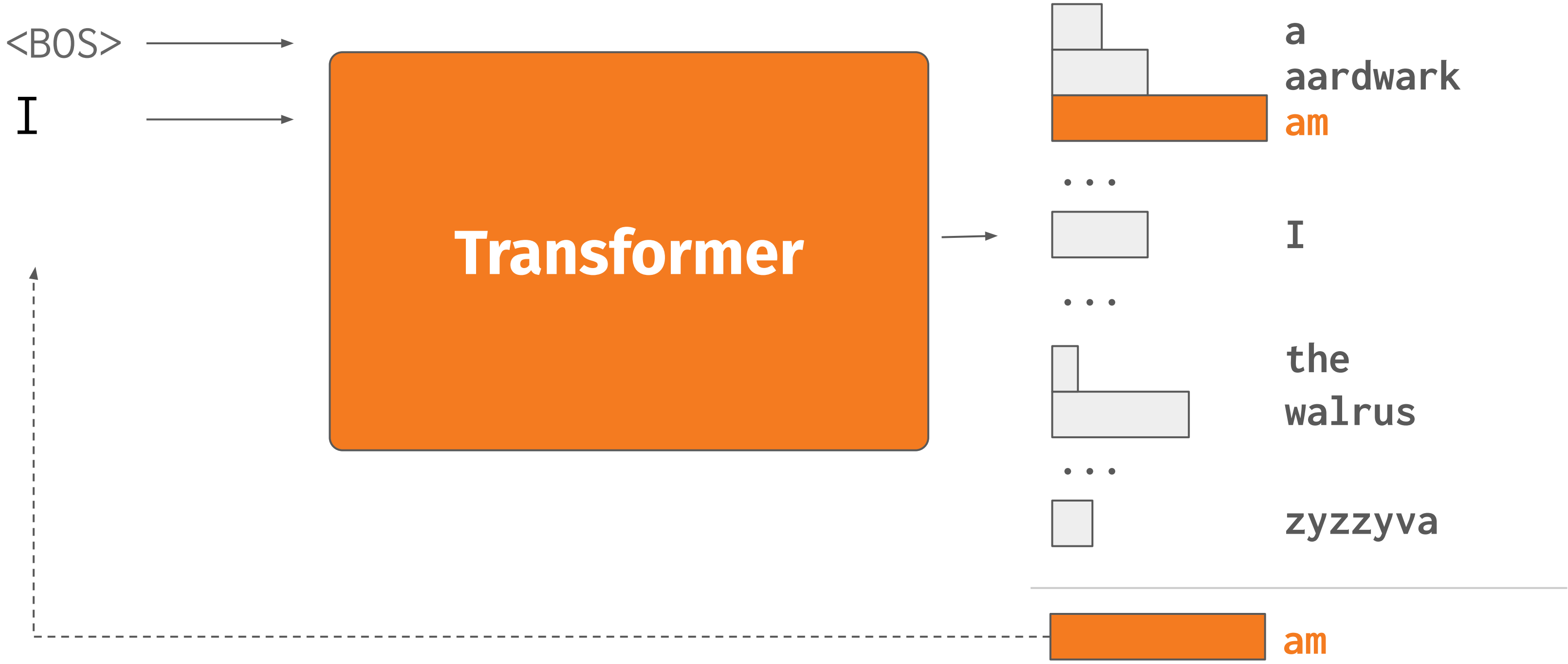


which token to select?

Autoregressive decoding

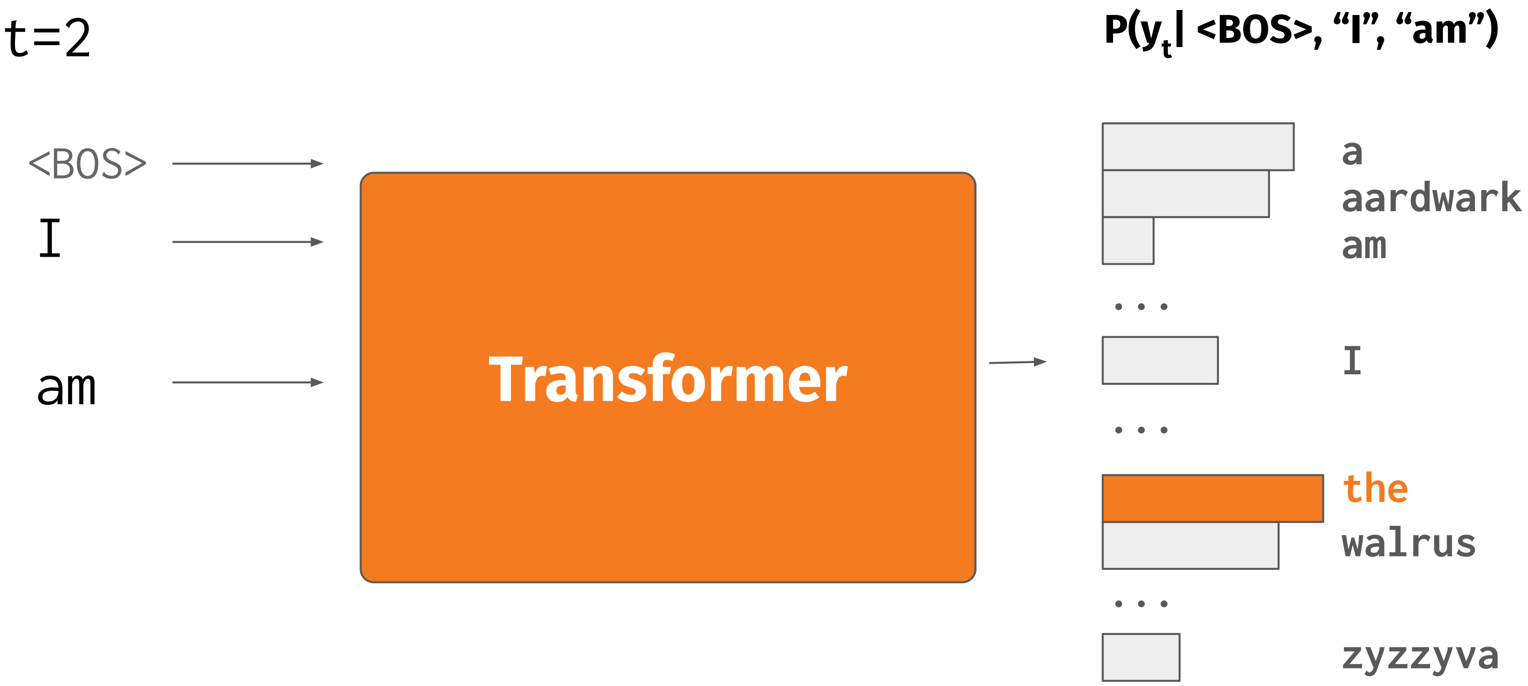
t=1

$P(y_t | \text{<BOS>}, \text{“I”})$

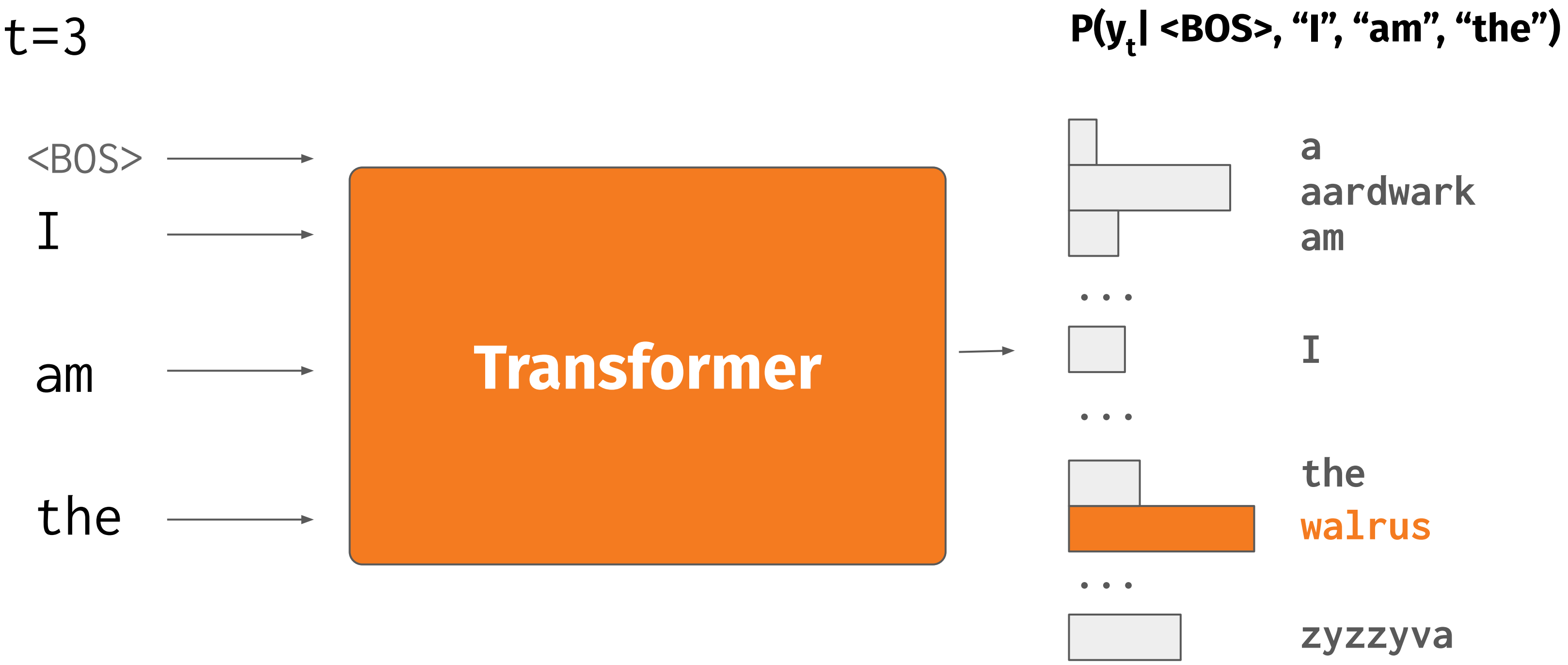


the most probable one?

Autoregressive decoding

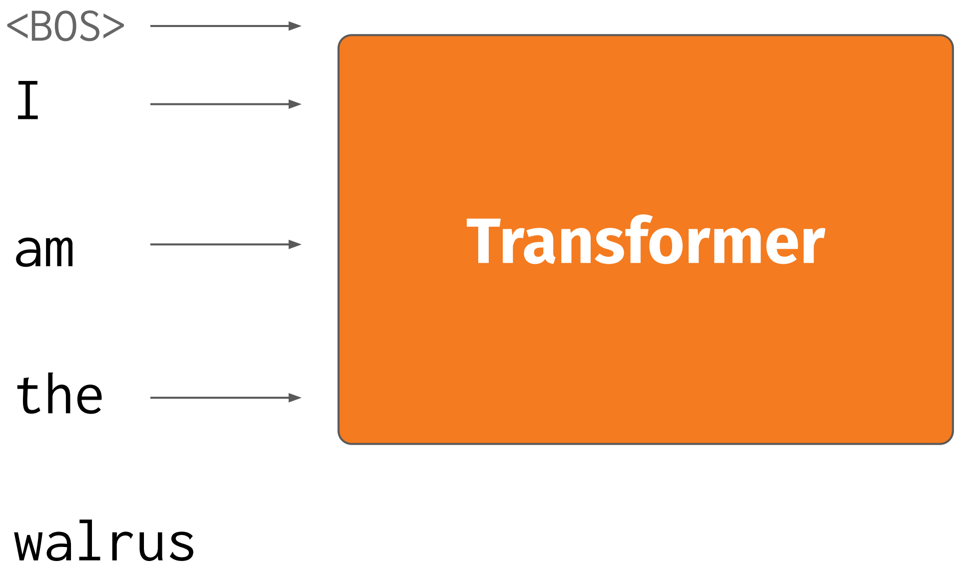


Autoregressive decoding



Autoregressive decoding

t=4



Autoregressive decoding

Have we generated **the most probable** sequence?

Do we **want** to generate the most probable sequence?

Decoding Algorithms

Decoding algorithms

finding the most probable sequence

- MAP decoding
- Greedy search
- Beam search

minimizing unwanted behavior

- MBR decoding

sampling a random sequence

- Top-k sampling
- Top-p (nucleus) sampling
- Mirostat
- Typical sampling

Exact Inference = Maximum a posteriori (MAP) decoding

- Finding **the most probable sequence** (=mode of the LM distribution) given the step-wise factorization of sequence probability:

$$y^* = \arg\max_{y \in \mathcal{Y}} P(y) = \arg\max_{y \in \mathcal{Y}} \prod_{i=1}^{t} P(y_i | y_1, \ldots, y_{t-1})$$

- Intractable (exponential search space) → approximation algorithms
- The mode may even not be a good solution! ([1], [2])
  - e.g. an empty sequence

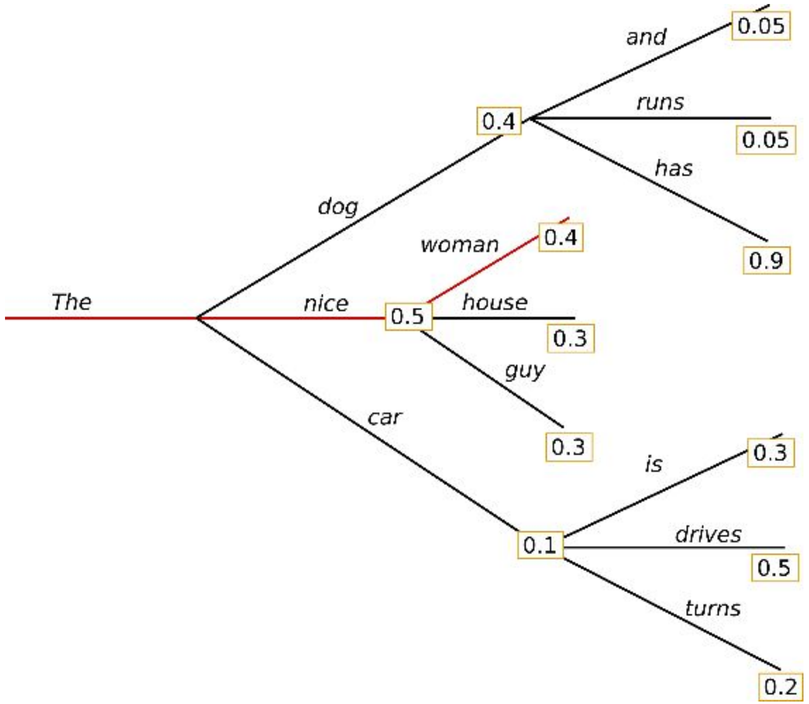
Greedy decoding

source: https://huggingface.co/blog/how-to-generate

- Selecting the **most probable token in each step** *t:*

$$y_t = \underset{y_t \in \mathcal{V}}{\arg\max}\, P(y_t | y_1, \ldots, y_{t-1})$$

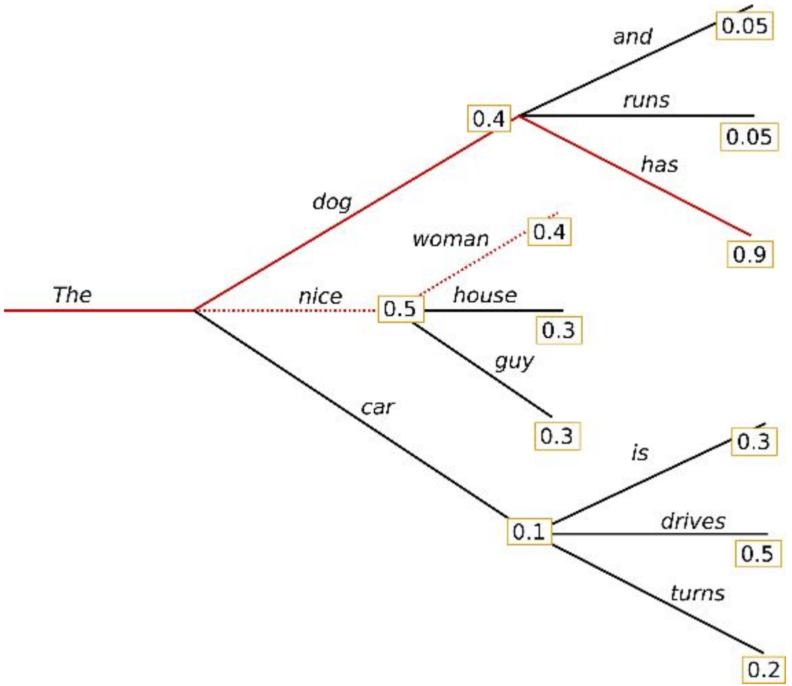
- Very fast, often works satisfactorily (especially with LLMs)
- Non-parameteric



Beam search

source: https://huggingface.co/blog/how-to-generate

- Parameter ***k*****: number of sequences**
- Each step *t*:
  - Extend the sequences from the step *t-1* with all possible tokens.
  - Select the *k* most probable sequences for the step *t+1*.
- Tuning k:
  - *k*=1 == greedy decoding
  - larger *k* → slower algorithm
  - k>1 allows re-ranking results



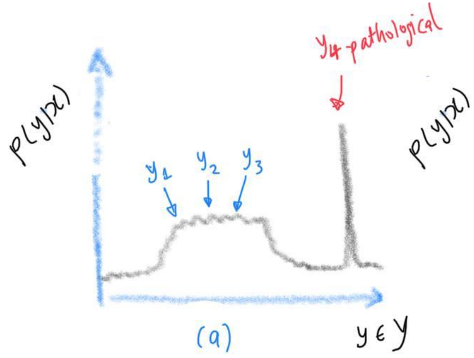
Minimum Bayes Risk (MBR) Decoding

source: [Minimum Bayes Risk Decoding]

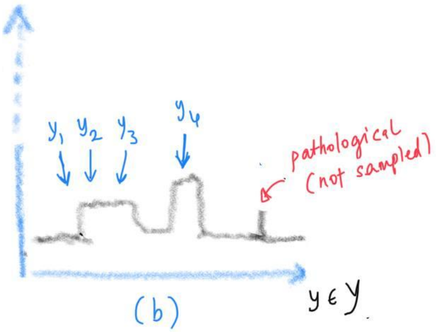
- Selecting the sequence most similar to other sequences = "**consensus decoding**"

$$y^* = \arg\max_{y_k \in \mathcal{Y}} \sum_{y_\ell \in \mathcal{Y} \setminus y_k} \operatorname{sim}(y_k, y_\ell)$$

- Useful for minimizing pathological behavior
- Intractable → we need a sampling algorithm
- Application in automatic speech recognition and machine translation



MAP 😕



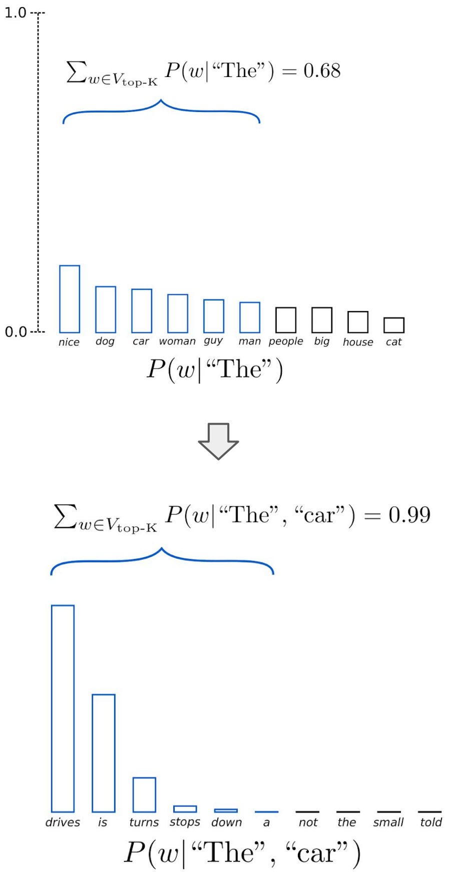
MBR 😊

Top-k sampling

source: https://huggingface.co/blog/how-to-generate

- Selecting the token in each step **randomly from *k*** ∈ {1, ..., |V|} **most probable tokens**
- The truncated distribution is re-weighed using softmax
- The shape of distribution can be adjusted using the **temperature *T***:

$$\mathrm{softmax}(y_i) = \frac{e^{y_i/T}}{\sum_{y_j \in \mathcal{V}_{\text{top-k}}} e^{y_j/T}}$$



t = 1

prefix = “The”
→ sampling from {nice, dog, car, woman, guy, man}

cum.prob. = 0.68

t = 2

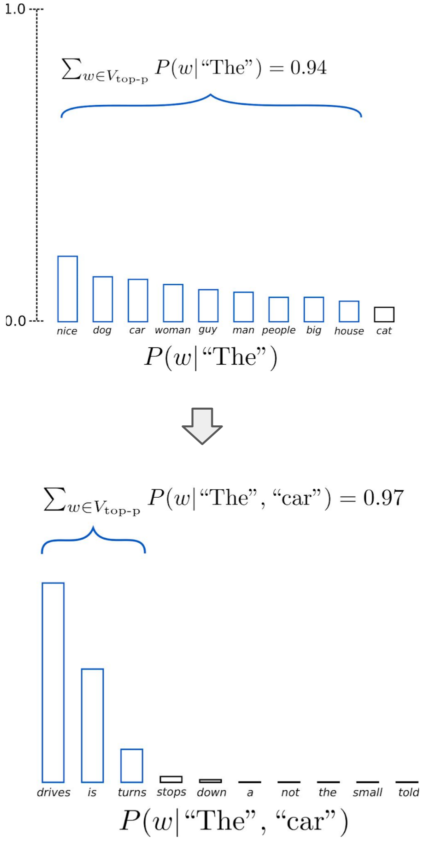
prefix = “The car”
→ sampling from {drives, is, turns, stops, down, a}

cum.prob. = 0.99

Top-p (nucleus) sampling

source: https://huggingface.co/blog/how-to-generate

- Similar to top-k sampling, but with a variable *k* in each step.
- Sampling from **“nucleus”:** set of the most probable tokens with combined probability summing to $p \in (0, 1]$
- The number of selected tokens is related to the “peakiness” of the distribution.



t = 1

prefix = “The”
→ sampling from {nice, dog, car, woman, guy, man, people, big, house}

cum.prob. = 0.94 (>0.9)

t = 2

prefix = “The car”
→ sampling from {drives, is, turns}

cum.prob. = 0.97 (>0.9)

Mirostat

reference: https://openreview.net/pdf?id=W1G1JZEIy5_

- Aims to eliminate repetition and incoherent text in stochastic algorithms
- Adapting the *k* parameter based on the **desired text perplexity** (“mirum” = surprise, “stat” = control)
- Parameters:
    - τ (tau) - the target perplexity
    - η (eta) - learning rate

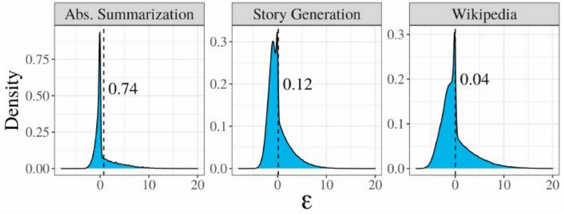
Algorithm 1: Adaptive top-k sampling for perplexity control

Target cross entropy τ, maximum cross entropy $\mu = 2 * \tau$, learning rate η
while *more words are to be generated* **do**
  Compute $\hat{s}$ from (40): $\frac{\sum_{i=1}^{N-1} t_i b_i}{\sum_{i=1}^{N-1} t_i^2}$
  Compute k from (41): $k = \left(\frac{\hat{\epsilon} 2^{\mu}}{1 - N^{-\hat{\epsilon}}}\right)^{\frac{1}{\hat{s}}}$
  Sample the next word X using top-k sampling
  Compute error: $e = \mathfrak{S}(X) - \tau$
  Update μ: $\mu = \mu - \eta * e$
end

(Locally) typical sampling

reference:
https://aclanthology.org/2023.tacl-1.7/

- Decodes text so that in each step, its perplexity is **close to the perplexity of the model**
  - Similar to Mirostat, but dynamic: the perplexity is not pre-specified
- Information theory: *typical* messages are the messages that we would expect from the process



p(H) = 0.75 H H H H → most probable sequence

p(T) = 0.25 H T H H → typical sequence

Other parameters

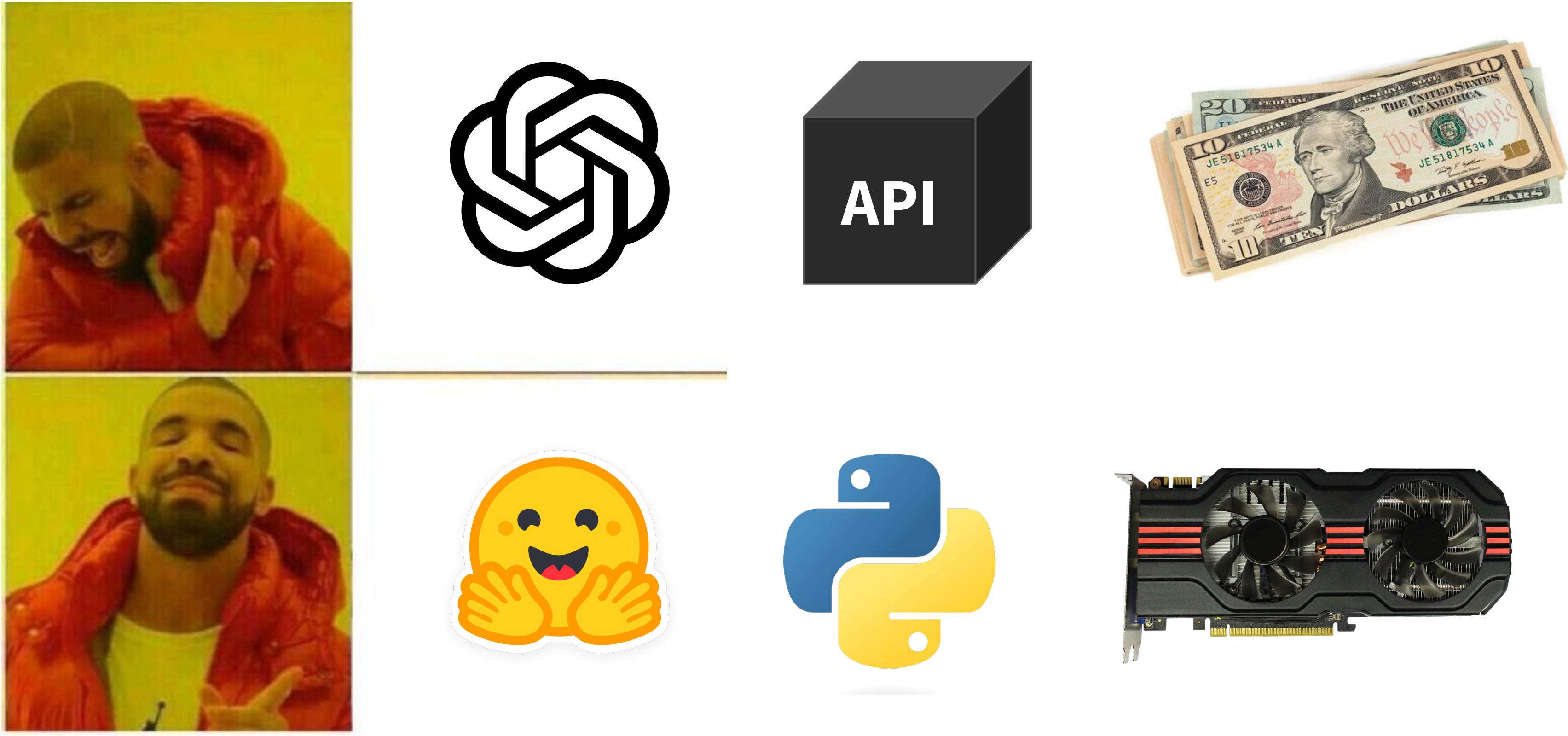
source:

- **repetition_penalty** - discounting the scores of previously generated tokens
- **length_penalty** - promoting shorter / longer sequences in beam search
- ...and many more, see HF docs

(we went through the most important ones, though)

Text Generation - hands-on

Text generation starter kit

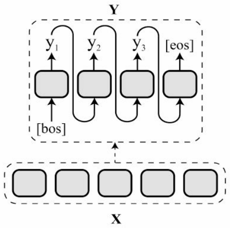


Demo time

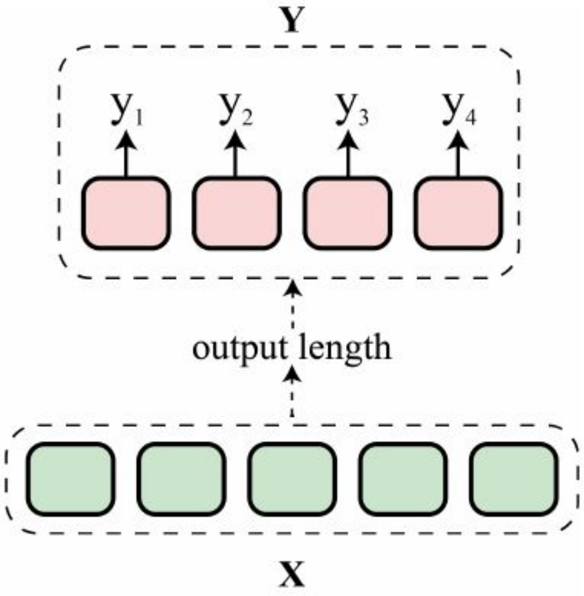
https://huggingface.co/docs/transformers/llm_tutorial
https://mlabonne.github.io/blog/posts/2023-06-07-Decoding_strategies.html

Bonus: Beyond Autoregressive Decoding

Bonus: Beyond Autoregressive Decoding



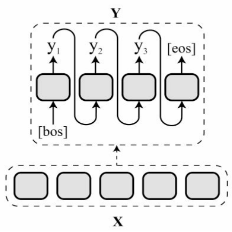
Autoregressive decoding



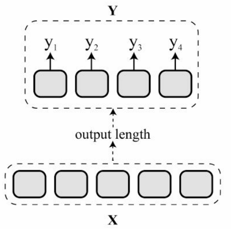
Non-autoregressive decoding [1]

- **can be parallelized**
- needs to predict output length
- weaker links between output tokens

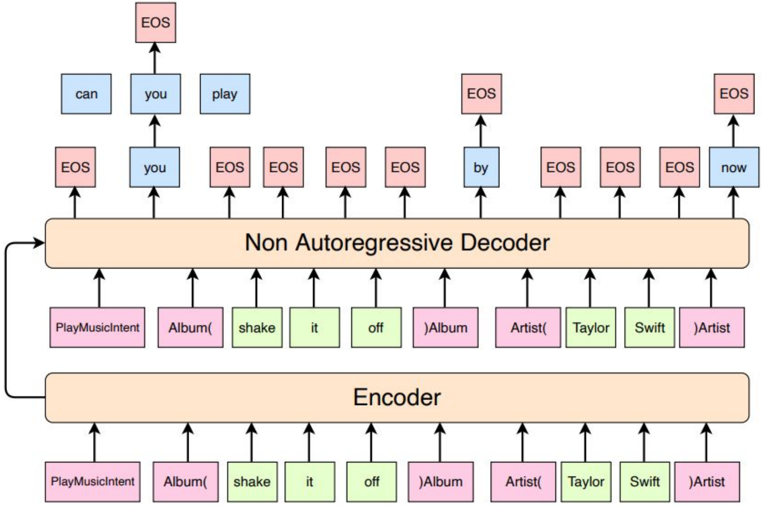
Bonus: Beyond Autoregressive Decoding



Autoregressive decoding



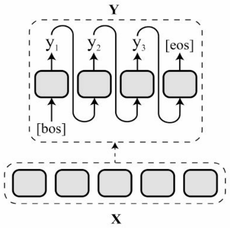
Non-autoregressive decoding



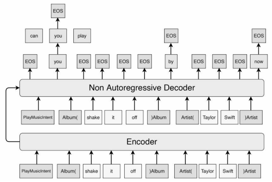
- middle ground **between autoregressive and non-autoregressive** decoding

Insertion Transformer [1][2]

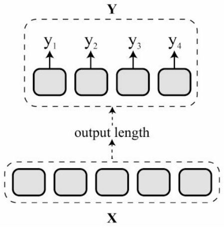
Bonus: Beyond Autoregressive Decoding



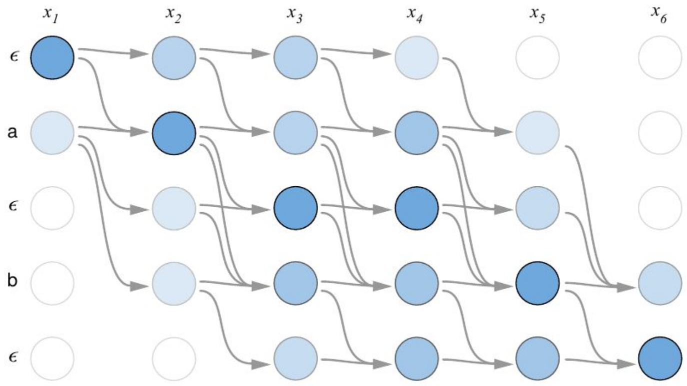
Autoregressive decoding



Insertion Transformer

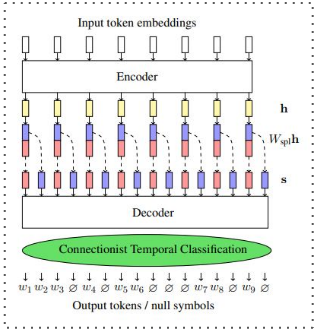


Non-autoregressive decoding



Connectionist Temporal Classification (CTC) layer [1][2]

- **algorithm for sequence alignment**
- useful for automatic speech recognition
- can be stacked on top of the non-autoregressive decoder



Bonus: Reverse-Engineering Decoding Strategies

Bonus: Reverse-Engineering Decoding Strategies

Reverse-Engineering Decoding Strategies Given Blackbox Access to a Language Generation System

Daphne Ippolito* dei@google.com
Nicholas Carlini* ncarlini@google.com
Katherine Lee* katherinelee@google.com
Milad Nasr* miladnasr@google.com
Yun William Yu† ywyu@math.toronto.edu

Abstract

Neural language models are increasingly deployed into APIs and websites that allow a user to pass in a prompt and receive generated text. Many of these systems do not reveal generation parameters. In this paper, we present methods to reverse-engineer the decoding method used to generate text (i.e., top-k or nucleus sampling). Our ability to discover which decoding strategy was used has implications for detecting generated text. Additionally, the process of discovering the decoding

sided die, we found that it only returns 14 of the 20 options, even though all should be equally likely.

Prior work has shown that knowing the decoding method makes it easier to detect whether a writing sample was generated by a language model or else was human-written (Ippolito et al., 2020). As generated text proliferates on the web, in student homework, and elsewhere, this disambiguation is becoming increasingly important.

Concurrent work to ours by Naseh et al. (2023) has developed similar strategies for detecting decoding

https://aclanthology.org/2023.inlg-main.28/

Stealing the Decoding Algorithms of Language Models

Ali Naseh
University of Massachusetts Amherst
Amherst, Massachusetts, USA
anaseh@cs.umass.edu

Kalpesh Krishna
University of Massachusetts Amherst
Amherst, Massachusetts, USA
kalpesh@cs.umass.edu

Mohit Iyyer
University of Massachusetts Amherst
Amherst, Massachusetts, USA
miyyer@cs.umass.edu

Amir Houmansadr
University of Massachusetts Amherst
Amherst, Massachusetts, USA
amir@cs.umass.edu

ABSTRACT

A key component of generating text from modern language models (LM) is the selection and tuning of *decoding algorithms*. These algorithms determine how to generate text from the internal probability distribution generated by the LM. The process of choosing a decoding algorithm and tuning its hyperparameters takes significant time, manual effort, and computation, and it also requires extensive human evaluation. Therefore, the identity and hyperparameters of such decoding algorithms are considered to be extremely valuable to their owners. In this work, we show, for the first time, that an adversary with typical API access to an LM can steal the type and hyperparameters of its decoding algorithms at very low monetary costs. Our attack is effective against popular LMs used in text generation APIs, including GPT-2, GPT-3 and GPT-Neo. We demonstrate the feasibility of stealing such information with only a *few dollars*, e.g., \$0.8, \$1, \$4, and \$40 for the four versions of GPT-3.

GPT-2 [37], GPT-3 [4] and GPT-Neo [3] have been shown to generate high-quality texts for these tasks. To generate a sequence of tokens, LMs produce a probability distribution over the vocabulary at each time step, from which the predicted token is drawn. Enumerating all possible output sequences for a given input and choosing the one with the highest probability is intractable; furthermore, relatively low-probability sequences may even be desirable for certain tasks (e.g., creative writing). Therefore, LMs rely on *decoding algorithms* to decide which output tokens to produce based on their probabilities, i.e., to decode the text.

As shown in the literature [11], the choice of the decoding algorithm and its hyperparameters is critical to the performance of the LM on text generation tasks. Thus, users of many LM-based APIs are offered a choice of decoding algorithms and also the ability to adjust any corresponding hyperparameters. For example, in machine translation, beam search is more common than other methods; however, in story generation, sampling-based methods

https://people.cs.umass.edu/~amir/papers/CCS23-LM-stealing.pdf

Bonus: Reverse-Engineering Decoding Strategies

- With **API access**, answers to certain questions (dice rolls, months, …) can be used to **estimate the *k* and *p* parameters** of the stochastic algorithms

  > *ChatGPT only returns 14 of the 20 options for a 20-sided dice roll*

- Distinguishing between **top-k** and **top-p**:
  If two prompts yield very different predictions of k, then top-k is probably not used.

- With access to **model's full distribution**, we can distinguish also between other algorithms (greedy vs. beam search vs. top-k vs. top-p …)

Links

- Huggingface models
- Awesome LLM: curated list of resources
- Transformer inference: 3D visualization
- Huggingface decoding algorithms overview
- Huggingface text generation strategies (includes a few extra ones)
- Common pitfalls when generating text with LLMs
- Visualizing decoding strategies
- Minimum Bayes Risk decoding

Further reading

- On Decoding Strategies for Neural Text Generators (Wiher et al., 2022)
  - Language generation tasks vs. decoding strategies.
- If beam search is the answer, what was the question? (Meister et al., 2020)
  - Why does beam search work so well?
- Understanding the Properties of Minimum Bayes Risk Decoding in Neural Machine Translation (Muller and Sennrich, 2021)
  - When can MBR be useful?