

# LLM Inference

Zdeněk Kasner

10 March 2025



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



unless otherwise stated

# Today's learning outcomes

After today's class, you should be able to:

- **Understand how to generate text** with a Transformer-based language model.
- **Explain differences between decoding algorithms** and the role of decoding parameters.
- **Choose a suitable LLM** for your task.
- **Run a LLM locally** on your computer or computational cluster.

**Generating text**

# Recap: Training

## Model stages:

random neural  
network

1



“autocomplete on steroids”

*base / foundational model*

2



assistant

*instruction-tuned model*

3



helpful assistant

## Training stages:

1

Pre-training



Prague is the capital of Czechia (...)

2

Instruction tuning



user: What is the capital of Czechia?  
assistant: Prague

3

Human preference optimization

user: What is the capital of Czechia?



answer #1: Prague.  
answer #2: The capital of Czechia is Prague.

This lecture: **LLM inference**.

= We have a trained model and we want to use it.

Question: What is the difference between **inference**, **generation**, and **decoding**?

## Inference

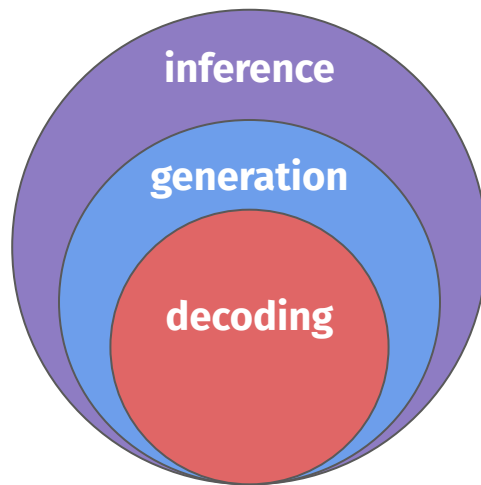
The concept of using a trained model for **making predictions** on new data (for classification, sequence tagging, text generation, ...).

## Generation

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

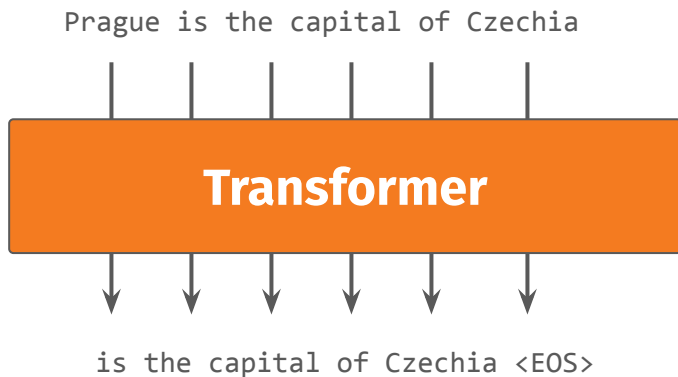
## Decoding

The algorithm of **selecting the next token** using the model's internal representation.



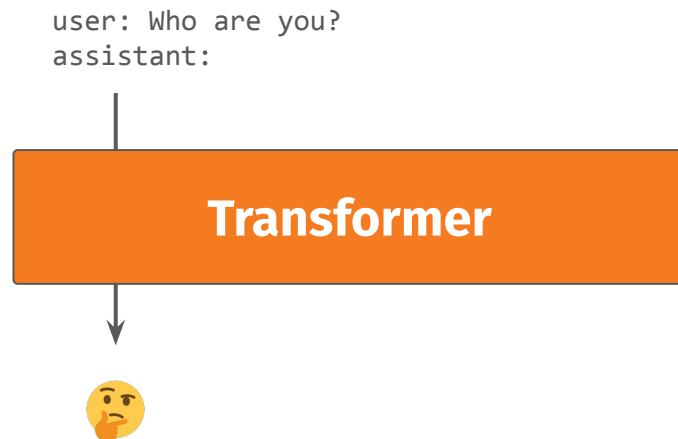
# Training vs. inference

## Training



**Teacher forcing:** We know what token should come next, so we use it to train the model.

## Inference



**Decoding:** We need to select what token should come next.

# What happens during LLM inference?

<https://bbycroft.net/llm>

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

**EQUATIONS**

**CODE**

**DIAGRAMS**

**ANIMATED 3D  
VISUALIZATIONS**



imgflip.com

# Generating text

For every sequence, the LLM generates a **probability distribution** over the vocabulary of tokens.

## To generate text:

1. Start with a sequence of tokens (“prompt”).
2. Feed the sequence into the LLM.
3. Select the next token from the model-generated probability distribution.
4. Append the selected token to the sequence.
5. Repeat from (2).

→ **Autoregressive decoding**

# Autoregressive decoding

$t=1$

<BOS>



I



Transformer



$P(y_t | \text{<BOS>, "I"})$

a  
aardwark  
am

...

I

...

the  
walrus

...

zyzzyva



which token to select?

# Autoregressive decoding

$t=1$

<BOS>

I

Transformer

$P(y_t | \text{<BOS>, "I"})$

a  
aardwark  
**am**

...

I

...

the  
walrus

...

zyzzzyva

**am**

the most probable one?

# Autoregressive decoding

$t=2$

<BOS>

I

am

Transformer

$P(y_t | \text{<BOS>, "I", "am"})$

a  
aardwark  
am

...

I

...

the  
walrus

...

zyzzzyva

the

every single time?

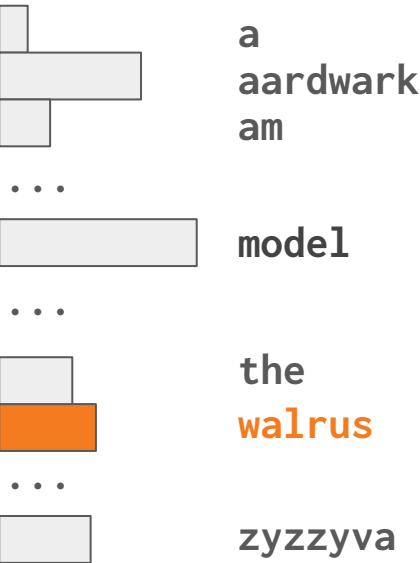
# Autoregressive decoding

t=3

<BOS> →  
I →  
am →  
the →



$P(y_t | \text{<BOS>, "I", "am", "the"})$



walrus

YOLO!

# Autoregressive decoding

$t=4$

<BOS>



I



am



the



walrus

**Transformer**

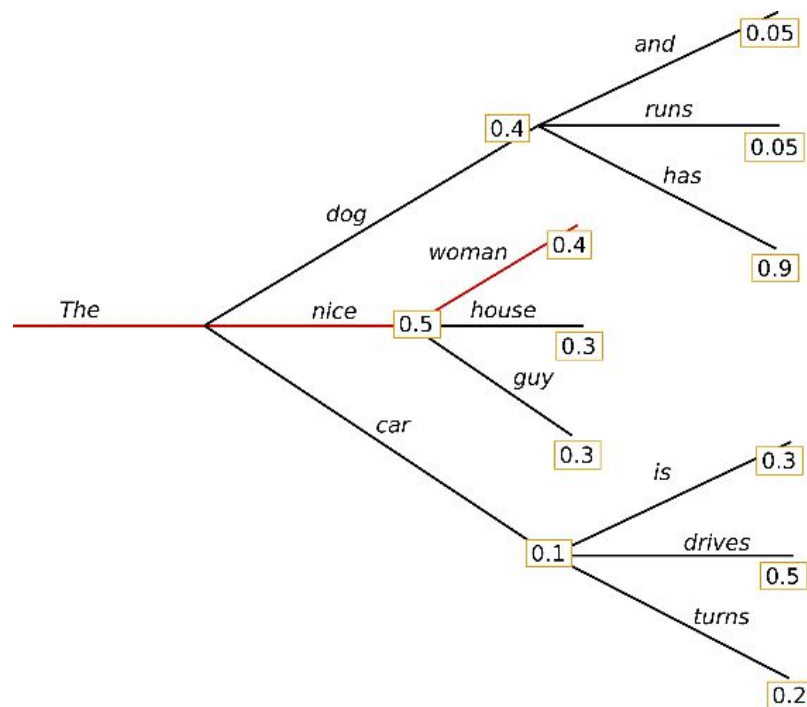


# Decoding algorithms

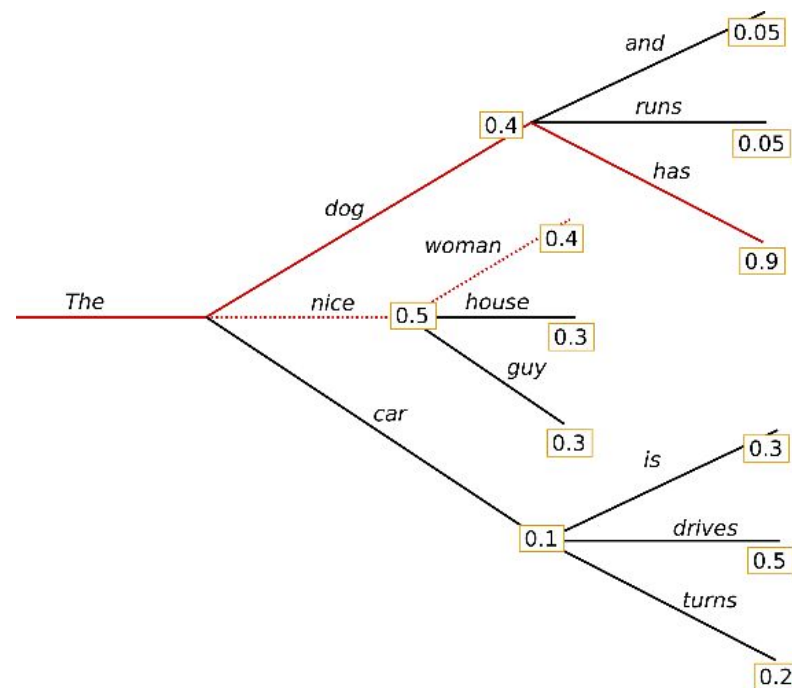
- Selecting the **most probable token** in each step  $t$ :

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

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



- 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  $k \rightarrow$  slower algorithm
  - $k>1$  allows re-ranking results



# 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:

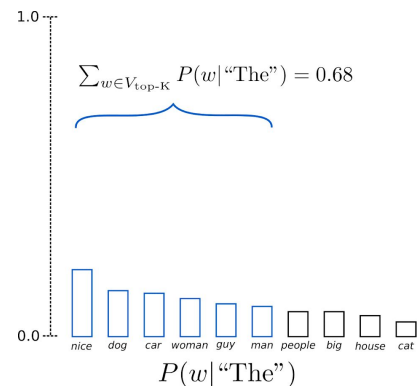
$$y^* = \arg \max_{y \in \mathcal{Y}} P(y) = \arg \max_{y \in \mathcal{Y}} \prod_{i=1}^t P(y_i | y_1, \dots, y_{t-1})$$

- Intractable (exponential search space)
- Can be approximated by greedy decoding or beam search
- The mode may not be a good solution! ([1], [2])
  - e.g. an empty sequence

# Top-k sampling

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

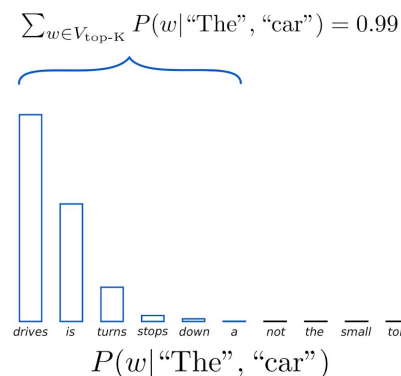
- Selecting the token in each step **randomly from  $k \in \{1, \dots, |V|\}$  most probable tokens**
- The truncated distribution is re-weighted using softmax



**step #1**

**prefix** = "The"  
→ sampling from  
{nice, dog, car,  
woman, guy, man}

**cum.prob.** = 0.68



**step #2**

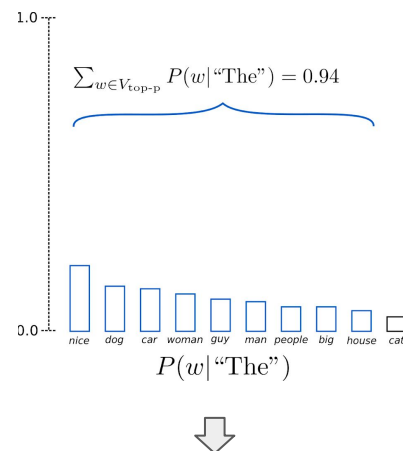
**prefix** = "The car"  
→ sampling from  
{drives, is, turns,  
stops, down, a}

**cum.prob.** = 0.99

# Top-p (nucleus) sampling

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

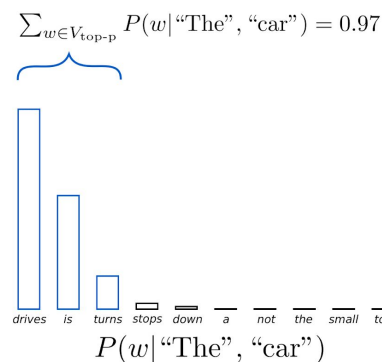
- Sampling from **“nucleus”**: set of the most probable tokens with combined probability summing to  $p \in (0, 1]$
- Similar to top-k sampling, but with a variable  $k$  in each step.



## step #1

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

**cum.prob.** = 0.94  
(>0.9)



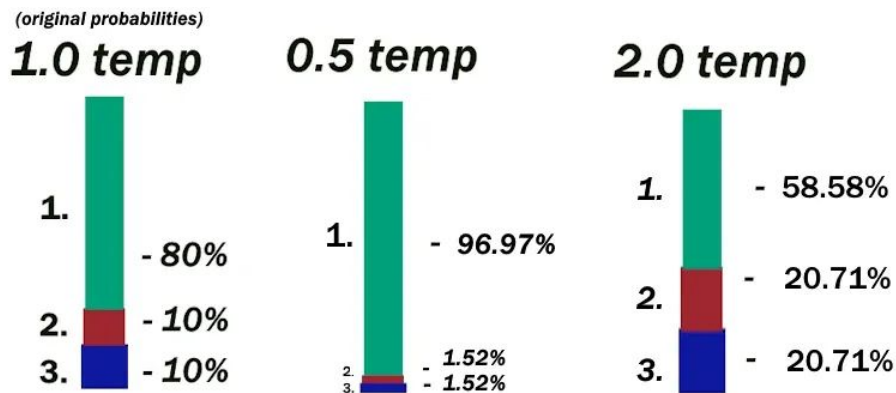
## step #2

**prefix** = "The car"  
→ sampling from  
{drives, is, turns}

**cum.prob.** = 0.97  
(>0.9)

The shape of the distribution can be adjusted using the **temperature  $T$** :

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





r/MachineLearning • 8 mo. ago  
zyl1024



## [D] What happened to "creative" decoding strategy?

Discussion

For GPT-2 and most models at that time, the naive greedy decoding is extremely prone to generating repetitive and nonsensical outputs very fast, and many techniques, such as top-p sampling, nucleus sampling, repetition penalty, n-gram penalty, etc. are needed. (e.g. <https://arxiv.org/pdf/1904.09751> )

For recent LLMs, I haven't been using any of these tricks, and instead, any temperature between 0 and 1 seems to work just fine. The only repetitive generation that I've observed seem to be in math reasoning, when the model wants to do some exhaustive search that didn't succeed.

So are all these custom decoding strategies a thing of the past, and we don't need to worry about degenerate content generation anymore?



23



12

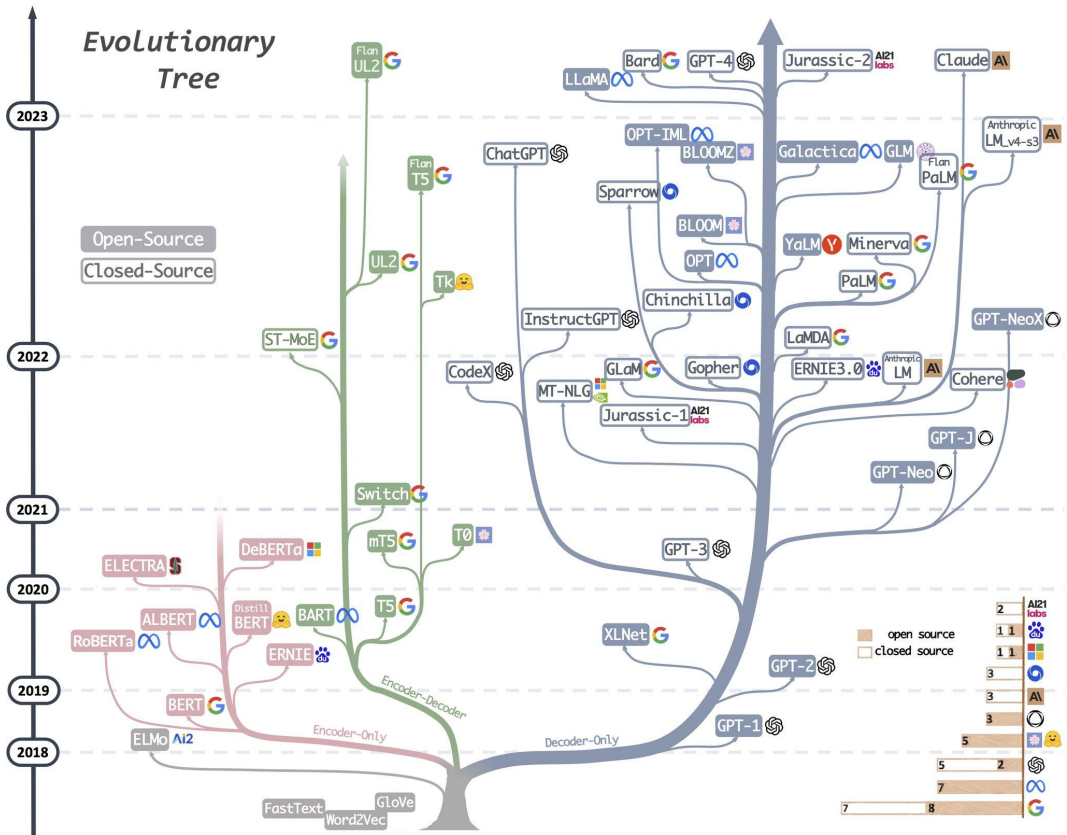


Share

# **Navigating the LLM zoo**

# LLM evolutionary tree

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



**SOURCE:** <https://informationisbeautiful.net/visualizations/the-rise-of-generative-ai-large-language-models-llms-like-chatgpt/>

ranked by capabilities, sized by billion parameters used for training

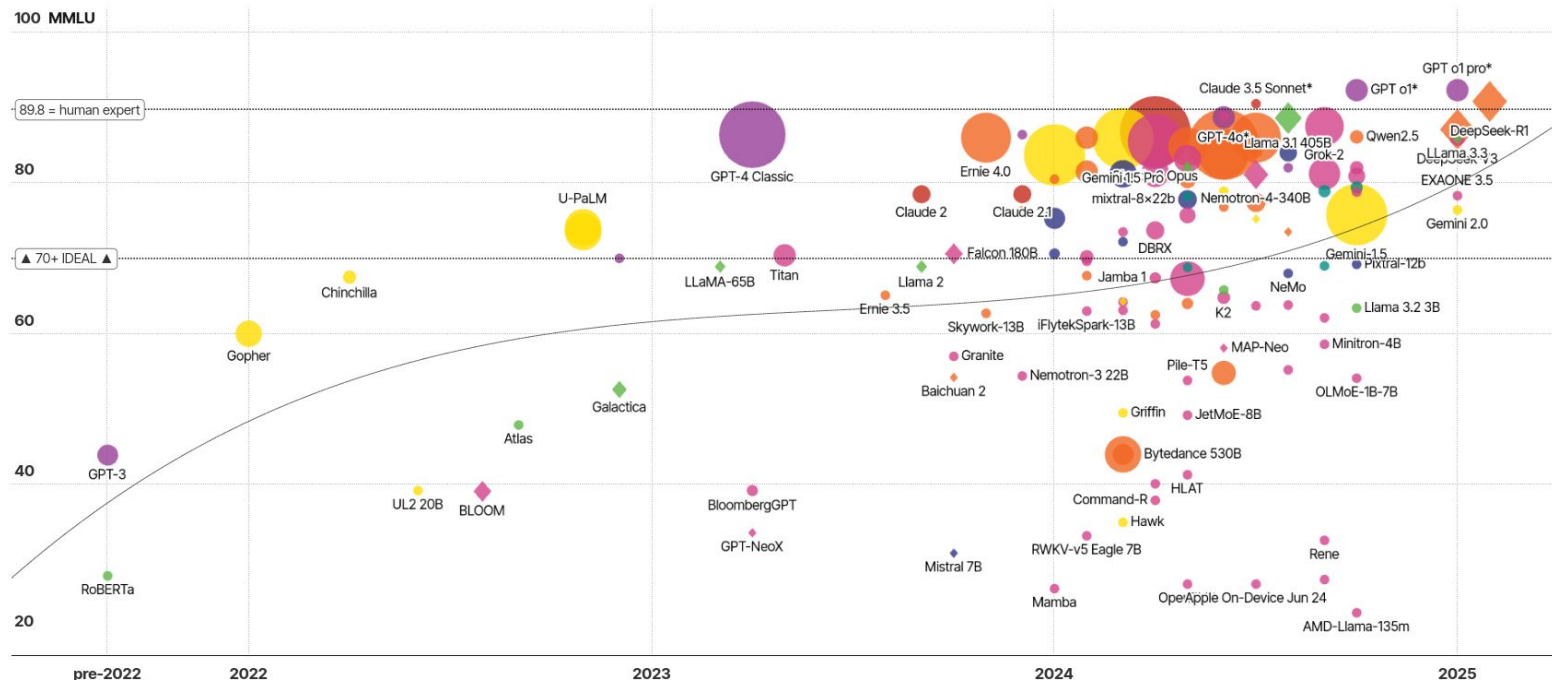
CLICK LEGEND ITEMS TO FILTER

anthropic chinese google meta microsoft mistral openAI other

Parameters (Bn)  open access

🔍 search...

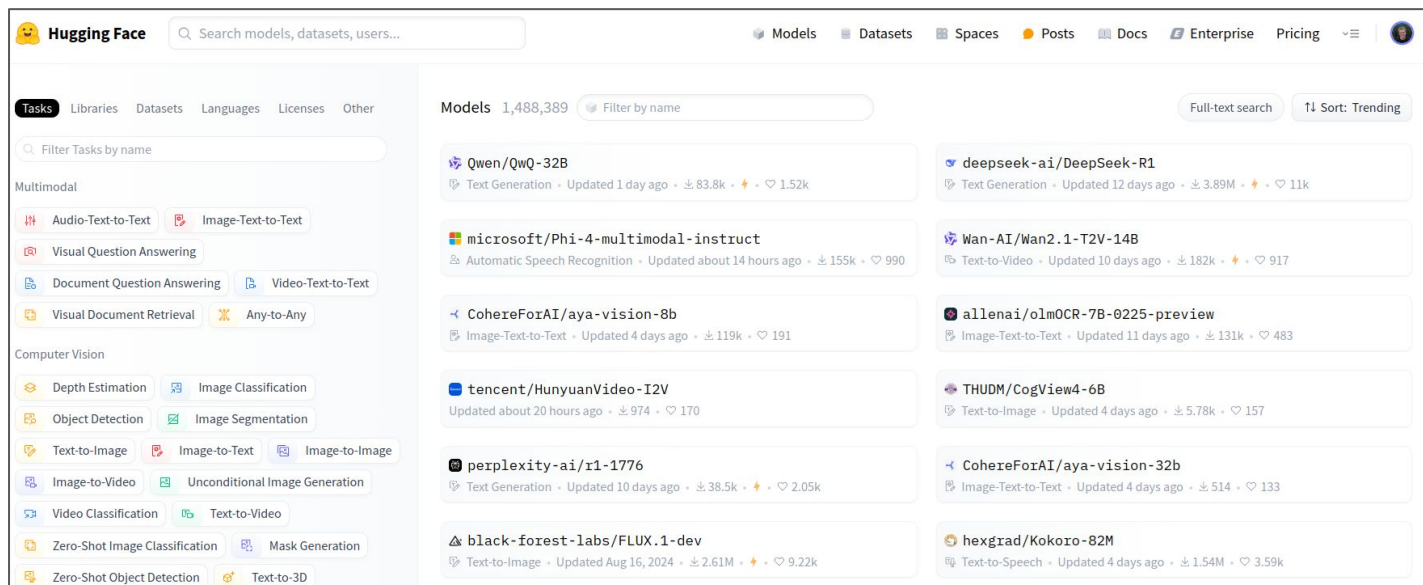
show only: all



# Model sources and leaderboards

HuggingFace: the largest repository of open LLMs.

As of March 2025, it contains **~1.5M models** (many of these are derivatives).



# Model sources and leaderboards

## Chatbot Arena: Elo rating of LLMs.

For a pair of answers from different models, users decide which is better.

**Chatbot Arena LLM Leaderboard: Community-driven Evaluation for Best LLM and AI chatbots**

Discord | Twitter | 小红书 | Blog | GitHub | Paper | Dataset | Kaggle Competition

Chatbot Arena is an open platform for crowdsourced AI benchmarking, developed by researchers at UC Berkeley [SkyLab](#) and [LMArena](#). With over 1,000,000 user votes, the platform ranks best LLM and AI chatbots using the Bradley-Terry model to generate live leaderboards. For technical details, check out our [paper](#).

Chatbot Arena thrives on community engagement — cast your vote to help improve AI evaluation!

Join us in our NEW Discord server: [discord.gg/LMArena](https://discord.gg/LMArena)

Language Overview Price Analysis WebDev Arena Vision Text-to-Image Copilot Arena Leaderboard Arena-Hard-Auto

Total #models: 211. Total #votes: 2,736,442. Last updated: 2025-03-02.

Code to recreate leaderboard tables and plots in this [notebook](#). You can contribute your vote at [lmarena.ai](#)

Category: Overall


Apply filter: ☐ Style Control ☐ Show Deprecated

Overall Questions  
#models: 211 (100%) #votes: 2,736,442 (100%)

Rank* (UB)	Rank (StyleCtrl)	Model	Arena Score	95% CI	Votes	Organization	License
1	2	<a href="#">Grok-3-Preview-02-24</a>	1412	+8/-10	3364	xAI	Proprietary

# Model sources and leaderboards

Open LLM Leaderboard: ratings of open LLMs on benchmarks.



## Open LLM Leaderboard

Comparing Large Language Models in an open and reproducible way

4400 / 4400 Advanced Filters

Supports strict search and regex • Use semicolons for multiple terms

**Quick Filters** For Edge Devices • 741 For Consumers • 410 Mid-range • 3076 For the GPU-rich • 164 ☐ Only Official Providers • 470

table options column visibility

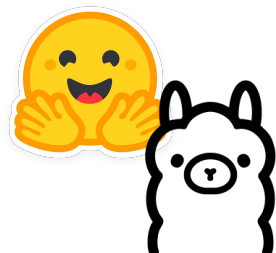
	Rank	Type	Model	Average	IFEval	BBH	MATH	GPQA	MUSR	MMLU-...	CO <sub>2</sub> Cost
1	1	🔹	<a href="#">MaziyarPanahi/calme-3.2-instruct-78b</a>	● 52.08 %	80.63 %	62.61 %	40.33 %	20.36 %	38.53 %	70.03 %	66.01 kg
2	2	💬	<a href="#">MaziyarPanahi/calme-3.1-instruct-78b</a>	● 51.29 %	81.36 %	62.41 %	39.27 %	19.46 %	36.50 %	68.72 %	64.44 kg
3	3	💬	<a href="#">dfurman/CalmeRys-78B-Orpo-v0.1</a>	● 51.23 %	81.63 %	61.92 %	40.63 %	20.02 %	36.37 %	66.80 %	25.99 kg

# Rules of thumb for selecting a model

- Try a **general-purpose model** first.
  - You can specify your task using in-context learning.
  - RAG can help you with a custom knowledge base.
- You may want to use a **fine-tuned model**, but think carefully about which data it was finetuned on.
- You probably **do not want an off-the-shelf base model** unless you want to fine-tune it (or you are interested in LM on its own).
- Out of the newest models, select the **largest model you can support**.

**Running LLMs locally**

# How to use LLMs



# Frameworks for running open LLMs

Huggingface transformers: Python library for loading models from the Huggingface model repository.



## Transformers

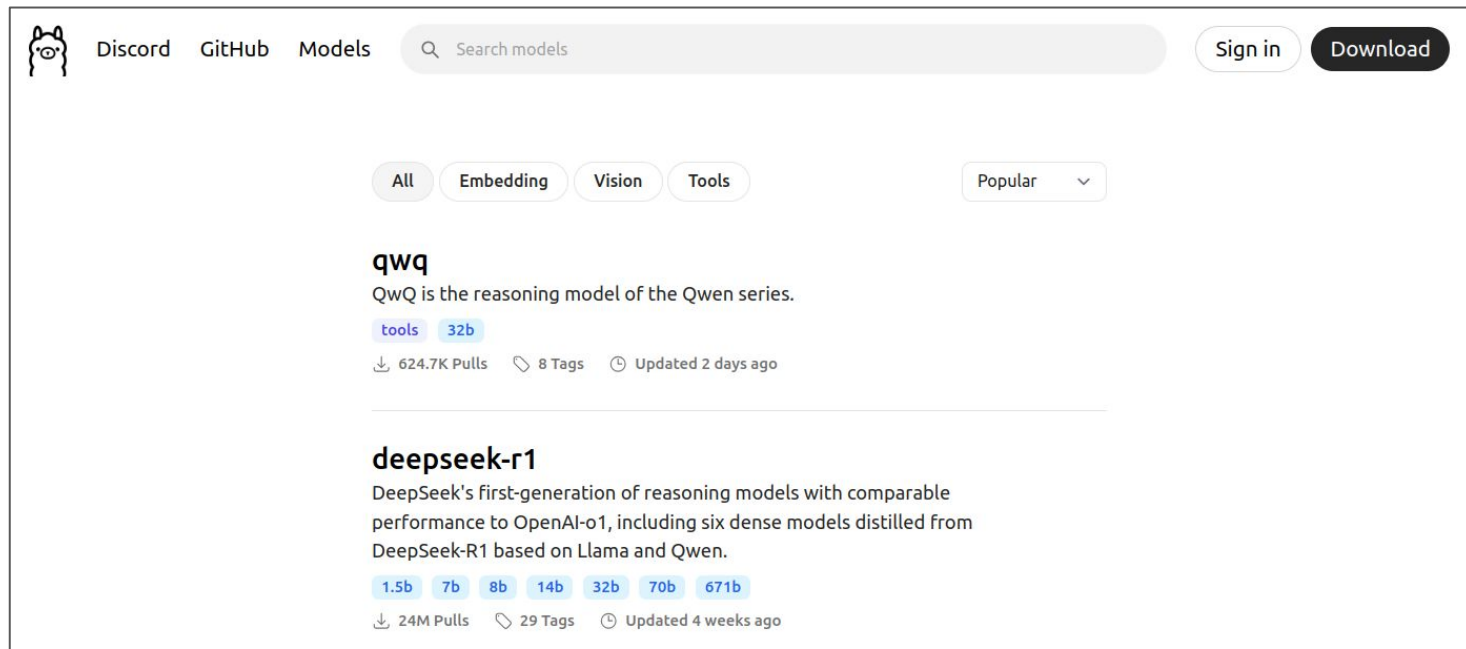
build failing license Apache-2.0 website online release v4.49.0 Contributor Covenant v2.0 adopted DOI 10.5281/zenodo.7391177

English | 简体中文 | 繁體中文 | 한국어 | Español | 日本語 | हिन्दी | Русский | Português | తెలుగు | Français | Deutsch | Tiếng Việt | اردو | العربية |

State-of-the-art Machine Learning for JAX, PyTorch and TensorFlow

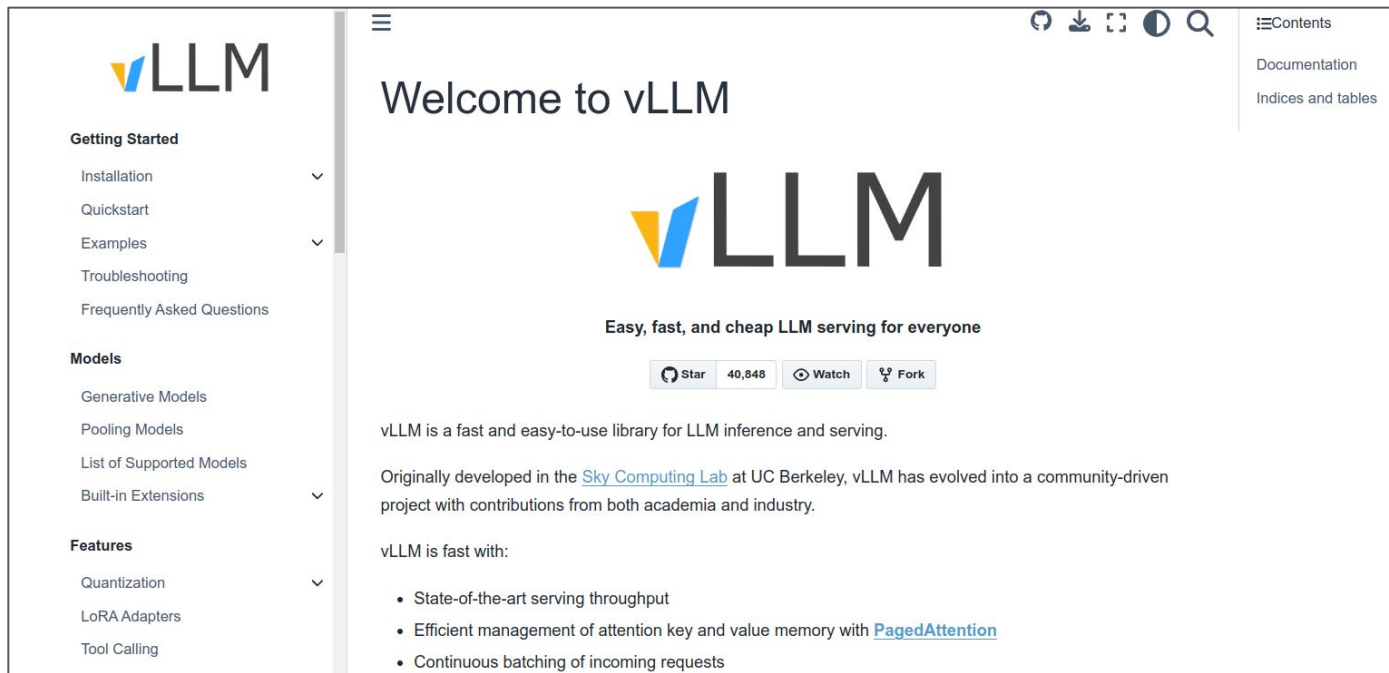
# Frameworks for running open LLMs

**Ollama**: running a local server, easy to use, focus on quantized models



# Frameworks for running open LLMs

**vLLM**: efficient library for serving of LLMs on an enterprise level



The screenshot shows the vLLM project homepage. On the left is a navigation sidebar with sections: 'Getting Started' (Installation, Quickstart, Examples, Troubleshooting, Frequently Asked Questions), 'Models' (Generative Models, Pooling Models, List of Supported Models, Built-in Extensions), and 'Features' (Quantization, LoRA Adapters, Tool Calling). The main content area has a large 'Welcome to vLLM' header with the vLLM logo. Below the logo is the tagline 'Easy, fast, and cheap LLM serving for everyone' and GitHub statistics (40,848 stars). The text describes vLLM as a fast and easy-to-use library for LLM inference and serving, originally developed at UC Berkeley. It lists three features: state-of-the-art serving throughput, efficient memory management with PagedAttention, and continuous batching of requests. A right-hand sidebar contains links to 'Contents', 'Documentation', and 'Indices and tables'.

**vLLM**

Getting Started

- Installation
- Quickstart
- Examples
- Troubleshooting
- Frequently Asked Questions

**Models**

- Generative Models
- Pooling Models
- List of Supported Models
- Built-in Extensions

**Features**

- Quantization
- LoRA Adapters
- Tool Calling

## Welcome to vLLM

**vLLM**

Easy, fast, and cheap LLM serving for everyone

Star 40,848 Watch Fork

vLLM is a fast and easy-to-use library for LLM inference and serving.

Originally developed in the [Sky Computing Lab](#) at UC Berkeley, vLLM has evolved into a community-driven project with contributions from both academia and industry.

vLLM is fast with:

- State-of-the-art serving throughput
- Efficient management of attention key and value memory with [PagedAttention](#)
- Continuous batching of incoming requests

Contents

- Documentation
- Indices and tables

## Demo time

[https://huggingface.co/docs/transformers/llm\\_tutorial](https://huggingface.co/docs/transformers/llm_tutorial)

[https://mlabonne.github.io/blog/posts/2023-06-07-Decoding\\_strategies.html](https://mlabonne.github.io/blog/posts/2023-06-07-Decoding_strategies.html)

- [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](#)

**Bonus: Extra decoding algorithms**

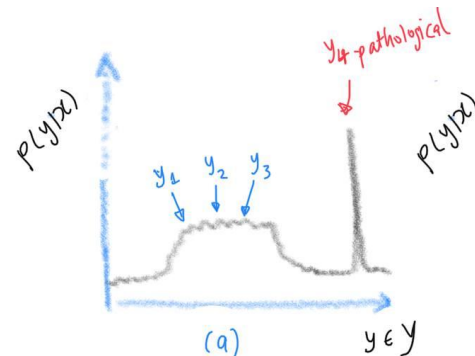
# Minimum Bayes Risk (MBR) Decoding

source: [Minimum Bayes Risk Decoding](#)

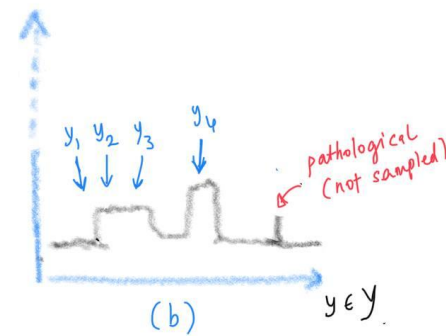
- Selecting the sequence most similar to other sequences = “**consensus decoding**”

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

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



MAP 😞



MBR 😊

- 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:
  - $\tau$  (tau) - the target perplexity
  - $\eta$  (eta) - learning rate

---

**Algorithm 1:** Adaptive top- $k$  sampling for perplexity control

---

Target cross entropy  $\tau$ , maximum cross entropy  $\mu = 2 * \tau$ , learning rate  $\eta$

**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):  $k = \left( \frac{e^{2\mu}}{1 - N^{-e}} \right)^{\frac{1}{8}}$

    Sample the next word  $X$  using top- $k$  sampling

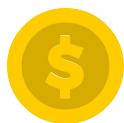
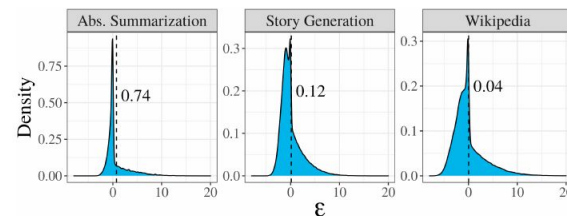
    Compute error:  $e = \mathfrak{S}(X) - \tau$

    Update  $\mu$ :  $\mu = \mu - \eta * e$

**end**

---

- 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



$$p(\text{H}) = 0.75$$

$$p(\text{T}) = 0.25$$

**H H H H** → most probable sequence

**H T H H** → typical sequence

# 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?