# CSC412
# Probabilistic Learning and Reasoning
## Week 11 LLM Interpretability:
## Sparse Auto Encoders, MoEs & Constrained Decoding

Denys Linkov

University of Toronto

# Why does Claude love the Golden Gate Bridge?

- On May 23rd, 2024 a mysterious version of Claude Appeared.
- It showed strong preference for the Golden Gate Bridge.
- Why?
- Understanding such behavior in LLMs is a challenge.

# Recap: GPT-1 Exerpt (Paraphrase)

Given an unsupervised corpus of tokens $\mathcal{U} = \{u_1, \ldots, u_n\}$, maximize the following likelihood:

$$L_1(\mathcal{U}) = \sum_i \log P(u_i | u_{i-k}, \ldots, u_{i-1}; \Theta) \tag{1}$$

where $k$ is the size of the context window, and the conditional probability $P$ is modeled using a neural network with parameters $\Theta$.

Use multi-headed self-attention operation over the input context tokens followed by position-wise feedforward layers to produce an output distribution over target tokens:

$$h_0 = UW_e + W_p$$
$$h_l = \text{transformer\_block}(h_{l-1}) \quad \forall l \in [1, n]$$
$$P(u) = \text{softmax}(h_n W_e^T)$$

where $U = (u_{-k}, \ldots, u_{-1})$ is the context vector of tokens, $n$ is the number of layers, $W_e$ is the token embedding matrix, and $W_p$ is the

# Large Language Models are Large

- Modern LLMs follow the autoregressive pattern, but with billions of parameters.
- For context, this is a visual representation of a nanoGPT model, 85,584 parameters.
- GPT-3 is 175 Billion parameters.
- The ratio between one neuron to nanoGPT is the same as nanoGPT to a 7.3B parameter model (!!!).
- Very large, so what goes on inside the different layers?

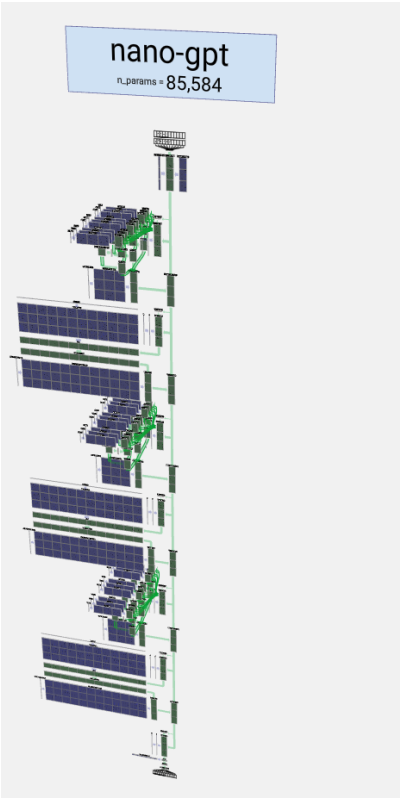



Figure: Nano GPT Architecture (Bycroft, 2022)

# LLM Architecture

- Parameters come in two main components
  - MLP and Attention blocks
- Other components like activation functions, embeddings layer, positional embeddings etc

| | 8B | 70B | 405B |
|---|---|---|---|
| Layers | 32 | 80 | 126 |
| Model Dimension | 4,096 | 8192 | 16,384 |
| FFN Dimension | 14,336 | 28,672 | 53,248 |
| Attention Heads | 32 | 64 | 128 |
| Key/Value Heads | 8 | 8 | 8 |
| Peak Learning Rate | $3 \times 10^{-4}$ | $1.5 \times 10^{-4}$ | $8 \times 10^{-5}$ |
| Activation Function | SwiGLU | | |
| Vocabulary Size | 128,000 | | |
| Positional Embeddings | RoPE ($\theta = 500,000$) | | |

Figure: LLaMa 3 architecture (Dubey, 2024)

# How can we understand LLM answers?

- As we covered in previous lectures, Autoencoders can be used for understanding internal representations by compressing key features into a latent space.
- Sparse Autoencoders (SAEs) can help identify concepts that activate specific layers by attempting to define granular features.
- This allows us to probe the internal workings of LLMs.

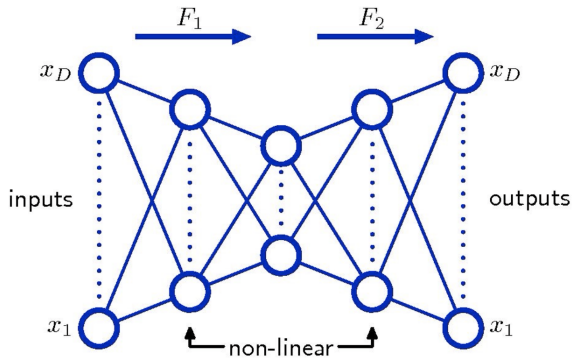
# Recap: Autoencoders

Autoencoders reconstruct their input via an encoder and a decoder.

- **Encoder**: $g(x) = z \in F, \quad x \in X$
- **Decoder**: $f(z) = \tilde{x} \in X$
- where $X$ is the data space, and $F$ is the feature (latent) space.
- $z$ is the code, compressed representation of the input, $x$. It is important that this code is a bottleneck, i.e. that

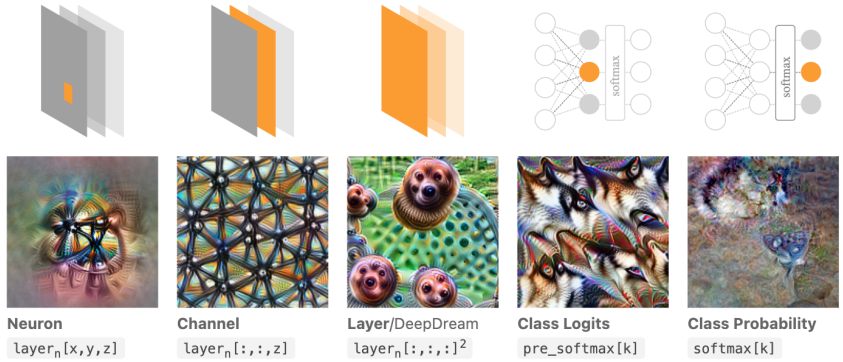
$$\dim F \ll \dim X$$

- Goal: $\tilde{x} = f(g(x)) \approx x$.



# Computer vision analogies

- Computer vision interpretability offers us some insights of the challenges of understanding deep learning.
- Some challenges
  - Networks are large
  - Neurons activations are sometimes sparse, sometimes Polysemantic
  - Polysemantic meaning they capture mutiple concepts within them
  - Many LLM neurons are “dead” neurons ex OPT (Voita, 2023)



# A trip back to MNIST in 2009

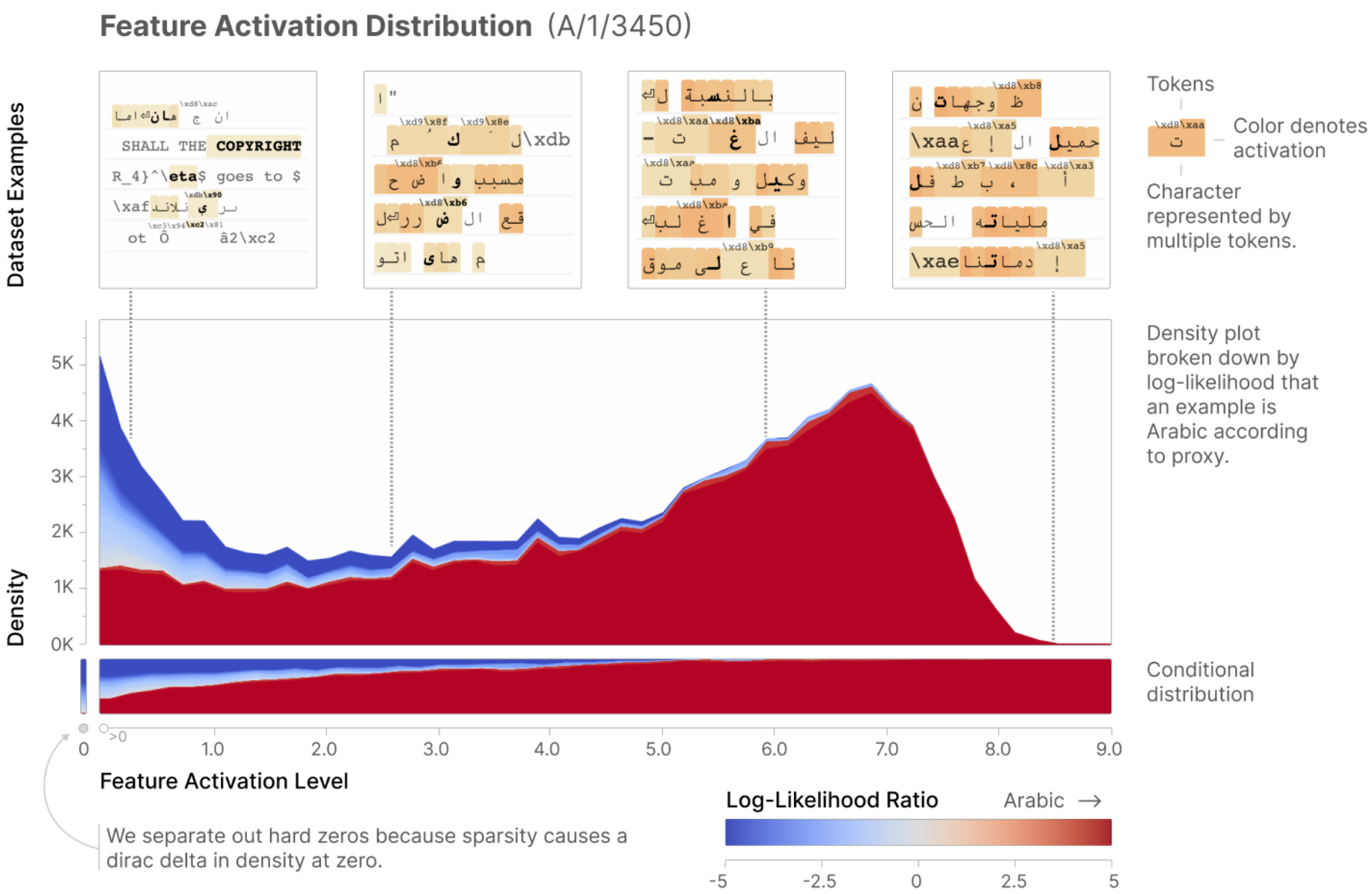
Consider a Deep Belief Network with j layers. In particular, layers j-1 and j form an RBM from which we can sample using block Gibbs sampling, which successively samples from $p(h_{j-1}|h_j)$ and $p(h_j|h_{j-1})$, denoting by $h_j$ the bi-nary vector of units from layer j. Along this Markov chain, we propose to "clamp" unit $h_{ij}$, and only this unit, to 1. We can then sample inputs x by performing ancestral top-down sampling in the directed belief network going from layer $j-1$ to the input, in the DBN. This will produce a distribution that we shall denote by $p_j(x|h_{ij}=1)$ where $h_{ij}$ is the unit that is clamped, and $p_j$ denotes the depth-j DBN containing only the first j layers. This procedure is similar to and inspired from experiments by Hinton et al. (2006a), where the top layer RBM is trained on the representations learned by the previous RBM and the label as a one-hot vector; in that case, one can"clamp" the label vector to a particular configuration and sample from a particular class distribution $p(x|class=k)$. (Erhan, 2009)

# Idea: Extract Features that are monosemantic*

- "One of the roadblocks to a better understanding of neural networks' internals is polysemanticity, where neurons appear to activate in multiple, semantically distinct contexts. Polysemanticity prevents us from identifying concise, humanunderstandable explanations for what neural networks are doing internally. One hypothesised cause of polysemanticity is superposition, where neural networks represent more features than they have neurons by assigning features to an overcomplete set of directions in activation space, rather than to individual neurons (Cunninghamm, 2023)"

- "The most important claim of our paper is that dictionary learning can extract features that are significantly more monosemantic than neurons. In this section, we give a detailed demonstration of this claim for a small number of features which activate in highly specific contexts. (Bricken, 2023)"

# Visualizing Monosemanetic Features



# Pause - What are we trying to do?

Map neuron activations to concepts

- Let $y$ be a class of food, $x$ as a description of food, $z$ as latent variables in a LLM
- Maximize $P(y|x, z, \theta)$, be able to predict model results when tweaking some latent variables (neurons)
  - Ex - $P(y = sushi|x, z, \theta)$ The probability of predicting "sushi" given a description of sushi and a model/neuron
- Maximize $P(x|y, z, \theta)$, input that most likely data points given some result
  - Ex - $P(x|y = sushi, z, \theta)$, generate a description of sushi
- Maximize $P(z|y, x, \theta)$, input that most likely data points given some result
  - Ex - $P(z|y = sushi, x, \theta)$, find the neuron that is most likely to know about sushi

**LLMs are weird since x and y can be easily swapped.

# SAE Architecture

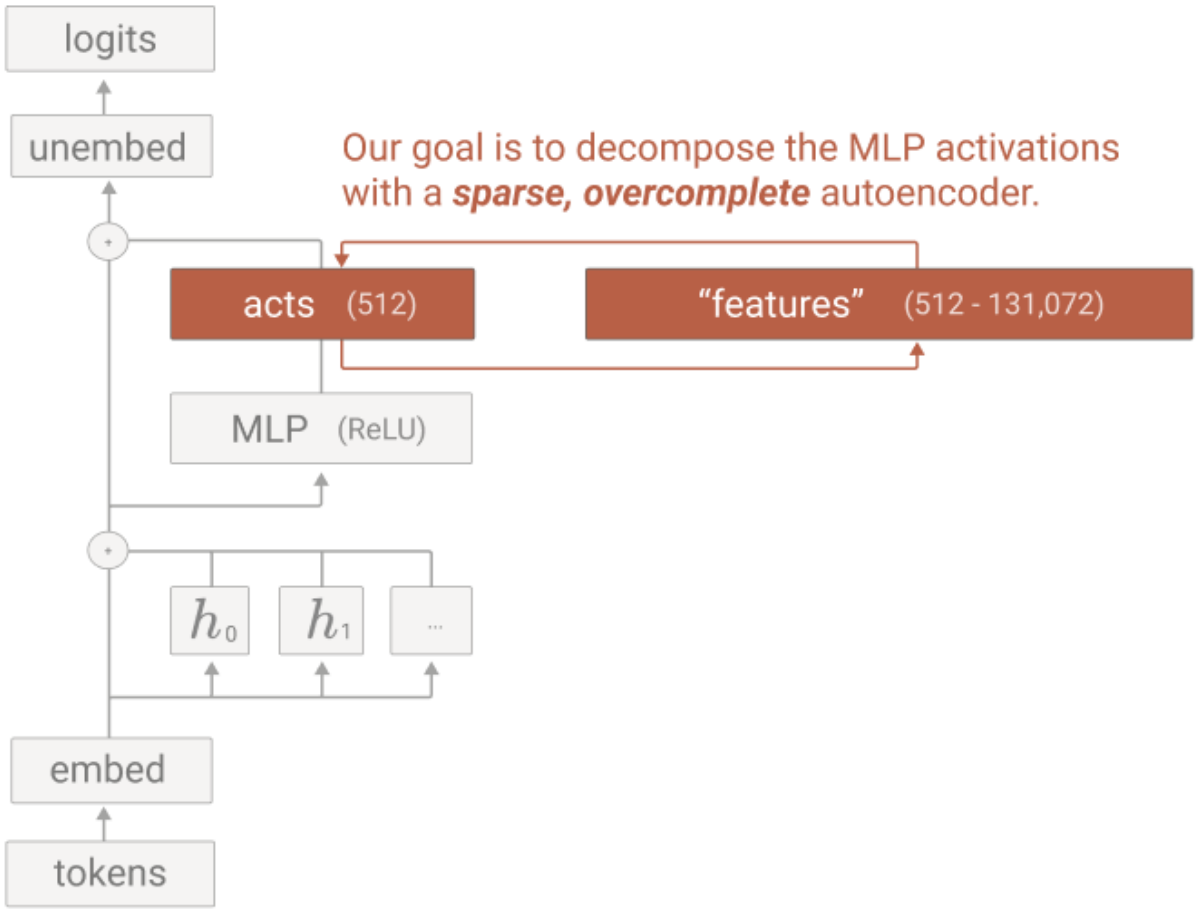



Figure: Training an SAE to extract features (Bricken, 2023)

# SAE Architecture

- Choose the number of features to extract $F$
- Wider than an AE, use an expansion factor $E$
- LLaMa-3 8B has a 14,336 dim vector, for an SAE set $E = 4$
- F is our width, known as dictionary size

$$F = 14,336 * 4 = 57,344$$

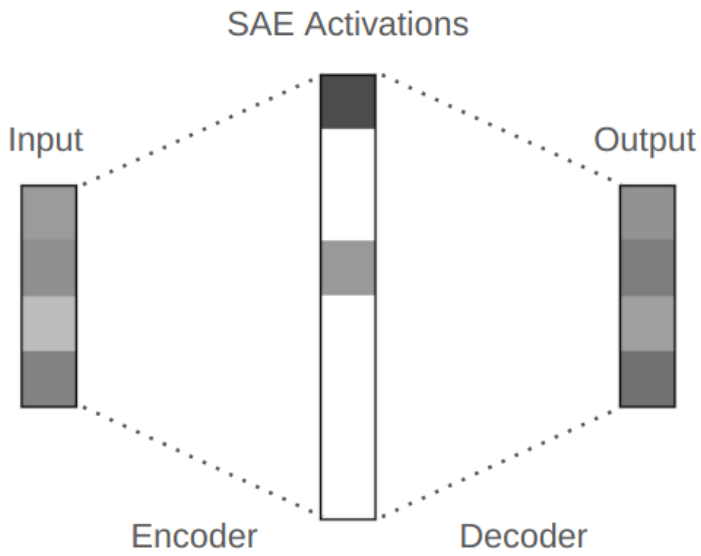



Figure: (Karvonen, 2024)

# SAE Forward Pass

Simple matrix multiplication

- Multiply input by encoder matrix
- Apply a technique to make the SAE portion sparse (RELU)
- Multiply input by decoder matrix

$$f(x) := \text{ReLU}(W_{\text{enc}}(x - b_{\text{dec}}) + b_{\text{enc}})$$

$$\hat{x}(f) := W_{\text{dec}} f + b_{\text{dec}}$$

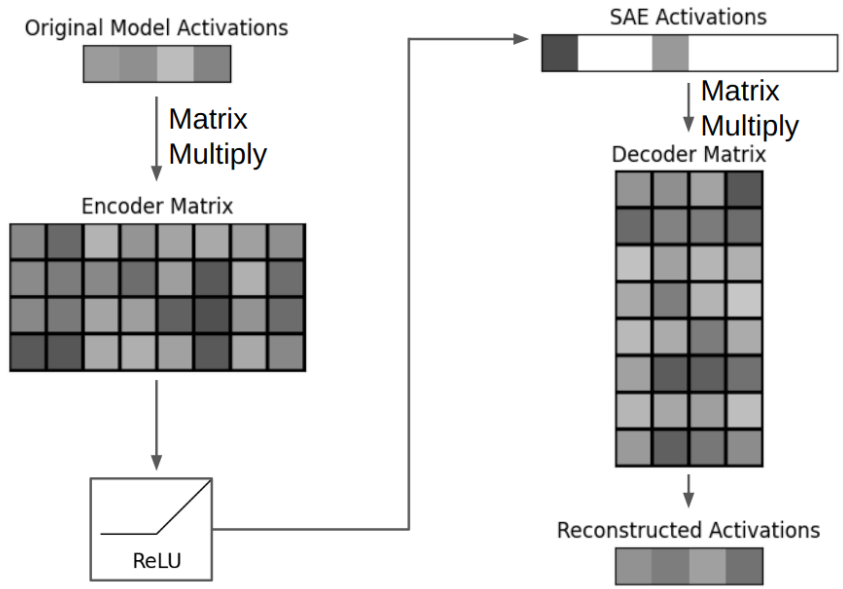



Figure: (Karvonen, 2024)

# Defining the loss function

SAE training balances reconstruction and sparsity.

- Reconstruction loss (L2) and Sparscity by (L1)
- $f(x)$ is active features
- $\lambda$ sparsity penalty

$$L(x) := \|x - \hat{x}(f(x))\|_2^2 + \lambda \|f(x)\|_1$$

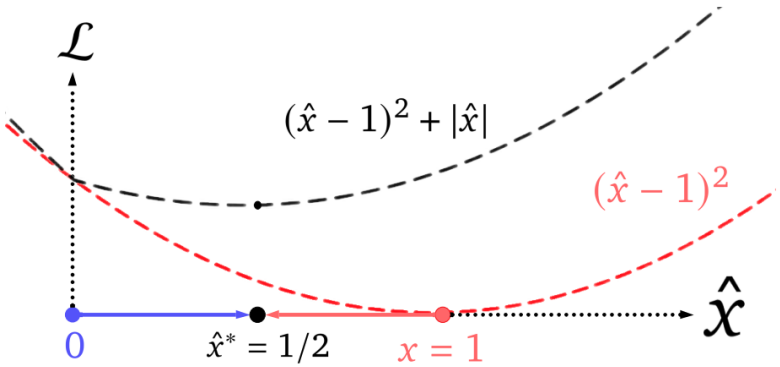



Figure 2 | The L1 penalty in sparse autoencoder causes *shrinkage* – reconstructions are biased towards smaller norms, even when perfect reconstruction is possible.
E.g. a single-feature SAE (with L1 coefficient $\lambda = 1$) reconstructs 1/2 rather than 1 when minimizing Equation (4).

Figure: SAE Loss visulization - (Rajamanoharan, 2024)

# Training an SAE

- Selecting a base model
  - Choose a dataset to train a baseline model, (Bricken, 2023) uses THE PILE which a curated opensource internet set.
  - Select a baseline model architecture, one layer model (Bricken, 2023), or LLM (Gao, 2024),(Rajamanoharan, 2024)
  - Train the baseline model, or use pretrained version
- Training the SAE
  - Initialize your SAE encoder and decoder with some values
  - Select data to pass through your baseline model
  - Select a layer to use for your SAE
  - Use many forward passes of the baseline model to train SAE
  - Data mix is important, can mix in low activation data and also resample
  - Usually you get many "dead latents"
  - OpenAI open sourced some basic training code for SAEs

# Evaluating an SAE

- Manual inspection of SAE activated layers
- Feature density - number of live features (L0)
- Reconstruction loss: How well does the autoencoder reconstruct the activations
- Statistical Tests
- Downstream loss - Replace layer weights with SAE reconstruction
- Testing Toy Models - Easy problems to verify
- Probe loss - handselected easily testable metrics
- From (Bricken, 2023), (Rajamanoharan, 2024), (Gao, 2024)

# Changing the Width of the SAE

- Features can become more granular as width increases

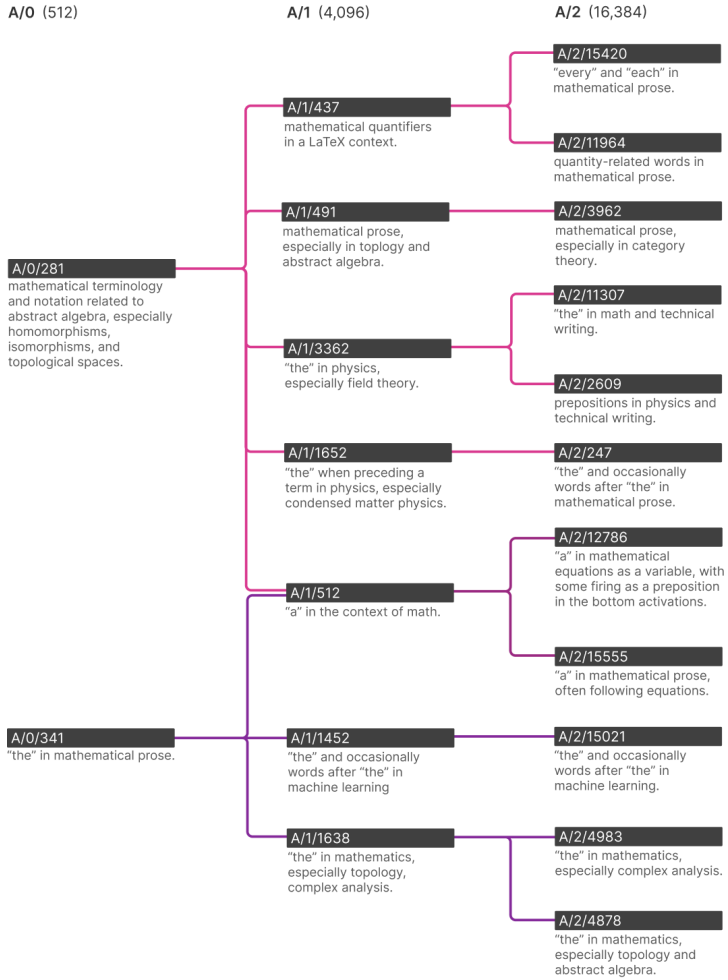
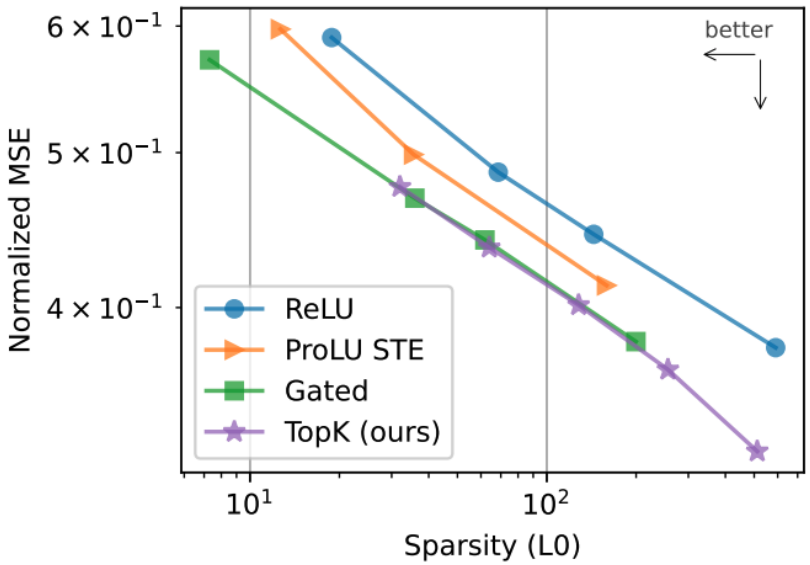


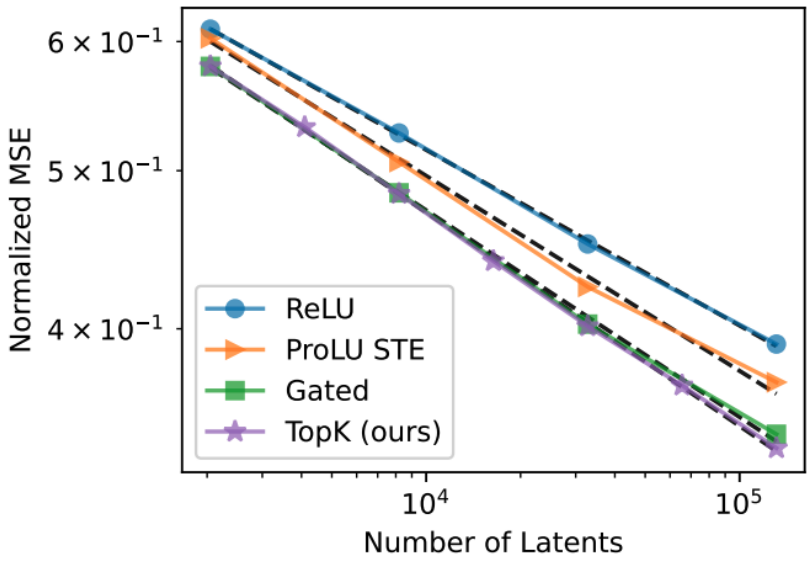

Figure: (Bricken, 2023)

# Architecture Updates

- SAE training balances reconstruction and sparsity.
- Solutions include Top-k activations and Gated SAEs.




(a) At a fixed number of latents ($n = 32768$), TopK has a better reconstruction-sparsity trade off than ReLU and ProLU, and is comparable to Gated.




(b) At a fixed sparsity level ($L_0 = 128$), scaling laws are steeper for TopK than ReLU.[6]

Figure: (Karvonen, 2024)

# Top-K activations

From the Open AI paper, applying a simple 11 year old idea.

"We use a k-sparse autoencoder [Makhzani and Frey, 2013], which directly controls the number of active latents by using an activation function (TopK) that only keeps the k largest latents, zeroing the rest. The encoder is thus defined as:

$$z = TopK(W_{enc}(x - b_{pre}))$$

and the decoder is unchanged. The training loss is simply

$$L = ||x - x^2||$$

"

Zero the lesser activated values

# Gated SAEs

The Deepmind paper introduces ideas from Gated Linear Unit (GRUs, 2017 and LSTMs, 1997)

$$\tilde{\mathbf{f}}(\mathbf{x}) := \underbrace{1\left[(\mathbf{W}_{\text{gate}}(\mathbf{x}-\mathbf{b}_{\text{dec}})+\mathbf{b}_{\text{gate}})>0\right]}_{\mathbf{f}_{\text{gate}}(\mathbf{x})} \odot \underbrace{\text{ReLU}(\mathbf{W}_{\text{mag}}(\mathbf{x}-\mathbf{b}_{\text{dec}})+\mathbf{b}_{\text{mag}})}_{\mathbf{f}_{\text{mag}}(\mathbf{x})}$$

$f_{gate}$ determines which features are active

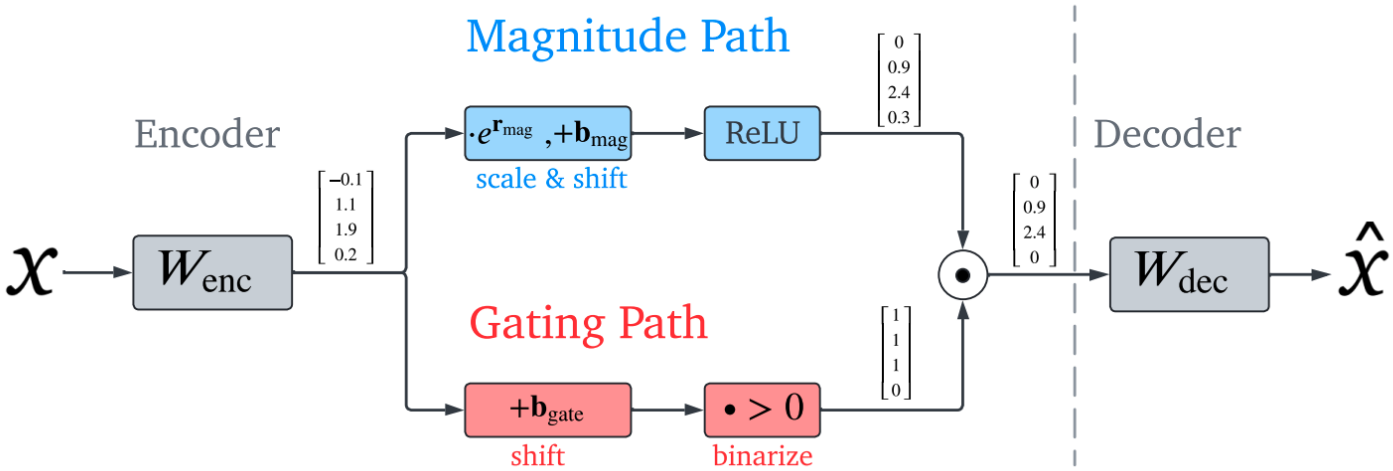



Figure: (Rajamanoharan, 2024)

# Open AI Paper Future Research Areas

- TopK forces every token to use exactly k latents, which is likely suboptimal. Ideally we would constrain $E[L0]$ rather than L0.
- A large fraction of the random activations of features we find, especially in GPT-4, are not yet adequately monosemantic. We believe that with improved techniques and greater scale17 this is potentially surmountable.
- A context length of 64 tokens is potentially too few tokens to exhibit the most interesting behaviors of GPT-4.

# From Claude's Bridge to Better LLMs: Leveraging SAE Insights for Explainability

- The Golden Gate Bridge example motivates the need for interpretability.
- SAE insights can help understand such preferences.
- Related work on monosemanticity: `https://transformer-circuits.pub/2024/scaling-monosemanticity/index.html`

# Another idea, what if we try to make layers specialized?

- Rather than guessing what $p(z|y,x)$ we can define $p(z)$ in a certain way, or train it to identify certain patterns.
- This idea emerged in the late 80s with papers by Jacobs and Hinton, including the 1991 paper "Adaptive mixture of local experts"
- "We compared standard backpropagation networks containing single hidden layer of 6 or 12 units with mixtures of 4 or 8 simple experts"

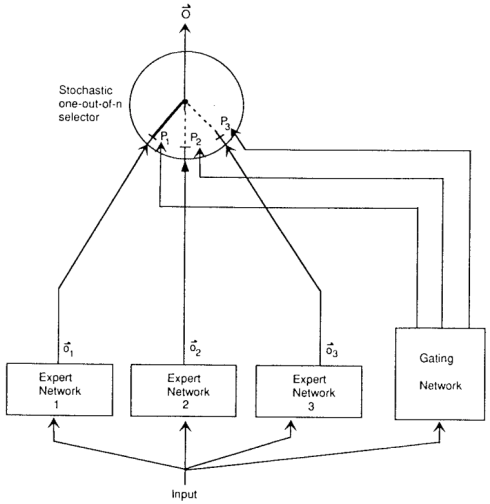



Figure: (Jacobs, 1991)

# MoEs through the years

- "In this work, we extend the Mixture of Experts to use a different gating network at each layer in a multilayer network, forming a Deep Mixture of Experts (DMoE)." (Eigan, 2013)
- "This article proposes a new mixture of SVMs that can be easily implemented in parallel and where each SVM is trained on a small subset of the whole data set." (Collobert, 2002)
- "A SVM ... Which is at least quadratic with respect to the number of examples. Hence, it is hopeless to try to solve real-life problems having more than a few hundred thousand examples with SVMs (Collobert, 2002)"
- "In practice, however, there are significant algorithmic and performance challenges. In this work, we address these challenges and finally realize the promise of conditional computation, achieving greater than 1000x improvements in model capacity with only minor losses in computational efficiency on modern GPU clusters.(Shazeer, 2017)"

# Concepts from THE SPARSELY-GATED MIXTURE-OF-EXPERTS LAYER

- "The MoE layer consists of a set of n "expert networks" $E_1, \ldots,$ $E_n$, and a "gating network" G whose output is a sparse n-dimensional vector."
- Define $y$ as output from a MoE layer where $y = \sum_{i=1}^{n}(G(x)_i E_i(x))$
- Use softmax gating function where $G_\sigma(x) = Softmax(x\dot{W}_g)$
- Add sparsity with Top-k and noise

$$G(x) = Softmax(KeepTopK(H(x), k))$$

$$H(x)_i = (x \cdot W_g)_i + StandardNormal() \cdot Softplus((x \cdot W_{noise})_i)$$

$$KeepTopK(v, k)_i = \begin{cases} v_i & \text{if } v_i \text{ is in the top } k \text{ elements of } v. \\ -\infty & \text{otherwise.} \end{cases}$$

# Visualizing a MoE

- "The MoE layer consists of a set of n "expert networks" $E_1, \ldots, E_n$, and a "gating network" G whose output is a sparse n-dimensional vector."
- Define $y$ as output from a MoE layer where $y = \sum_{i=1}^{n}(G(x)_i E_i(x))$
- Use softmax gating function where $G_\sigma(x) = Softmax(x\dot{W}_g)$
- Add sparsity with Top-k and noise

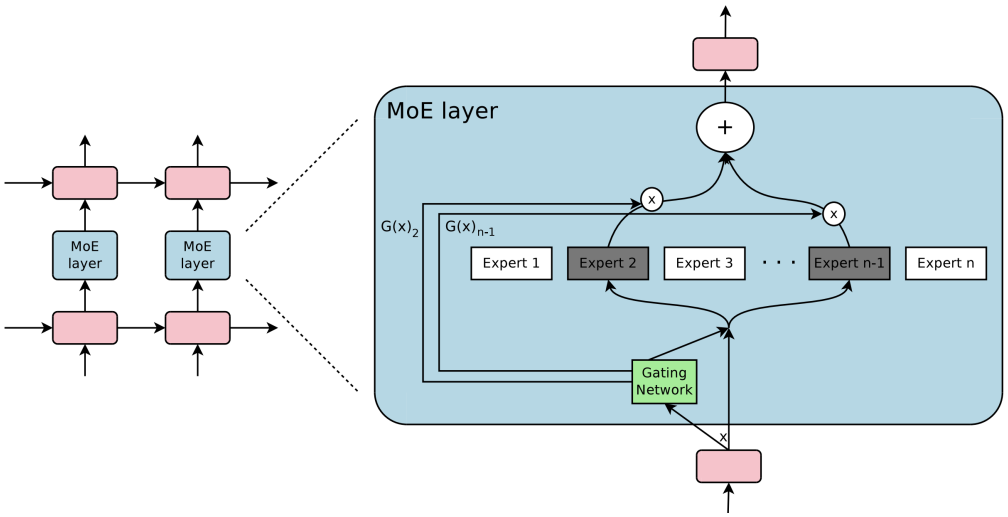



Figure: (Shazeer, 2017)

# Concepts from Switch Transformers (Fedus, 2021)

"Inspired by the success of model scale, but seeking greater computational efficiency, we instead propose a sparsely-activated expert model: the Switch Transformer. In our case the sparsity comes from activating a subset of the neural network weights for each incoming example."

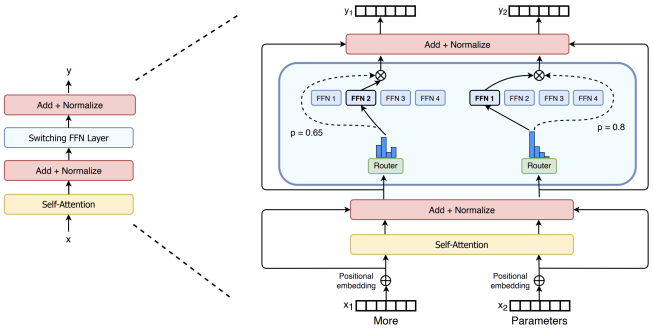



Figure: Switchformer Architecture

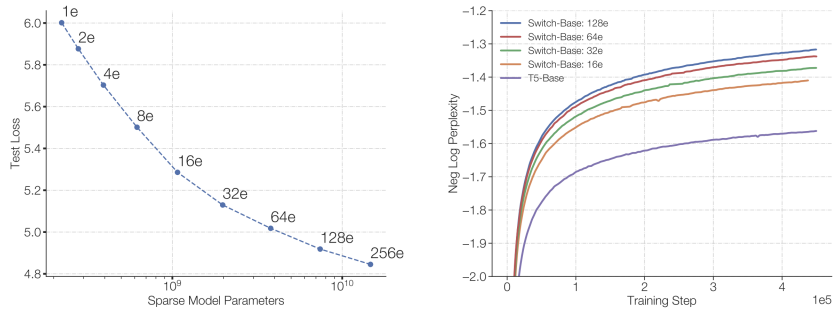



Figure: Performance on training sets

Key advancements in data processing, parralelization, regularization and scaling.

# What do experts learn?

One challenge is understanding what each expert learns, some work in Mixtral (Jiang, 2024) attempts to understand it

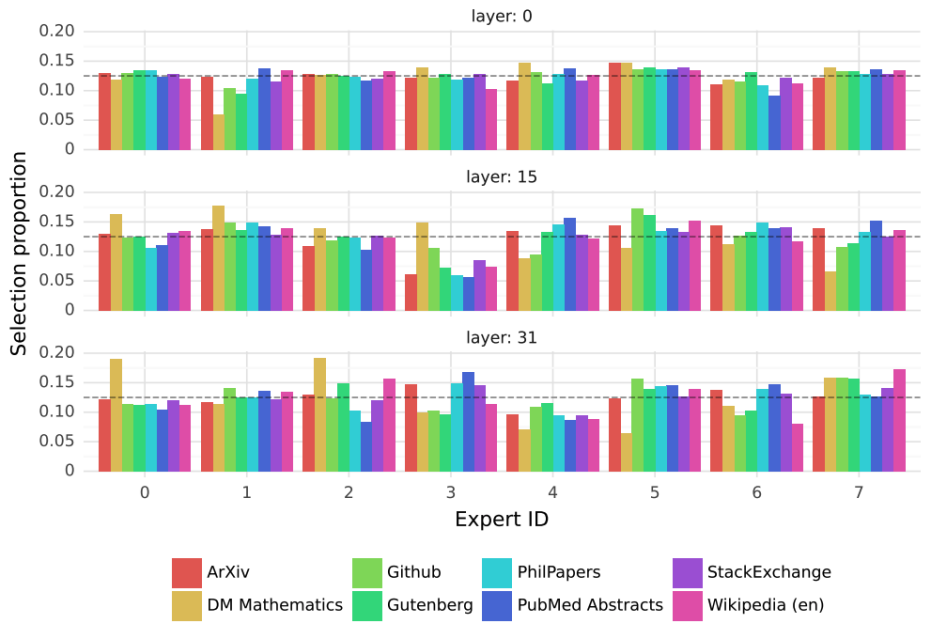



Figure: Mixtral Layer Activations

| Expert specialization | Expert position | Routed tokens |
|---|---|---|
| **Sentinel tokens** | Layer 1 | been <extra_id_4><extra_id_7>floral to<br><extra_id_10><extra_id_12><extra_id_15><br><extra_id_17><extra_id_18><extra_id_19>... |
| | Layer 4 | <extra_id_0><extra_id_1><extra_id_2><br><extra_id_4><extra_id_6><extra_id_7><br><extra_id_12><extra_id_13><extra_id_14>... |
| | Layer 6 | <extra_id_0><extra_id_4><extra_id_5><br><extra_id_6><extra_id_7><extra_id_14><br><extra_id_16><extra_id_17><extra_id_18>... |
| **Punctuation** | Layer 2 | , , , , , , , , , - , , , , , ). ) |
| | Layer 6 | , , , , , : . : , & , & & ? & - , , ? , , , . <extra_id_27> |
| **Conjunctions and articles** | Layer 3 | The the the the the the the the the The the the<br>the the the The the the the |
| | Layer 6 | a and and and and and and and or and a and .<br>the the if ? a designed does been is not |
| **Verbs** | Layer 1 | died falling identified fell closed left posted lost felt<br>left said read miss place struggling falling signed died<br>falling designed based disagree submitted develop |
| **Visual descriptions**<br>*color, spatial position* | Layer 0 | her over her know dark upper dark outer<br>center upper blue inner yellow raw mama<br>bright bright over open your dark blue |
| **Proper names** | Layer 1 | A Mart Gr Mart Kent Med Cor Tri Ca Mart<br>R Mart Lorraine Colin Ken Sam Ken Gr Angel A<br>Dou Now Ga GT Q Ga C Ko C Ko Ga G |
| **Counting and numbers**<br>*written and numerical forms* | Layer 1 | after 37 19. 6. 27 I I Seven 25 4, 54 I two dead we<br>Some 2012 who we few lower each |

Figure: Experts used

# Deep Seek MoE (Dai, 2024)

"Conventional MoE architectures like GShard, which activate the top-k out of N experts, face challenges in ensuring expert specialization, i.e. each expert acquires non-overlapping and focused knowledge. ... It involves two principal strategies: (1) finely segmenting the experts into $mN$ ones and activating $mK$ from them, allowing for a more flexible combination of activated experts; (2) isolating $K_s$ experts as shared ones, aiming at capturing common knowledge and mitigating redundancy in rou

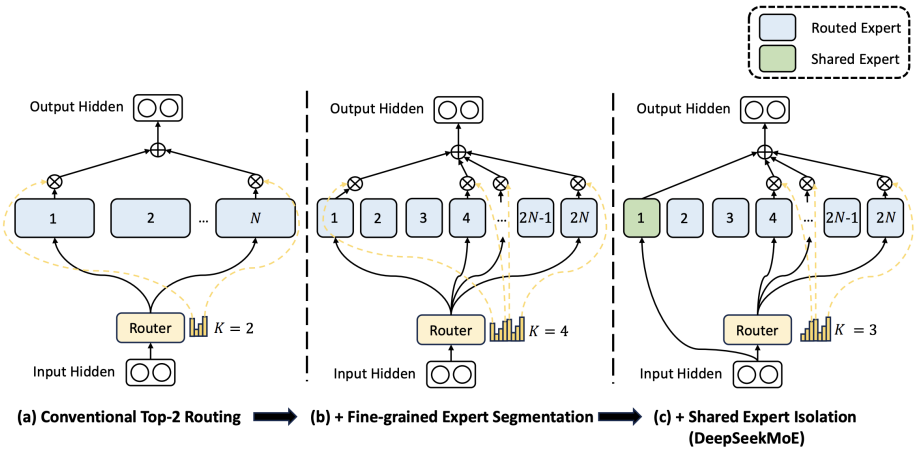
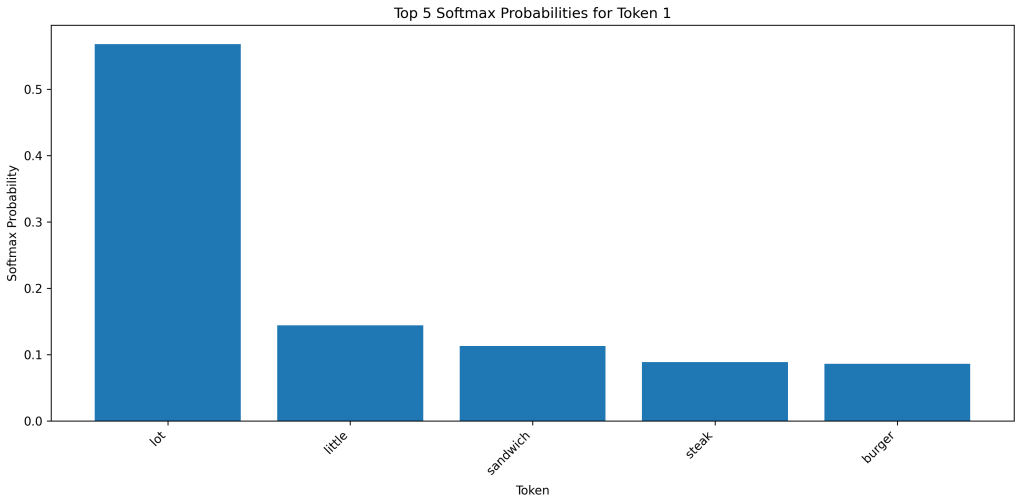



Figure: DeepSeekMoE architecture

# LLMs for more deterministic tasks

- You are using LLMs to do a classification tasks
- But you want the set to be finite
- You are using the output in another program so it could should be an enum
- Could train a linear head on the model
- Or sample more efficiently!
- First let's discuss how LLMs sample

# LLM sampling - Choosing the next token

- At the final layer of LLM we have a linear layer which maps our hidden layer outputs into our vocabulary size.
- We then apply softmax to each to get the results as a probability
- For example our input $x$ might be "I want to eat a", assume each word is a single token
- $x =$ ["I", "want", "to", "eat","a"] $= [40, 765, 284, 4483, 257]$
- We then see 

bability

Figure: GPT 2 Softmax

# Temperature

Sometimes we don't want to use regular softmax, we can use a temperature parameter to adjust the distribution

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

A simple way to get out of local minima is to occasionally allow ju
to configurations of higher energy. An algorithm with this property wa
troduced by Metropolis, Rosenbluth, Rosenbluth, Teller, & Teller (195
study average properties of thermodynamic systems (Binder, 1978) and
recently been applied to problems of constraint satisfaction (Kirkpatr
Gelatt, & Vecchi, 1983). We adopt a form of the Metroplis algorithm th
suitable for parallel computation: If the energy gap between the *on* and
states of the $k^{th}$ unit is $\Delta E_k$ then regardless of the previous state set $s_k = 1$
probability

$$p_k = \frac{1}{(1 + e^{-\Delta E_k / T})}$$

where $T$ is a parameter that acts like temperature (see Figure 1).

Figure: Temperature in NN (Ackley,1985)

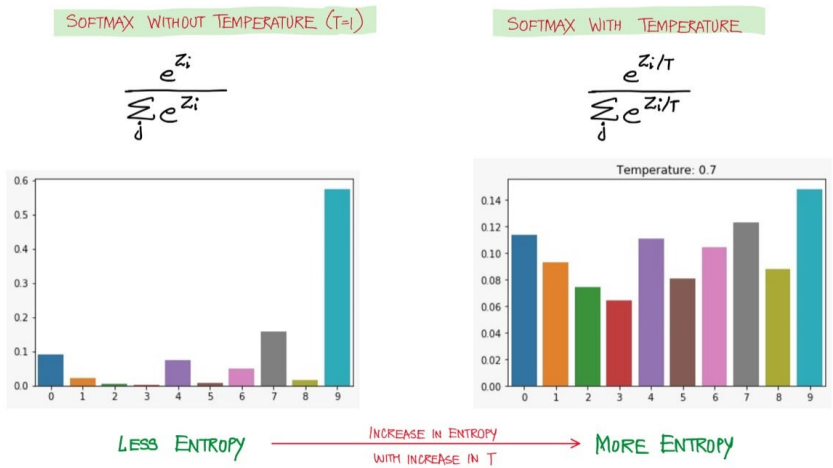



Figure: Temperature impact on Softmax (Sharma,2022)

# Top k sampling

- Now when sampling we often want to constrain the breadth of tokens we sample from
- We can introduce a parameter called top-k
- Top-k only takes the $k$ most likely members of the vocabulary when sampling
- In our previous example if we set $k = 3$ we only sample the top 3 values

# Top p sampling

- Similarly to using k to limit the vocab size, we can use a Top-p technique to limit based on cumulative probability.
- Top-p only takes the values that sum up to $p$
- In our previous example if we set $p = 0.5$ we only sample up to cumulitive sum of 0.5

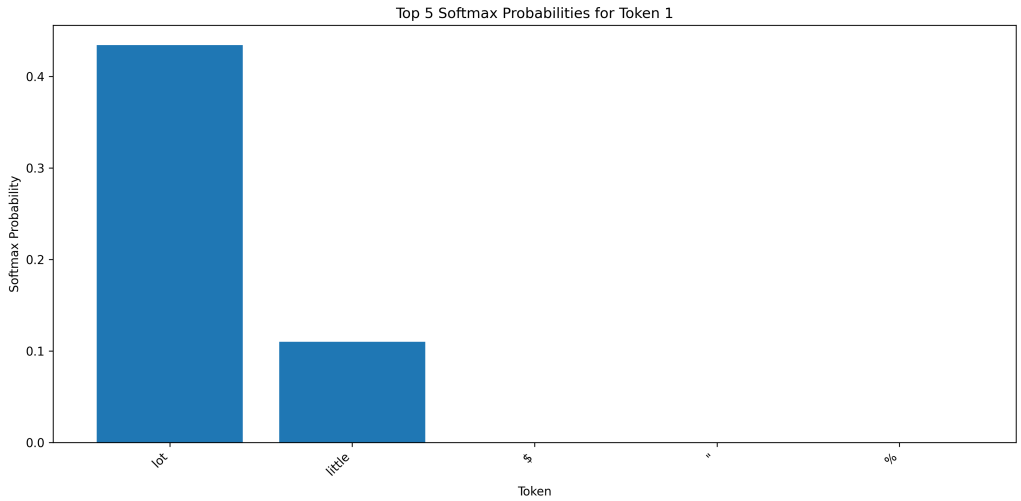



Figure: GPT 2 top-p=0.5

# Beam Search

- In our previous examples we used a greedy decoding, with just one token. We can also decode a few tokens ahead and check for the most likely n-gram.

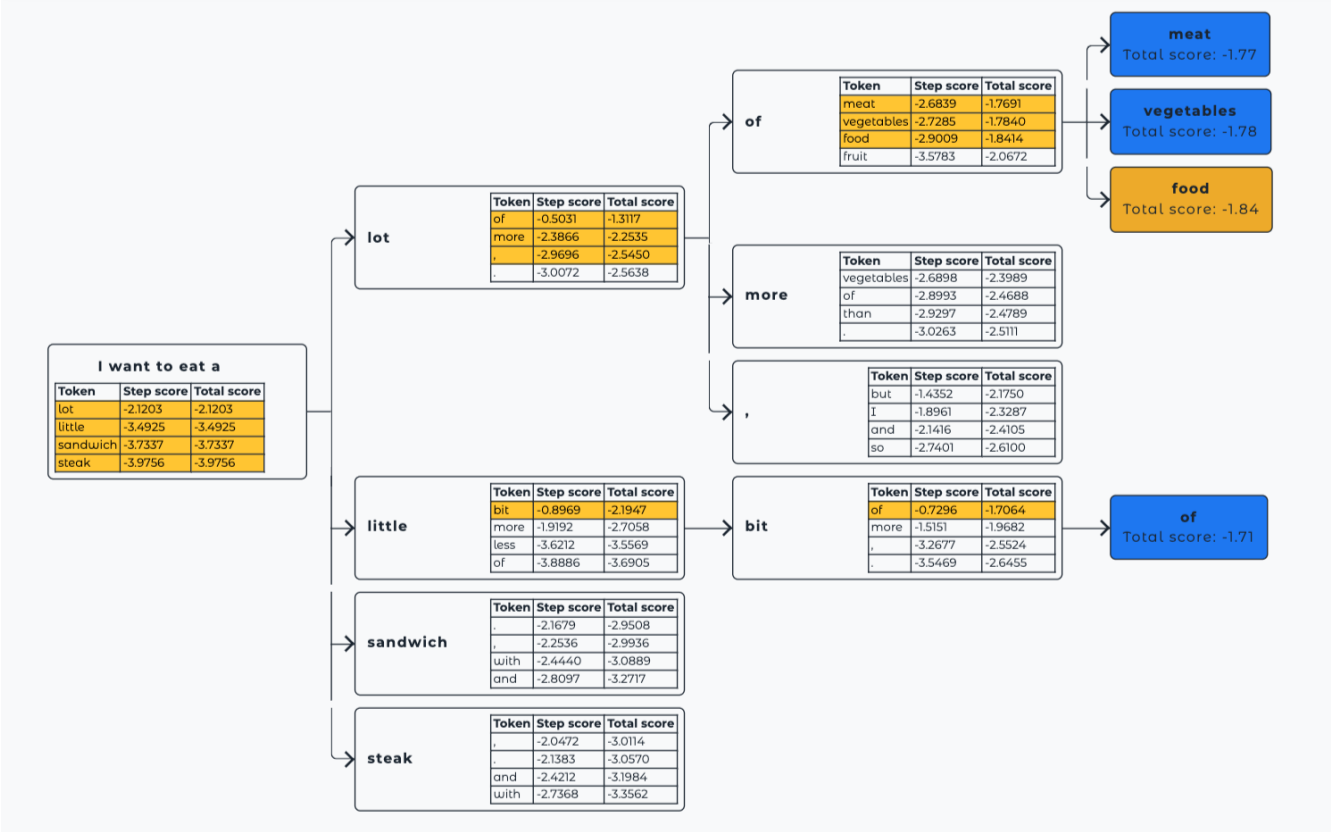



Figure: Beamsearch visual using (M-ric, 2024)

# Greedy vs Beam Search formulation

When doing a greedy search we want to maximize each token individually.

$$y_t = \underset{y}{\text{argmax}}\, P(y_t|y_1, \ldots, y_{t-1}, \theta)$$

Which can be related to the joint probability as:

$$P(Y|\theta) = \prod_{t=1}^{T} P(y_t|y_{<t}, \theta)$$

For beam search we maximize across a beam length + number of beams and maximize across each beam length.

$$y_t = \underset{y,beam}{\text{argmax}}\, P(y|y_{beam_1}, \ldots, y_{beam_t-1}, \theta)$$

# Beam Search

- In our previous examples we used a greedy decoding, with just one token. We can also decode a few tokens ahead and check for the most likely n-gram.

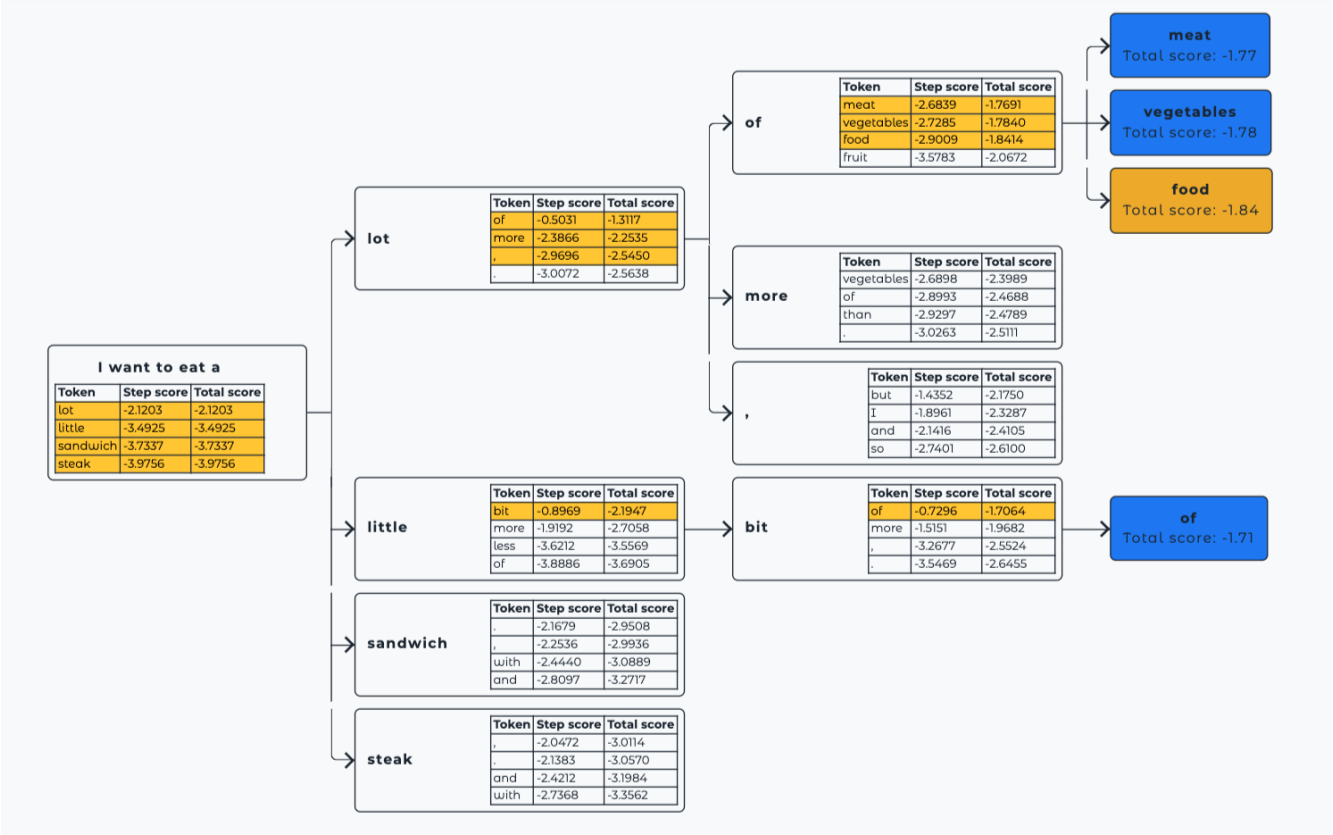



Figure: Beamsearch visual using (M-ric, 2024)

# Back to our goal, limit LLM outputs

- So how do we sample from our given enum values
- We only want to output strawberry or blueberry
- What if we only allow outputs with a given token?

```python
from enum import Enum
from transformers import pipeline

class Fruit(Enum):
    STRAWBERRY = "strawberry"
    BLUEBERRY = "blueberry"
def predict_fruit(prompt):
    """Predicts a fruit using a language model."""
    generator = pipeline('text-generation', model='gpt2')
    full_prompt = f"{prompt} "
    generated_text = generator(full_prompt, max_length=20, num_return_sequences=2)
```

# Limiting our tokens

- What if we generate our logits, but only allow our set of tokens?
- Strawberry is represented by [301, 1831, 8396]
- Blueberry is represented by [17585, 8396]

$$L_i' = \begin{cases} l_i & \text{if } i \in \{301, 1831, 8396\} \text{ (part of "strawberry")} \\ l_i & \text{if } i \in \{17585, 8396\} \text{ (part of "blueberry")} \\ -\infty & \text{otherwise} \end{cases}$$

- What if our model just outputs "I want to eat blue" or "I want to eat straw"?

# A world of grammers

- Let's look at another example, generating valid JSON
- We have a set of rules in the JSON spec
- The total set of tokens in the vocabulary is unconstrained

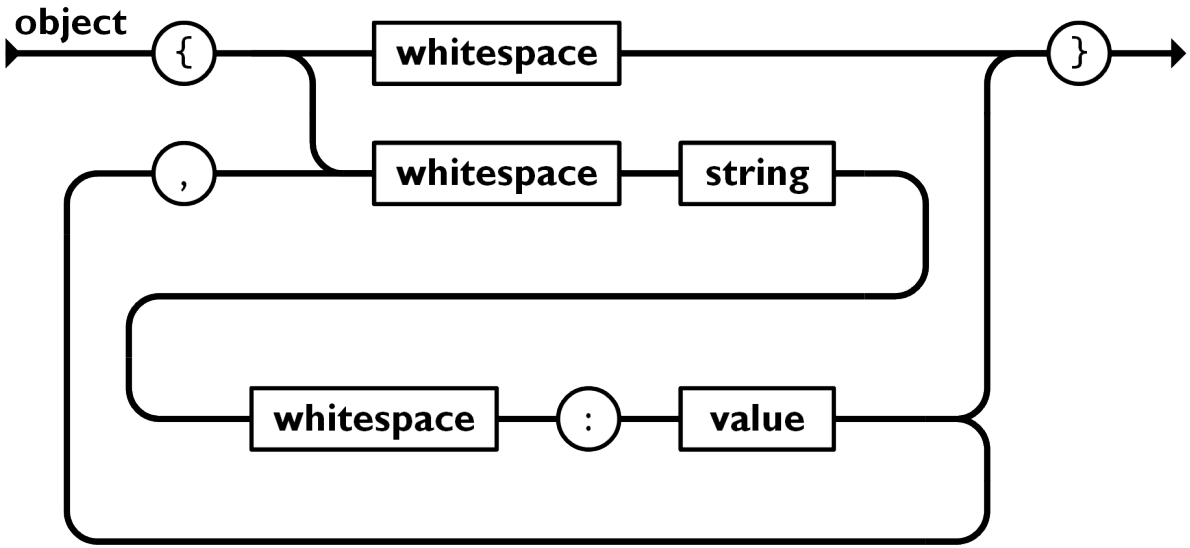



Figure: State diagram of the 'object' spec in JSON

- Uh oh, now what?

# What if we ask the model nicely

- "Only output blueberry or strawberry"
- Things you might see in a prompt, doesn't always workings
- "Sure I can help you only output blueberry or strawberry"

# Constrained decoding

- Limiting model output is called constrained decoding
- Can be applied at generation or sampling stage
- Sampling stage, limit the tokens we sample

$$p(y_i) = \begin{cases} p(y_i|y_1, \ldots, y_{i-1}, \theta) & \text{if } i \in \{C\} \text{ Constrained set of tokens} \\ 0 & \text{otherwise} \end{cases}$$

- Generation stage, add additional tokens $\mathbf{P}$ to help with generation

$$p(y_i) = p(y_i|y_1, \ldots, y_{i-1}, \theta, \mathbf{P})$$

- Which method affects accuracy?

# Constrained decoding

**Algorithm 1** Constrained Decoding

**Input:** Checker $C$, LLM $f$, Tokenized Prompt $x$
**Output:** Completion $o$ adhering to $C$

$o \leftarrow []$
$C$.`init`()
**loop**
  $C$.`update`($o$) // advance state of $C$
  $\boldsymbol{m} \leftarrow C$.`mask`() // compute mask
  $\boldsymbol{v} \leftarrow f(x + o)$ // compute logits
  $\boldsymbol{v}' \leftarrow \boldsymbol{m} \odot \boldsymbol{v}'$
  $t \leftarrow$ `decode`($\alpha'$) // e.g., argmax or sample
  **if** $t = EOS$ **then break**
  $o$.`append`($t$)
**end loop**
**return** $o$ // optionally detokenize

Figure: Pseudocode for Constrained decoding(Beurer-Kellner, 2024)

# Challenges of Constrained Decoding

- Minimally invasive methods
- Supports different techniques like Regex, CFGs, Templates
- Low inference overhead
- Paper from 2024 introduces DOMINO to handle these challenges (Beurer-Kellner, 2024)

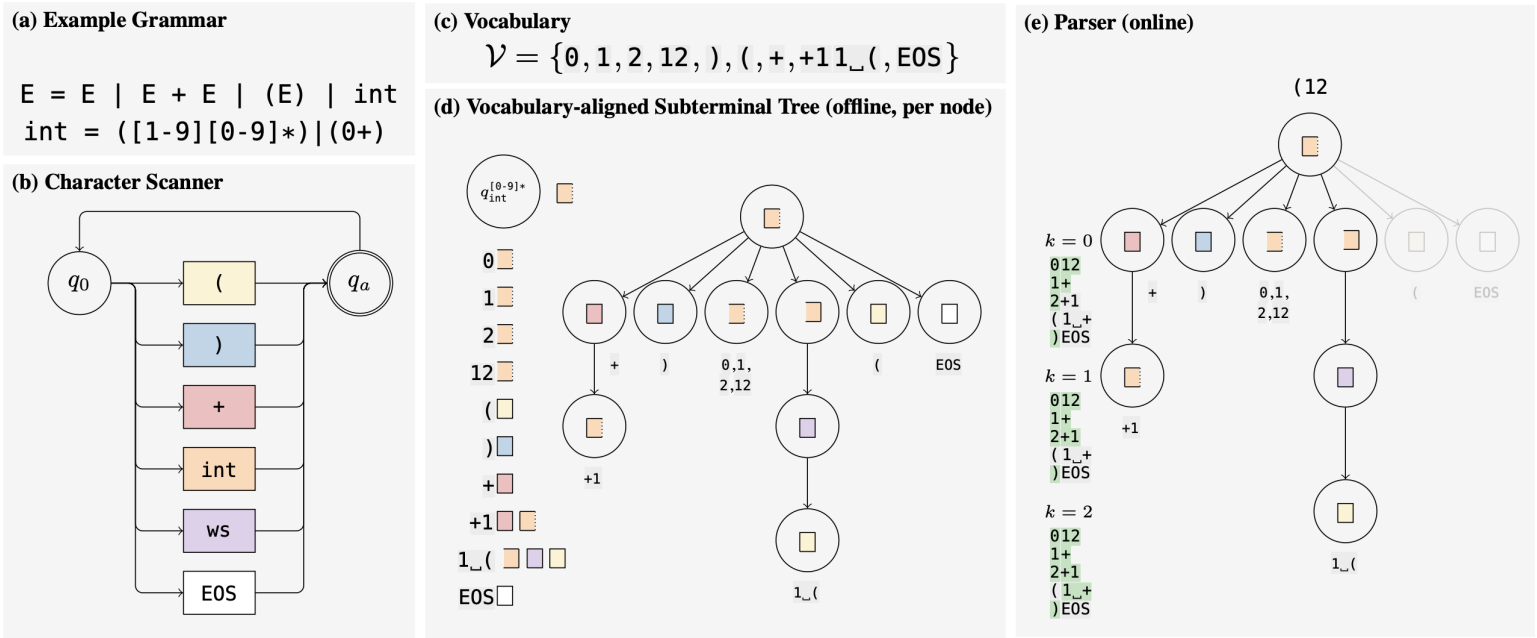



Figure: Method for DOMINO (Beurer-Kellner, 2024)

# Accuracy and Inference overhead

| Dataset | Model | Method | Accuracy | Well-Formed | Perplexity | Performance Impact |
|---|---|---|---|---|---|---|
| GSM8K | *Mistral 7B* | Unconstrained | 0.415 | 0.952 | 1.636 | 1.0× |
| | | GUIDANCE Lundberg et al. | 0.345 | 0.960 | 1.624 | 0.98× |
| | | GUIDANCE $^{WS}$ Lundberg et al. | 0.403 | 0.976 | 1.737 | 0.54× |
| | | `llama.cpp` Gerganov & et. al. | 0.375 | 0.973 | 1.751 | 0.80× |
| | | DOMINO ($k = \infty$) | **0.418** | 0.968 | 1.739 | **1.77**× |
| | *Llama-2 13B* | Unconstrained | **0.262** | 0.904 | 1.650 | 1.0× |
| | | GUIDANCE Lundberg et al. | 0.152 | 0.947 | 1.659 | 1.12× |
| | | GUIDANCE $^{WS}$ Lundberg et al. | 0.259 | 0.977 | 1.760 | 0.73× |
| | | `llama.cpp` Gerganov & et. al. | 0.237 | 0.978 | 1.780 | 0.86× |
| | | DOMINO ($k = \infty$) | **0.262** | 0.920 | 1.750 | **1.66**× |
| CoNLL2003 | *Mistral 7B* | Unconstrained | **0.21** | 0.988 | 1.573 | 1.0× |
| | | GUIDANCE Lundberg et al. | 0.098 | 0.998 | 1.780 | 2.02× |
| | | GUIDANCE $^{WS}$ Lundberg et al. | 0.19 | 0.998 | 1.896 | 0.82× |
| | | `llama.cpp` Gerganov & et. al. | 0.117 | 0.995 | 1.560 | 0.80× |
| | | DOMINO ($k = \infty$) | **0.21** | 0.988 | 1.902 | **2.66**× |
| | *Llama-2 13B* | Unconstrained | **0.09** | 0.897 | 1.579 | 1.0× |
| | | GUIDANCE Lundberg et al. | 0.062 | 1.000 | 1.820 | 2.18× |
| | | GUIDANCE $^{WS}$ Lundberg et al. | 0.087 | 0.980 | 1.767 | 0.90× |
| | | `llama.cpp` Gerganov & et. al. | 0.080 | 0.922 | 1.786 | 0.86× |
| | | DOMINO ($k = \infty$) | **0.09** | 0.897 | 1.812 | **2.71**× |

Figure: Method for DOMINO (Beurer-Kellner, 2024)

DOMINO uses a technique called speculative decoding, something we'll cover next lecture!

# Summary

- Sparse Autoencoders are useful for finding features in models
- MoE attempt to have parts of the model learn specific behaviors
- Constrained decoding let's us introduce smarter token sampling when generating sequences