Language model inference: theory and algorithms

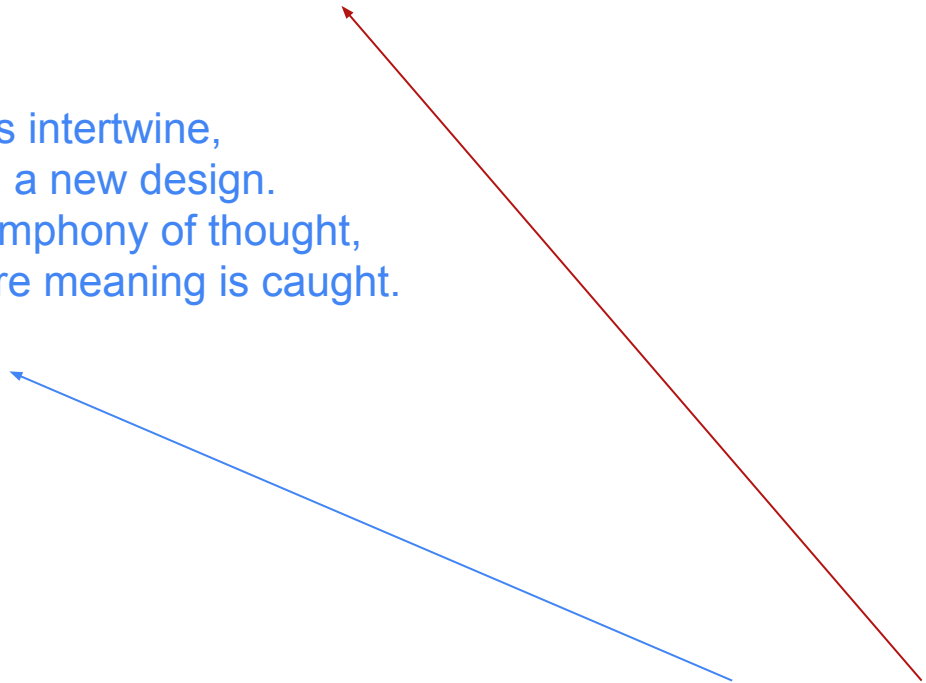
ISIT 2024

Ahmad Beirami, Ananda Theertha Suresh
Google

Generative models

Write a poem about information theory and large language models. Limit it to 4 sentences.

In depths of data, patterns intertwine,
A language model learns, a new design.
From bits and bytes, a symphony of thought,
Information's dance, where meaning is caught.



Generative model a tool to sample from Prob (output | input)

Agenda

Background on language models

- Non-transformer models
- Transformer models
- Inference from language models

Aligned (user-preferred) language model inference

- Best-of-N
- KL-regulartized RL
- Controlled decoding

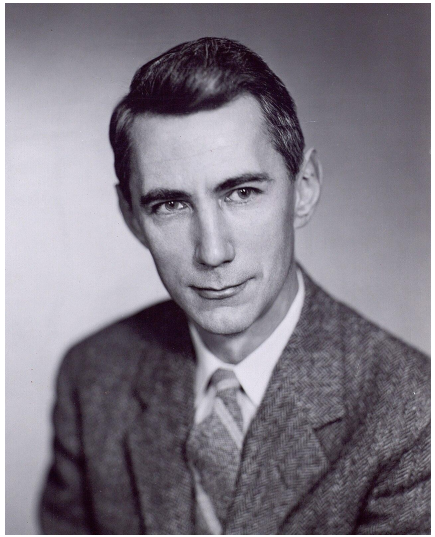
Efficient language model inference

- Compression
- Speculative decoding

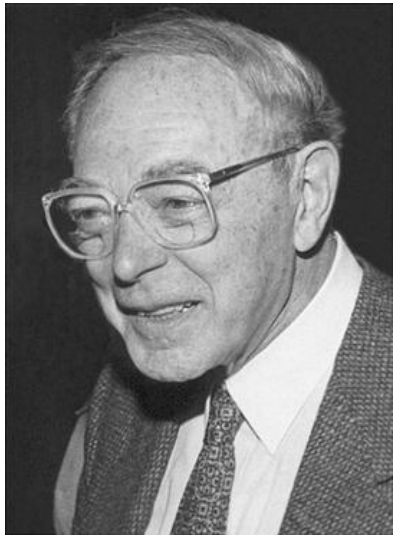
Background on language models

Phase 1: pre-1970s

Andrey Markov



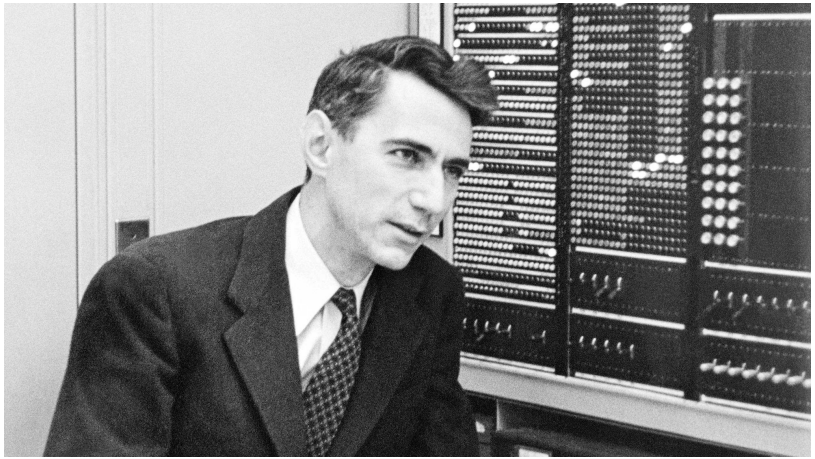
Claude Shannon



Zellig Harris

Key takeaway: languages can be represented as a probabilistic process

Language models as probabilistic processes



The Bell System Technical Journal

Vol. XXVII *July, 1948* *No. 3*

A Mathematical Theory of Communication

By C. E. SHANNON

2. The Discrete Source of Information

Language models as probabilistic processes

1. Zero-order approximation (symbols independent and equiprobable).

   XFOML RXKHRJFFJUJ ZLPWCFWKCYJ FFJEYVKCQSGHYD QPAAMKBZAACIBZL-HJQD.

2. First-order approximation (symbols independent but with frequencies of English text).

   OCRO HLI RGWR NMIELWIS EU LL NBNESEBYA TH EEI ALHENHTTPA OOBTTVA NAH BRL.

3. Second-order approximation (digram structure as in English).

   ON IE ANTSOUTINYS ARE T INCTORE ST BE S DEAMY ACHIN D ILONASIVE TU-COOWE AT TEASONARE FUSO TIZIN ANDY TOBE SEACE CTISBE.

4. Third-order approximation (trigram structure as in English).

   IN NO IST LAT WHEY CRATICT FROURE BIRS GROCID PONDENOME OF DEMONS-TURES OF THE REPTAGIN IS REGOACTIONA OF CRE.

5. First-order word approximation. Rather than continue with tetragram, . . . , *n*-gram structure it is easier and better to jump at this point to word units. Here words are chosen independently but with their appropriate frequencies.

   REPRESENTING AND SPEEDILY IS AN GOOD APT OR COME CAN DIFFERENT NAT-URAL HERE HE THE A IN CAME THE TO OF TO EXPERT GRAY COME TO FURNISHES THE LINE MESSAGE HAD BE THESE.

6. Second-order word approximation. The word transition probabilities are correct but no further struc-ture is included.

   THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH WRITER THAT THE CHAR-ACTER OF THIS POINT IS THEREFORE ANOTHER METHOD FOR THE LETTERS THAT THE TIME OF WHO EVER TOLD THE PROBLEM FOR AN UNEXPECTED.

Each token is a character

"It appears then that a sufficiently complex stochastic process will give a satisfactory representation of a discrete source."

Each token is a word

Tokens for language models

Possible choices

- Character
- Word ← this talk (for simplicity)
- Word-pieces ← most widely used currently (motivated by minimum description principle)

Smaller units	Larger units
Small memory footprint (embedding table)	Large memory footprint (embedding table)
Harder to model (long Markov chain)	Easier to model (short Markov chain)

Toward a Theory of Tokenization in LLMs, Rajaraman et al., 2024

Probabilistic sources

- $\mathcal{X}$: set of all tokens {I, food, coffee, eat, drink, tea}
- Terminal token $
- $\mathcal{X}^*$: set of all sequences that have one $ at the end and nowhere else.
  - {I eat food $, I drink coffee $, I drink tea $, …}
- A language model is a probability distribution over $\mathcal{X}^*$

p(I eat food $) + p(I drink coffee $) + p(I drink tea $) + … = 1

Phase 1 - Phase 2

Debate: are language models probabilistic or deterministic grammars?

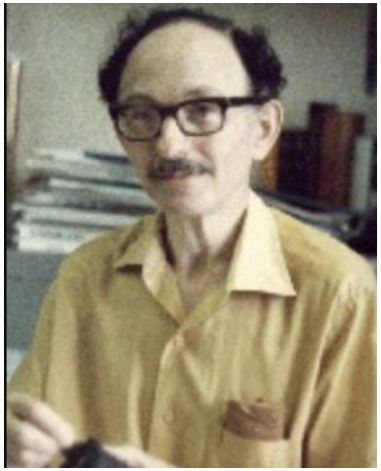
Lack of applications and computing

Phase 2: 1970s-2000s, non-parametric models

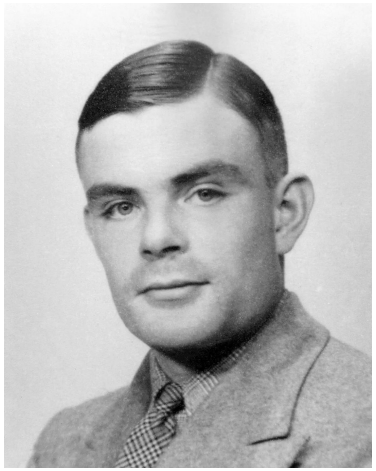
- Motivated by applications in speech recognition and machine translation
- Limited compute (non-neural models)
- Limited data: number of sequences is exponential in length
  - Cannot model distributions over sequences directly
  - Model distributions over tokens and use chain rule
  - Generalize to unseen sequences

From statistics

Pierre-Simon Laplace



I.J. Good



Alan Turing

Tools from statistics: add-constant estimators

- $\mathcal{X}$:set of all tokens {I, food, coffee, eat, drink, tea, $}
- Dataset: {I drink coffee $}
- count(I) = 1, count(coffee) = 1, count(drink) = 1, count($) = 1, count(rest) = 0
- Add-constant estimator (Laplace estimator (c = 1))

$$p(x) = \frac{\text{count}(x) + c}{n + vc}$$

- p(I) = 2/11, p(food) = 1/11, p(coffee) = 2/11, p(eat) = 1/11, p(drink) = 2/11, p(tea) = 1/11, p($) = 2/11 (all tokens get non-zero probability)

On learning distributions from samples, Kamath et al., 2015

k-gram models

$$p(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} p\left(x_i \,|\, x_1, \ldots, x_{i-1}\right)$$
$$\approx \prod_{i=1}^{n} p\left(x_i \,|\, x_{i-k+1}, \ldots, x_{i-1}\right)$$

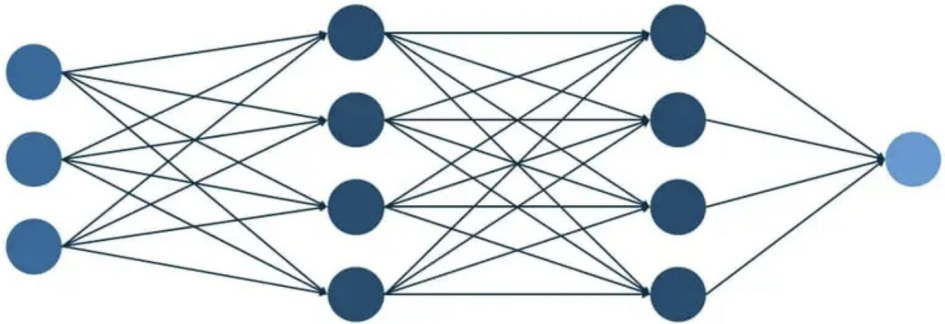
- Estimate $p\left(x_i \,|\, x_{i-k+1}, \ldots, x_{i-1}\right)$ using statistical tools
- No neural networks
- Limited data is sufficient for small values of k: # of k-tuples = V^k

Extending it to very large values of k is nearly-impossible ($k \lesssim 5$)

Phase 3: 2000-current, parametric models

$$p(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} p\left(x_i \mid x_1, \ldots, x_{i-1}\right)$$

$$\approx \prod_{i=1}^{n} p_\theta\left(x_i \mid x_1, \ldots, x_{i-1}\right)$$



Efficient training methods for maximum entropy language modeling, Wu and Khudanpur, 2000

Learning parametric models: maximum likelihood approach

Given a dataset D with examples $x^*(1), x^*(2), \ldots, x^*(m)$

$$\theta^* = \arg\max_\theta \prod_{i=1}^{m} p_\theta(x^*(i))$$

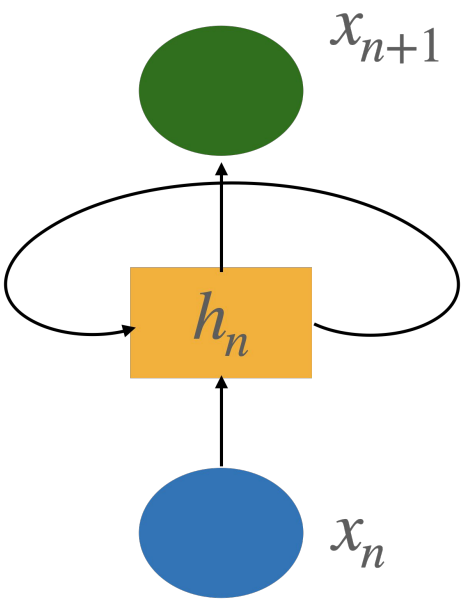
$$= \arg\min_\theta - \sum_{i=1}^{m} \log p_\theta(x^*(i))$$

$$= \arg\min_\theta - \sum_{i=1}^{m} \sum_{j=1}^{n_i} \log p_\theta(x_j \mid x_1, \ldots, x_{j-1})$$

N-gram models: generalization via statistical tools (add-constant)

Parametric models: generalization via explicit regularizer / model capacity / early stopping

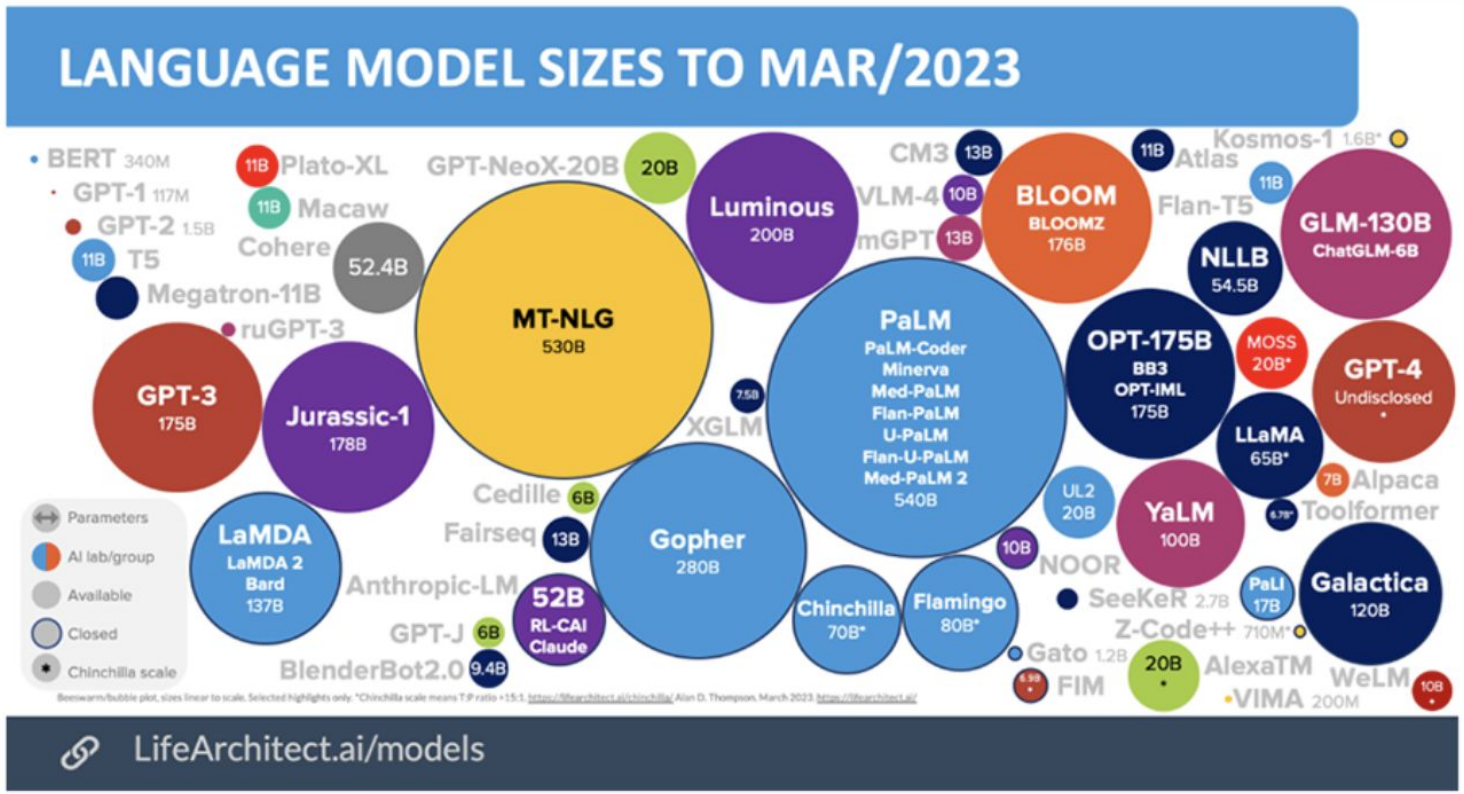
Recurrent models



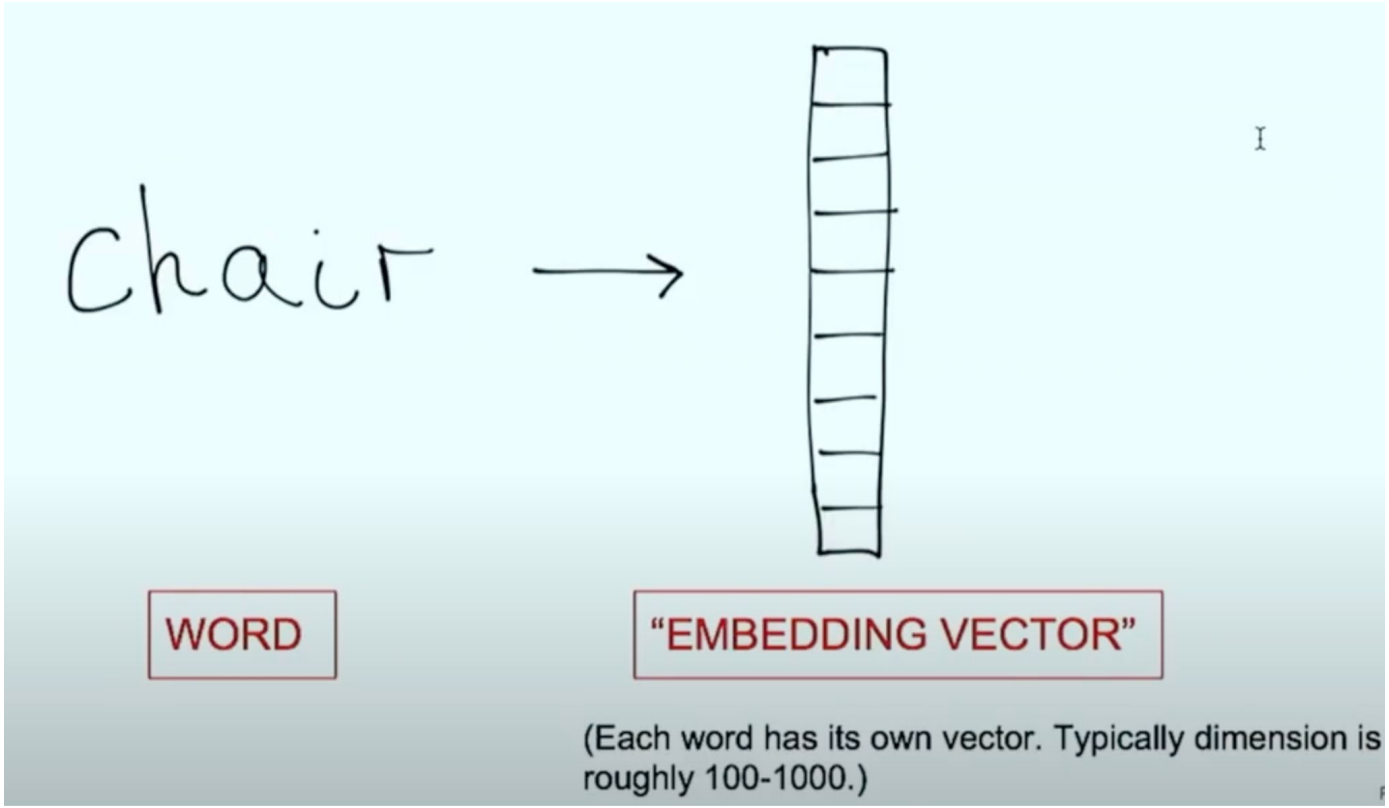
$$p(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} p\left(x_i \mid x_1, \ldots, x_{i-1}\right)$$
$$\approx \prod_{i=1}^{n} p_\theta\left(x_i \mid x_1, \ldots, x_{i-1}\right)$$
$$\approx \prod_{i=1}^{n} p_\theta\left(x_i \mid h_{i-1}\right)$$

- Maintains a hidden state
- Computationally-inefficient to train
- Maximum (k $\lesssim$ 100)

Phase 3.5: 2017-current, very large parametric models

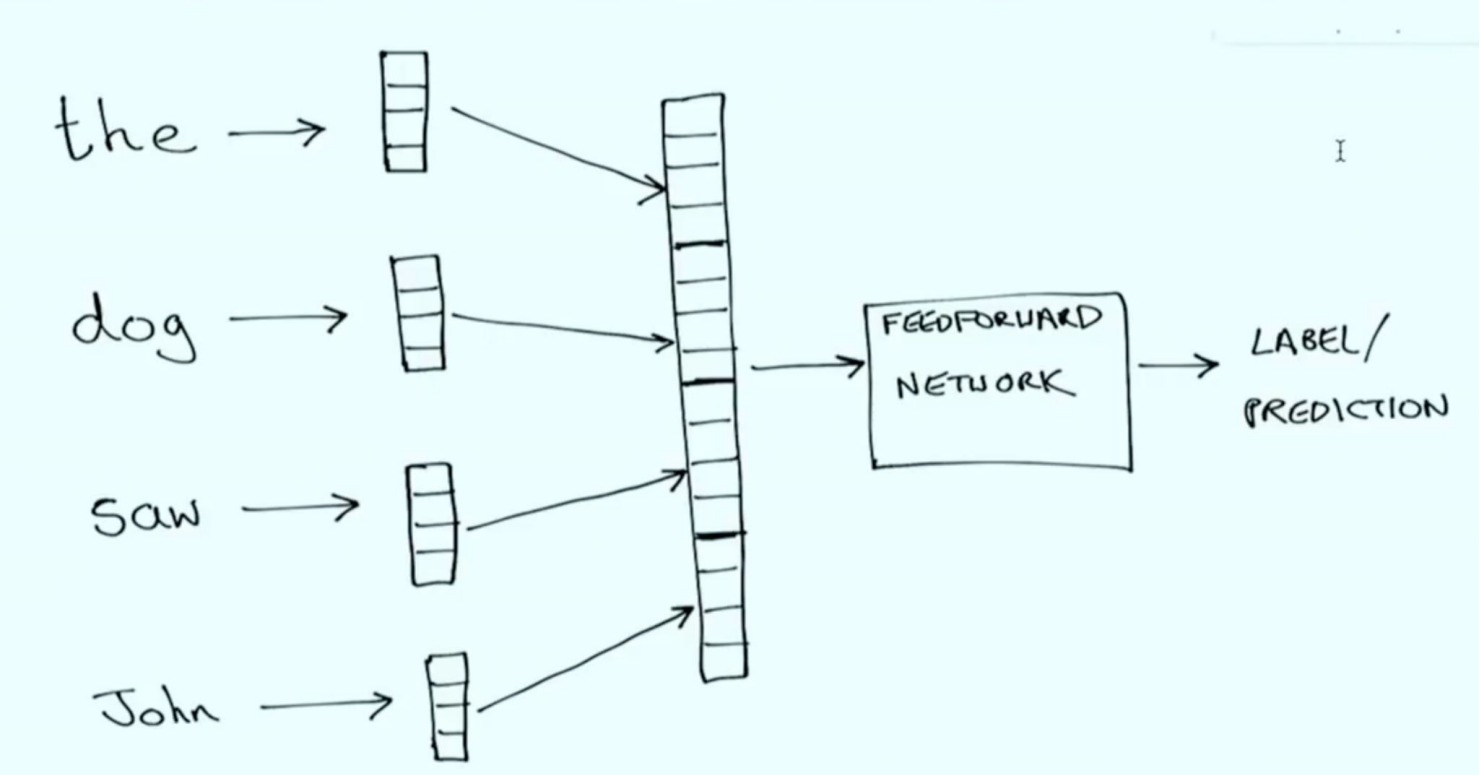


Word embeddings



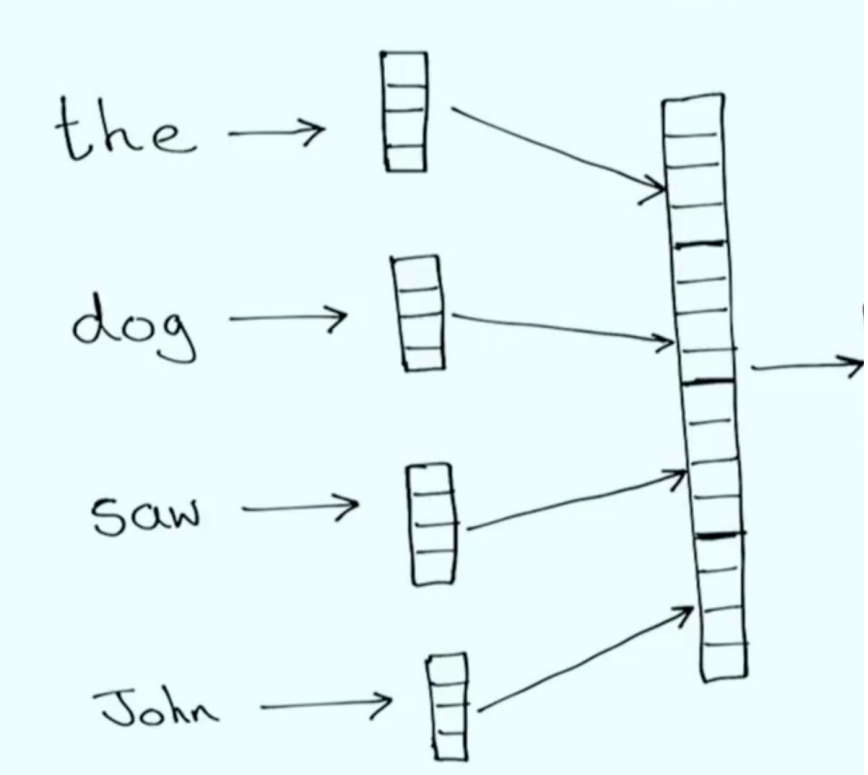
Graphic credit: Michael Collins, Google DeepMind

Predictions from word tuples



Graphic credit: Michael Collins, Google DeepMind

Transformer models

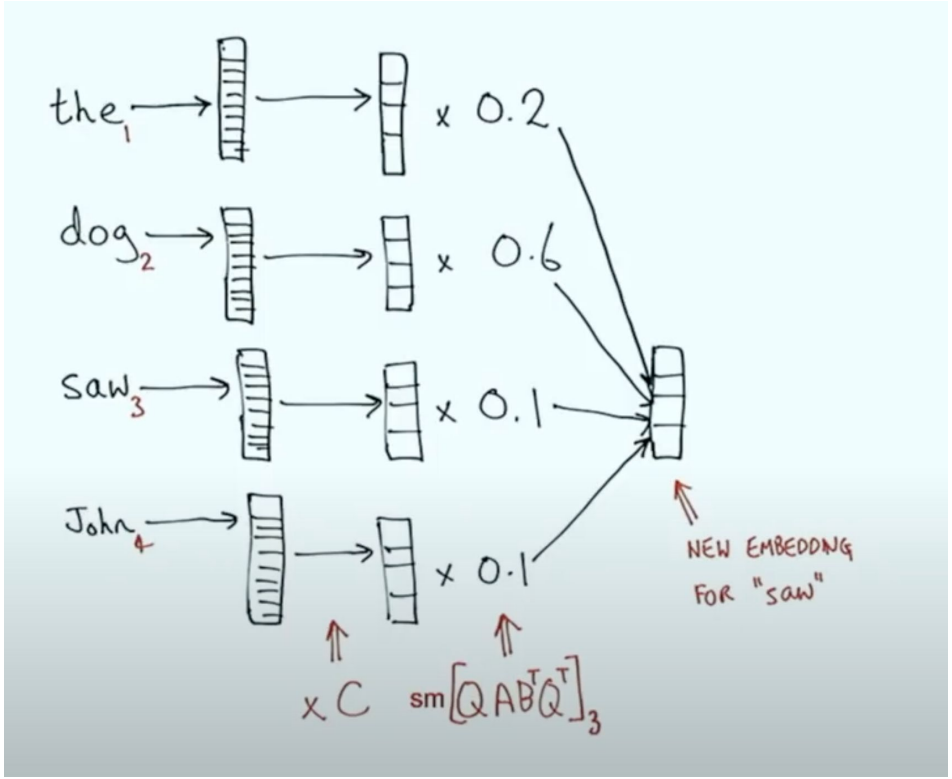


Goal:
map the sequence of embeddings to another sequence of the same length with improved representations incorporating the context

Graphic credit: Michael Collins, Google DeepMind

Attention is all you need, Vaswani et al., 2017

Transformer models



Graphic credit: Michael Collins, Google DeepMind

Attention is all you need, Vaswani et al., 2017

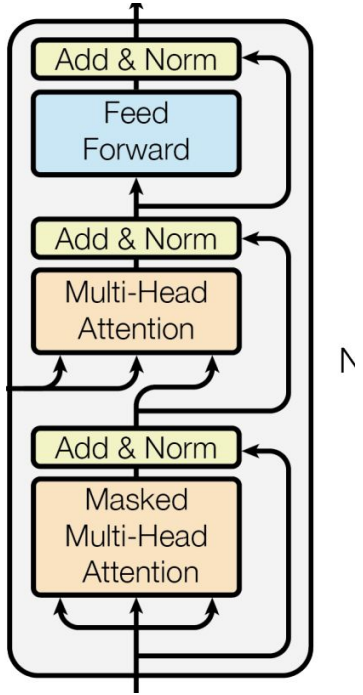
Transformer models

Transformer layer

- Feedforward layers
- Layernorm
- Attention mechanism

Transformer models

- Repeated stack of Transformer layers
- k ≅ thousands



Attention is all you need, Vaswani et al., 2017

Some theoretical observations on Transformer models

Can Transformer models represent any probability distribution over sequences?

- Yes. Yun et al., 2019, Are Transformers universal approximators of sequence-to-sequence functions?

Can Transformer models represent any function of unbounded sequence length?

- With additional memory, yes. Schurmanns 2023, Memory Augmented Large Language Models are Computationally Universal

How to use language models?

Write a poem about information theory and large language models. Limit it to 4 sentences.

In depths of data, patterns intertwine,
A language model learns, a new design.
From bits and bytes, a symphony of thought,
Information's dance, where meaning is caught.

Language models provide:

Prob (output | input)

Transformer models are next token predictors

Given $x_1, x_2, \ldots, x_n$, transformer model computes

$$p(x_{n+1} \mid x_1, x_2, \ldots, x_n)$$

Language model inference

Technique 1: most-likely output

Return the most likely output $\arg\max_{x^*} p(x^* \mid \text{input})$

Useful when the output is expected to a deterministic function of input.

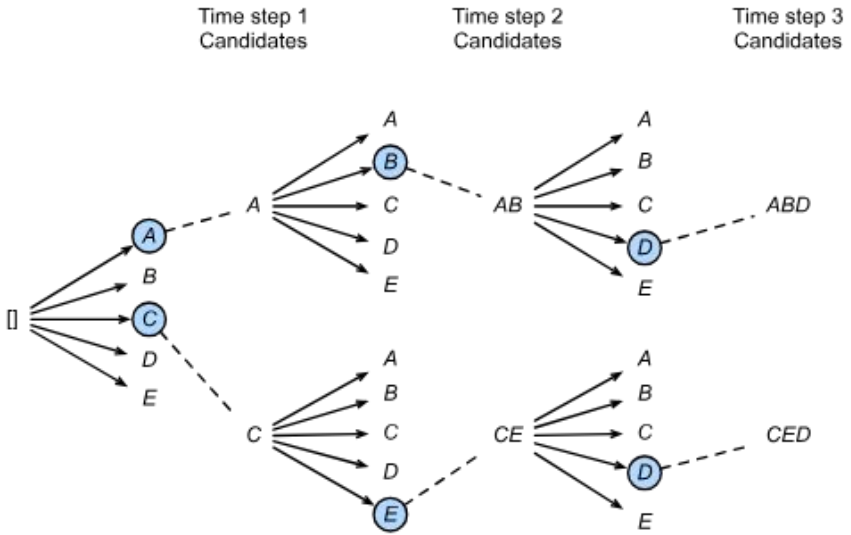
- What is the expansion of ISIT?
- Translate to greek: “this is the best conference”

Hard to compute it efficiently: requires a brute force search over all outputs

Technique 1: most-likely output

Approximation 1: greedy decoding $x_n = \arg\max_x p(x \mid \text{input}, x_1, \ldots, x_{n-1})$

Approximation 2: beam search



Graphic credit: Dive into Deep Learning Zhang et al., 2021

Technique 2: sampling

$$x_n \sim p(\cdot \mid \text{input}, x_1, \ldots, x_{n-1})$$

Additional techniques

- Temperature sampling: $x_n \sim p^{1/T}(\cdot \mid \text{input}, x_1, \ldots, x_{n-1})$
- Nucleus sampling: sample restricted to top-p probability mass.
- Top-k sampling: sample restricted to top-k tokens.
- Local typical sampling: sample to ensure that the sequence has log-likelihood close to entropy.

Most likely output vs sampling

Most likely output	Sampling
Deterministic output	Diverse outputs
Short and terse outputs	Longer outputs
Repeated tokens	More hallucinations
Summarization, translation	Chatbots, creative tasks

Aligned (user-preferred) language model inference

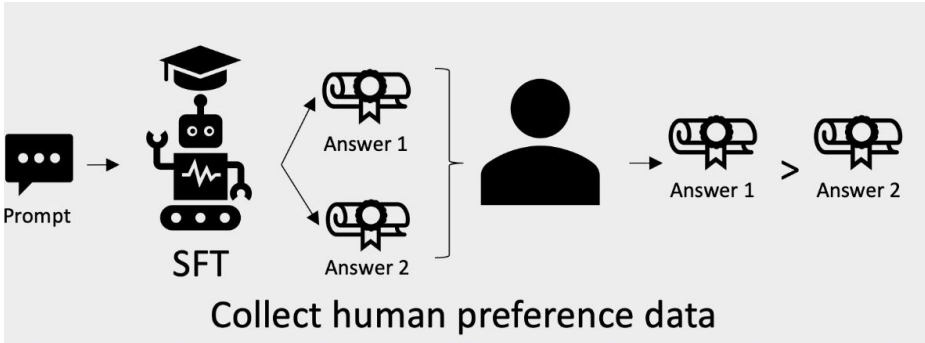
Stages of language modeling

- Pretraining (PT)
  - Train on large amounts of cleaned up language data (15T for Llama 3)
  - Long turnaround, and generally less controllable
- Supervised finetuning (SFT)
  - Train on specialized data whose style we want to imitate
  - Shorter turnaround, easier to steer model for desired behaviors

The task in both stages is next-word prediction using cross-entropy loss.

Last stage of language modeling (post-training)

- Preference optimization (PO)
  - Train on pairs of positive and negative examples to adapt to preferences
  - Very short turnaround, applies the final touches



Graphic credit: Amazon AWS

What is the definition of alignment?

- A generative **language model p(.|x)** is a distribution over outcome y given x.

- A **reward model r(x,y)** may be thought of as the log-likelihood of another generative **alignment language model q(.|x)**

$$r(x,y) = \log q(y|x)$$

- Expected reward is the negative cross entropy

$$E_{y\sim \mathbf{p}}[r(y)] = - H(p \,||\, q)$$

- Alignment Goal: Sample from the **aligned distribution π(.|x)** that leads improve *expected reward* but *remain "close to **p**."*

Side-by-side (s x s) reward modeling

- Elo rating

The difference in the ratings between two players serves as a predictor of the outcome of a match. If players A and B have ratings R^A and R^B, then the expected scores are given by:

$$E_A = \frac{1}{1 + 10^{(R_B - R_A)/400}}$$

Credit: Wikipedia entry for Elo rating

Side-by-side (s x s) reward modeling

- Bradley Terry model

$$p(y_1 \prec y_2 \mid x) = \sigma(r(x, y_2) - r(x, y_1))$$

- Reward model optimization with (x, y^-, y^+) triplets

$$\mathcal{J}(r) = \mathbb{E}_{(x,y^+,y^-)\sim D}\left[\log p(y^- \prec y^+ \mid x)\right]$$

Graphic credit: Amazon AWS

Best-of-n: A simple baseline for alignment

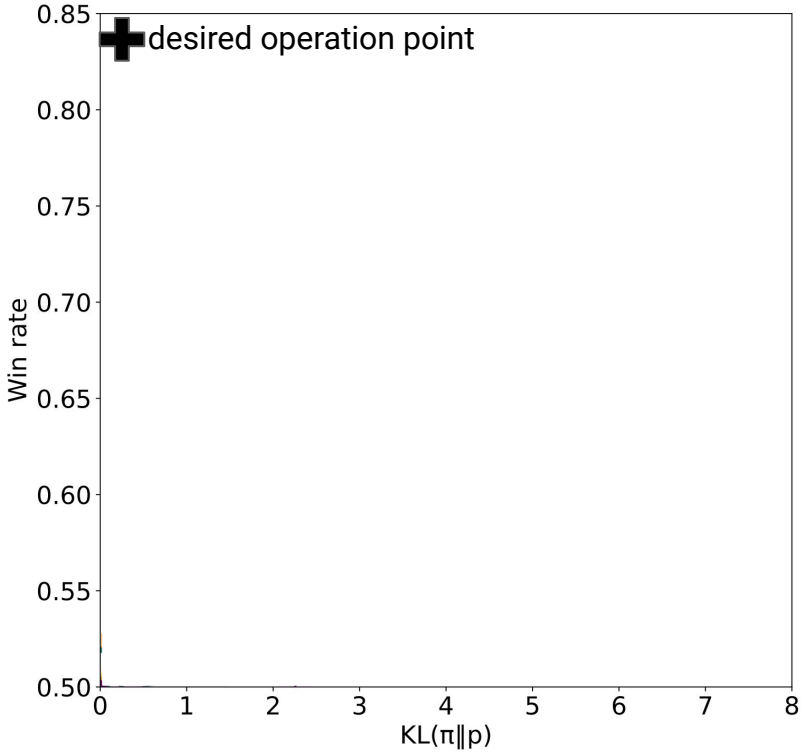
Let $y_1, \ldots, y_n$ be n i.i.d. draws from $p(\cdot|x)$. The best-of-n strategy is denoted by $\pi^{(n)}$ and returns

$$y = y_{k^*} \qquad \text{where} \qquad k^* := \arg\max_{k \in [n]} r(x, y).$$

How do we evaluate alignment methods?

- Human evaluations
- Auto-evals that are correlated with human judgement

How do we evaluate alignment methods?



Estimating KL divergence

- We estimate KL divergence via aggregating the log-likelihood ratios between the aligned model and the base model

$$D([\mathbf{x}, y^t]; \pi) := KL(\pi(\cdot|[\mathbf{x}, y^t]) \| p(\cdot|[\mathbf{x}, y^t]))$$
$$= \sum_{z \in \mathcal{Y}} \pi(z|[\mathbf{x}, y^t]) \log \left(\frac{\pi(z|[\mathbf{x}, y^t])}{p(z|[\mathbf{x}, y^t])} \right)$$

Log-likelihood ratio is an unbiased estimate of KL divergence

Estimating KL divergence

- We estimate KL divergence via aggregating the log-likelihood ratios between the aligned model and the base model

$$D([\mathbf{x}, y^t]; \pi) := KL(\pi(\cdot|[\mathbf{x}, y^t]) \| p(\cdot|[\mathbf{x}, y^t]))$$
$$= \sum_{z \in \mathcal{Y}} \pi(z|[\mathbf{x}, y^t]) \log \left(\frac{\pi(z|[\mathbf{x}, y^t])}{p(z|[\mathbf{x}, y^t])} \right)$$

Log-likelihood ratio is an unbiased estimate of KL divergence

- We don't have the logits of best-of-n.
  How can we estimate KL divergence?

Analytical formula for KL divergence of best-of-n

- An analytical formula that has appeared many times in the literature[1,2]

$$\mathrm{KL}(\pi^{(n)}_{\boldsymbol{y}|\boldsymbol{x}} \| p_{\boldsymbol{y}|\boldsymbol{x}}) \overset{\text{claim}}{=\!=} \widetilde{\mathrm{KL}}_n := \log(n) - (n-1)/n.$$

- This formula remarkably
  - doesn't depend on the prompt **x** or its distribution $\mathbf{p}_{\mathbf{x}}$
  - doesn't depend on the base policy $\mathbf{p}_{\mathbf{y|x}}$

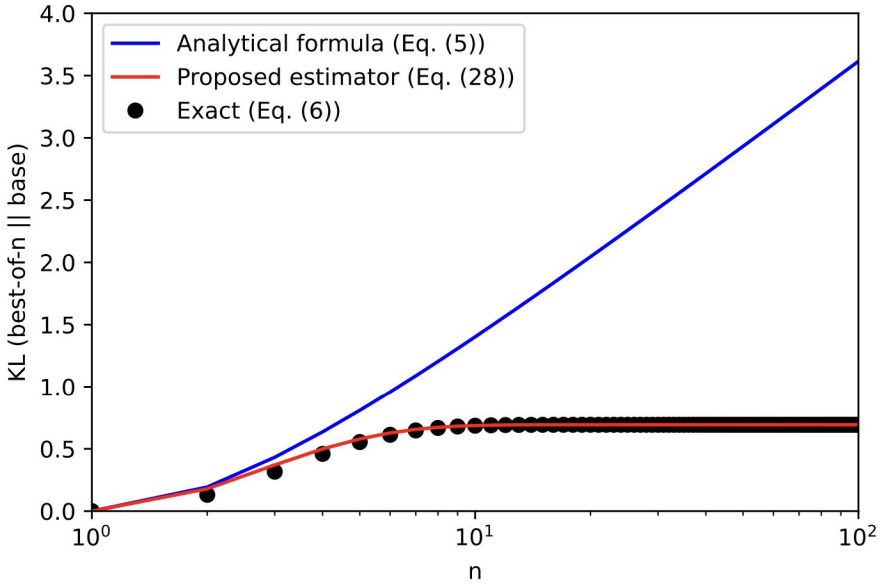
- Is this formula true?

[1]Measuring Goodhart's law (Hilton & Gao,2022).
[2]Learning to summarize with human feedback (Stiennon et al.,NeurIPS 2020).

Analytical formula is wrong!

Example 1. *Consider an unprompted model with* $\boldsymbol{x} = \emptyset$ *(no input) and binary output,* $\boldsymbol{y} \in \{0, 1\}$*. Let the two outcomes be equiprobable, i.e.,* $p_{\boldsymbol{y}|\boldsymbol{x}}(0) = p_{\boldsymbol{y}|\boldsymbol{x}}(1) = \frac{1}{2}$*. Further, let* $r(0) = 0$*, and* $r(1) = 1$*, i.e., outcome* 1 *is more desirable than outcome* 0*. Here, we can compute* $\pi^{(n)}_{\boldsymbol{y}|\boldsymbol{x}}$ *in closed form. Specifically, we can see that* $\pi^{(n)}_{\boldsymbol{y}|\boldsymbol{x}}(0) = \frac{1}{2^n}$ *and* $\pi^{(n)}_{\boldsymbol{y}|\boldsymbol{x}}(1) = 1 - \frac{1}{2^n}$*. Thus,*

$$KL(\pi^{(n)}_{\boldsymbol{y}|\boldsymbol{x}} \| p_{\boldsymbol{y}|\boldsymbol{x}}) = \log(2) - h\left(\frac{1}{2^n}\right)$$



[1]Theoretical guarantees on the best-of-n alignment policy (Beirami et al., arXiv preprint 2024).

The PMF of best-of-n

$$\pi^{(n)}_{\boldsymbol{y}|\boldsymbol{x}}(\boldsymbol{y}|\boldsymbol{x}) = \mathcal{F}(\boldsymbol{y}|\boldsymbol{x})^n - \mathcal{F}^-(\boldsymbol{y}|\boldsymbol{x})^n,$$

where

$$\mathcal{F}(\boldsymbol{y}|\boldsymbol{x}) := P_{\boldsymbol{z}\sim p_{\boldsymbol{y}|\boldsymbol{x}}}[r(\boldsymbol{x},\boldsymbol{z}) \le r(\boldsymbol{x},\boldsymbol{y})],$$
$$\mathcal{F}^-(\boldsymbol{y}|\boldsymbol{x}) := P_{\boldsymbol{z}\sim p_{\boldsymbol{y}|\boldsymbol{x}}}[r(\boldsymbol{x},\boldsymbol{z}) < r(\boldsymbol{x},\boldsymbol{y})].$$

[1]Theoretical guarantees on the best-of-n alignment policy (Beirami et al., arXiv preprint 2024).

Proof

First notice that sampling from $p_{\boldsymbol{y}|\boldsymbol{x}}$ is akin to sampling $u \sim \mathcal{U}[0,1]$, and returning $\widetilde{\boldsymbol{y}}_i$, such that

$$\mathcal{F}(\widetilde{\boldsymbol{y}}_{i-1}|\boldsymbol{x}) \leq u < \mathcal{F}(\widetilde{\boldsymbol{y}}_i|\boldsymbol{x}). \tag{10}$$

Similarly, sampling from the best-of-n strategy is akin to sampling $u_1, \ldots, u_n \overset{\text{i.i.d.}}{\sim} \mathcal{U}[0,1]$, and returning $\widetilde{\boldsymbol{y}}_i$, such that

$$\mathcal{F}(\widetilde{\boldsymbol{y}}_{i-1}|\boldsymbol{x}) \leq \max_{k\in[n]} u_k < \mathcal{F}(\widetilde{\boldsymbol{y}}_i|\boldsymbol{x}). \tag{11}$$

On the other hand, we know that the CDF of the maximum for all $\tau \in [0,1]$ is given by

$$P\left[\max_{k\in[n]} u_k \leq \tau\right] = \tau^n. \tag{12}$$

Hence, for all $n \in \mathbb{N}$, the PMF of the best-of-n policy, denoted as $\pi^{(n)}_{\boldsymbol{y}|\boldsymbol{x}}$ is given by

$$\pi^{(n)}_{\boldsymbol{y}|\boldsymbol{x}}(\widetilde{\boldsymbol{y}}_i|\boldsymbol{x}) = \mathcal{F}(\widetilde{\boldsymbol{y}}_i|\boldsymbol{x})^n - \mathcal{F}(\widetilde{\boldsymbol{y}}_{i-1}|\boldsymbol{x})^n \qquad \forall i \in [L_{\boldsymbol{x}}], \tag{13}$$

where $\mathcal{F}(\widetilde{\boldsymbol{y}}_0|\boldsymbol{x}) := 0$, and

$$\mathcal{F}(\widetilde{\boldsymbol{y}}_i|\boldsymbol{x}) = \sum_{l\in[i]} p_{\boldsymbol{y}|\boldsymbol{x}}(\widetilde{\boldsymbol{y}}_l|\boldsymbol{x}). \tag{14}$$

The proof is completed by noticing that $\mathcal{F}(\widetilde{\boldsymbol{y}}_{i-1}|\boldsymbol{x}) = \mathcal{F}^{-}(\widetilde{\boldsymbol{y}}_i|\boldsymbol{x})$. □

[1]Theoretical guarantees on the best-of-n alignment policy (Beirami et al., arXiv preprint 2024).

Guarantees on the analytical formula

Theorem 2. *For any $n \in \mathbb{N}$ and any $\boldsymbol{x}$,*

$$KL(\pi^{(n)} \| p) \leq \widetilde{KL}_n = \log(n) - \frac{n-1}{n}.$$

- Analytical formula is an upper bound
- Let $y \sim \pi^{(n)}$. Then, let $\varepsilon_n := p(y)$
  - Theorem: The gap is small if $\mathbf{n.\varepsilon_n \ll 1}$
  - Theorem: The gap is large if $\mathbf{n.\varepsilon_n \gg 1}$

[1]Theoretical guarantees on the best-of-n alignment policy (Beirami et al., arXiv preprint 2024).

Proof of the upper bound on KL

Lemma 3. *For any* $0 \leq a < b \leq 1$, *and* $n \in \mathbb{N}$,

$$(b^n - a^n) \log \frac{b^n - a^n}{b - a} = \int_a^b nv^{n-1} \log \left(nv^{n-1}\right) dv - g_n(a, b) \leq \int_a^b nv^{n-1} \log \left(nv^{n-1}\right) dv,$$

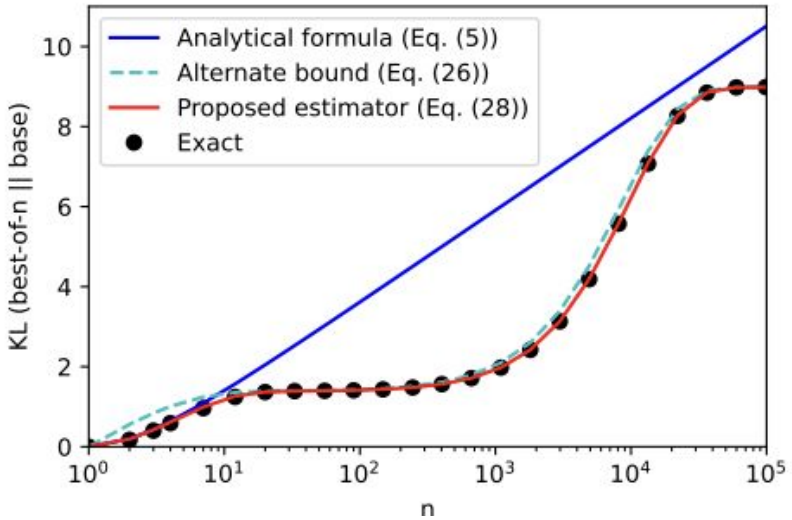
- The result could also be proved through mapping of measures and data-processing inequality.[2]

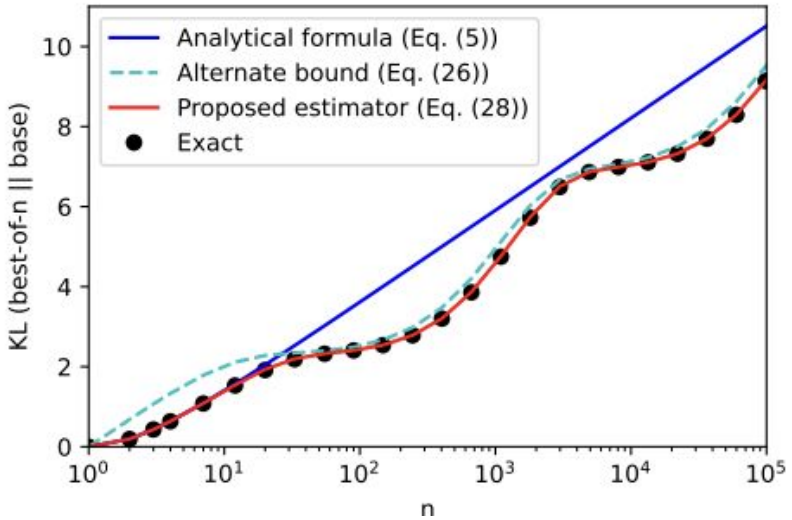
[1]Theoretical guarantees on the best-of-n alignment policy (Beirami et al., arXiv preprint 2024).
[2]Information Theoretic Guarantees For Policy Alignment In Large Language Models (Mroueh, arxiv preprint 2024)

New estimator for KL divergence of best-of-n

Approximation 1. *Let $y \sim \pi^{(n)}$. Then, let $\varepsilon_n := p(y)$. We propose the following estimator for the KL divergence of the best-of-n policy and the base policy:*

$$\widehat{KL}(\varepsilon_n) := (1-\varepsilon_n)^n \left(\log n + (n-1)\log(1-\varepsilon_n) - \frac{n-1}{n} \right) + (1-(1-\varepsilon_n)^n) \log \left(\frac{1-(1-\varepsilon_n)^n}{\varepsilon_n} \right).$$





How to estimate win-rate for best-of-n?

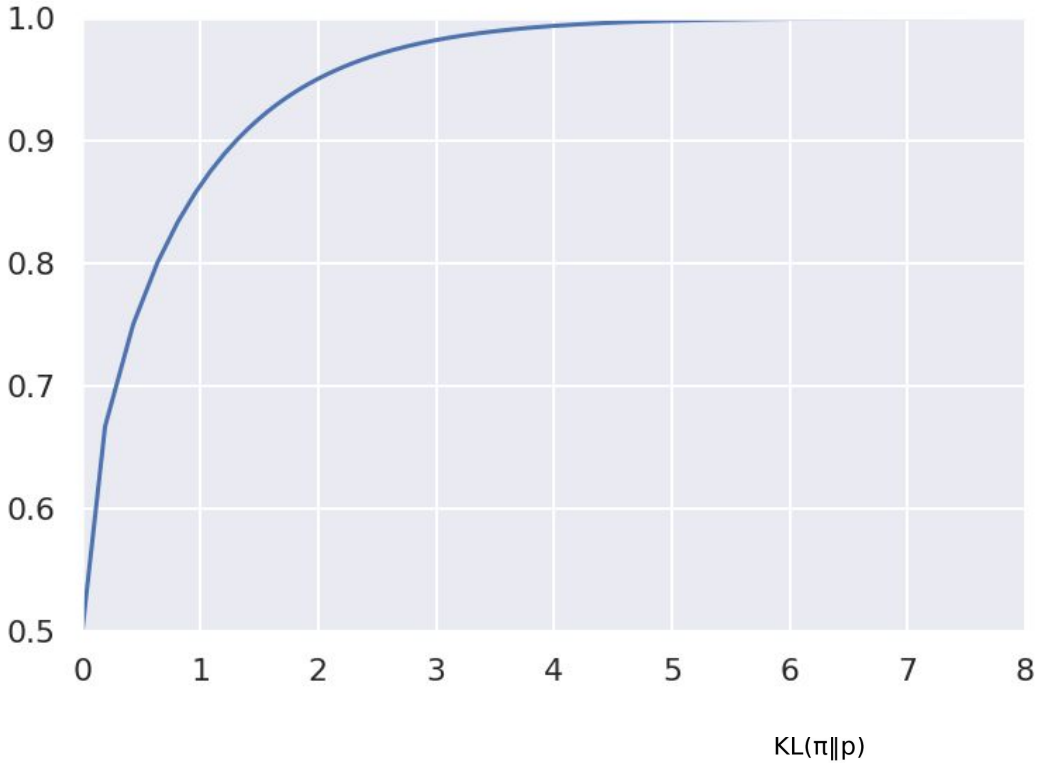
$$\text{win-rate}(\pi \| p) := E_{y\sim\pi} E_{z\sim p}\{\mathbf{1}(r(x,y) > r(x,z)) + \frac{1}{2}\mathbf{1}(r(x,y) = r(x,z))\}$$

Lemma 1. *The win-rate for best-of-n policy is given by*

$$\textit{win-rate}(\pi^{(n)} \| p) = 1 - \frac{1}{2} E_{y\sim p}\{F(y|x)^n + F^-(y|x)^n\} \leq \frac{n}{n+1}.$$

- Roughly the upper bound argument follows from
  - Draw (n+1) outcomes from the base model; and order them from the highest to lowest reward
  - Randomly assign 1 of the outcomes to base model
  - Choose the best of the remaining n to be a draw from the best-of-n model.
  - Best-of-n wins against the reference model with probability n/(n+1).
- The upper bound would be exact if w/p 1 no two outcomes were identical

Win-rate vs KL for best-of-n



- KL values <10 are enough to reach a high win-rate against base policy
- This is ignoring the noise in reward and generalization

Best-of-n: A simple baseline for alignment

Let $y_1, \ldots, y_n$ be n i.i.d. draws from $p(\cdot|x)$. The best-of-n strategy is denoted by $\pi^{(n)}$ and returns

$$y = y_{k^*} \qquad \text{where} \qquad k^* := \arg\max_{k \in [n]} r(x, y).$$

- expensive in terms of throughput
- incompatible with streaming

- In the sequel, we explore practical alternatives with reasonable throughput and latency

KL-constrained reinforcement learning

Definition 1 (alignment distribution). *For a given reward function* $\boldsymbol{r}$*, we denote* $\boldsymbol{q}_{y|x}$ *as the corresponding alignment distribution defined as*

$$\boldsymbol{q}_{y|x}(\boldsymbol{y}|\boldsymbol{x}) = \exp\left(\boldsymbol{r}(\boldsymbol{x},\boldsymbol{y})\right)/R_{\boldsymbol{x}},$$

where $R_{\boldsymbol{x}}$ *is the partition function, i.e.,* $R_{\boldsymbol{x}} := \sum_{y\in\mathcal{Y}} \exp\left(\boldsymbol{r}(\boldsymbol{x},\boldsymbol{y})\right)$.

Definition 2 (KL-constrained RL). *We say* $\boldsymbol{\phi}_\Delta$ *is an optimal aligned model of* $\boldsymbol{p}$ *with KL divergence* Δ*, if*

$$\boldsymbol{\phi}_\Delta(\cdot|\boldsymbol{x}) \in \arg\min_{\boldsymbol{\phi}\in\mathcal{D}_\Delta(\boldsymbol{p})} H(\boldsymbol{\phi}(\cdot|\boldsymbol{x})\|\boldsymbol{q}(\cdot|\boldsymbol{x})), \tag{2}$$

where $\mathcal{D}_\Delta(\boldsymbol{p}) := \{\boldsymbol{\phi} : D_{\mathrm{KL}}(\boldsymbol{\phi}(\cdot|\boldsymbol{x})\|\boldsymbol{p}(\cdot|\boldsymbol{x})) \le \Delta\}$.

Closed-form solution of KL-regularized RL

Lemma 1 (optimal aligned model). *The solution of the KL-constrained RL problem in Definition 2 is unique and is given by*

$$\boldsymbol{\phi}_{\Delta}(\cdot|\boldsymbol{x}) = T(\boldsymbol{q}(\cdot|\boldsymbol{x}), \boldsymbol{p}(\cdot|\boldsymbol{x}), \alpha(\Delta)), \tag{4}$$

where $T(\boldsymbol{q}(\cdot|\boldsymbol{x}), \boldsymbol{p}(\cdot|\boldsymbol{x}), \alpha)$ *is the mismatched tilt (Salamatian et al., 2019, Definition 1):*

$$T(\boldsymbol{q}(\cdot|\boldsymbol{x}), \boldsymbol{p}(\cdot|\boldsymbol{x}), \alpha)(\boldsymbol{y}) := \frac{\boldsymbol{p}(\boldsymbol{y}|\boldsymbol{x})\boldsymbol{q}^{\alpha}(\boldsymbol{y}|\boldsymbol{x})}{Z_{\boldsymbol{x},\alpha}}, \tag{5}$$

where

$$Z_{\boldsymbol{x},\alpha} := \sum_{\boldsymbol{z}\in\mathcal{Y}} \boldsymbol{p}(\boldsymbol{z}|\boldsymbol{x})\boldsymbol{q}^{\alpha}(\boldsymbol{z}|\boldsymbol{x}), \tag{6}$$

and

$$\alpha(\Delta) := \arg_{\alpha\in\mathbb{R}^{\geq 0}} \{D_{\mathrm{KL}}(T(\boldsymbol{q}(\cdot|\boldsymbol{x}), \boldsymbol{p}(\cdot|\boldsymbol{x}), \alpha)\|\boldsymbol{p}) = \Delta\}. \tag{7}$$

Proof

Proof of Lemma 1. The Lagrangian is written as

$$\mathcal{L}(\boldsymbol{\phi}) = H(\boldsymbol{\phi}\|\boldsymbol{q}) + \lambda\left(D_{\mathrm{KL}}(\boldsymbol{\phi}\|\boldsymbol{p}) - \Delta\right). \tag{29}$$

Fix $\lambda > 0$ we can solve for its minimum with respect to $\boldsymbol{\phi}$ following steps,

$$\min_{\boldsymbol{\phi}} \mathcal{L}(\boldsymbol{\phi}) = \min_{\boldsymbol{\phi}} H(\boldsymbol{\phi}\|\boldsymbol{q}) + \lambda\left(D_{\mathrm{KL}}(\boldsymbol{\phi}\|\boldsymbol{p}) - \Delta\right) \tag{30}$$

$$= \min_{\boldsymbol{\phi}} \sum_{y\in\mathcal{Y}} \boldsymbol{\phi}(y)\left(\log\frac{1}{\boldsymbol{q}(y)} + \lambda\log\frac{\boldsymbol{\phi}(y)}{\boldsymbol{p}(y)} - \lambda\Delta\right) \tag{31}$$

$$= \min_{\boldsymbol{\phi}} \lambda\sum_{y} \boldsymbol{\phi}(y)\left(\log\frac{\boldsymbol{\phi}(y)}{\boldsymbol{p}(y)\boldsymbol{q}(y)^{1/\lambda}} - \Delta\right) \tag{32}$$

$$= \min_{\boldsymbol{\phi}} \lambda\sum_{y} \boldsymbol{\phi}(y)\left(\log\frac{\boldsymbol{\phi}(y)}{T(\boldsymbol{q},\boldsymbol{p},1/\lambda)(y)}\right) + C(\lambda,\Delta), \tag{33}$$

$$= \min_{\boldsymbol{\phi}} \lambda D_{\mathrm{KL}}(\boldsymbol{\phi}(y)\|T(\boldsymbol{q},\boldsymbol{p},1/\lambda)(y)) + C(\lambda,\Delta), \tag{34}$$

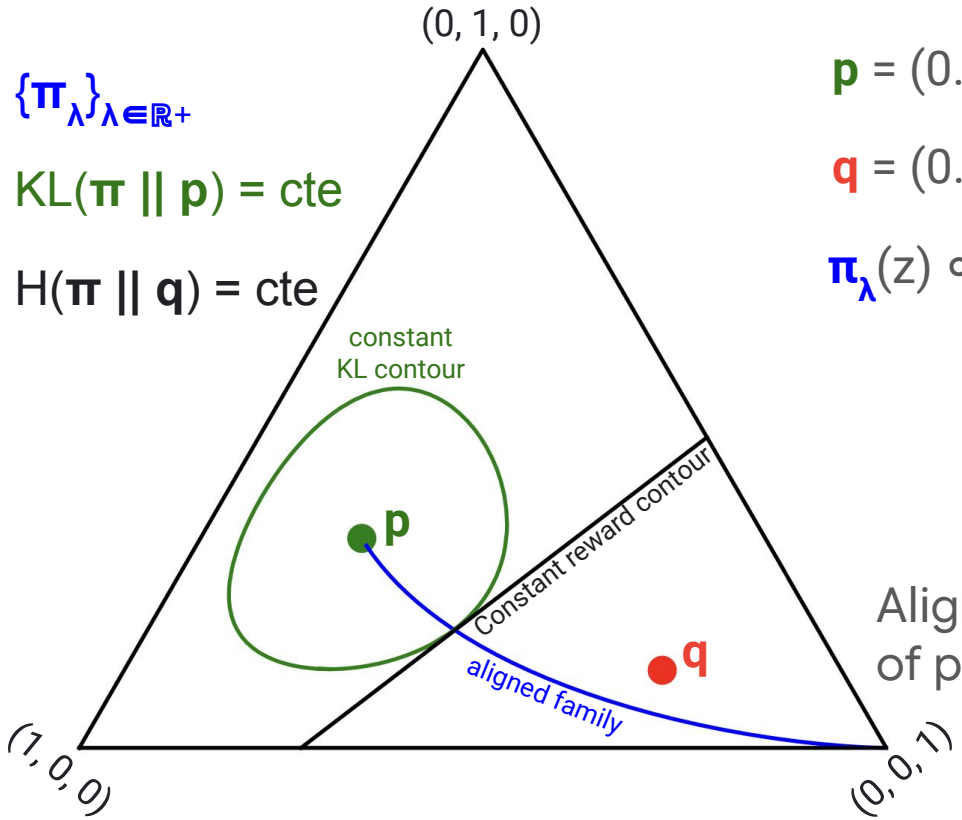
which is uniquely minimized by $\boldsymbol{\phi}(y) = T(\boldsymbol{q},\boldsymbol{p},1/\lambda)(y)$. □

The solution for a tabular model

Aligned family: $\{\pi_\lambda\}_{\lambda \in \mathbb{R}+}$

KL contour: $KL(\pi \,||\, p) = cte$

Reward contour: $H(\pi \,||\, q) = cte$



$p = (0.50, 0.30, 0.20)$

$q = (0.22, 0.11, 0.67)$

$\pi_\lambda(z) \propto p(z)\, q(z)^\lambda$

Alignment leads to loss of perplexity

The closed-form solution is intractable

Proximal Policy Optimization (PPO)

- In practice, we use policy gradient theorem to solve policy optimization:

$$
\begin{aligned}
\nabla \mathbb{E}_{\pi}\left[r(\tau)\right] &= \nabla \int \pi(\tau) r(\tau) d\tau \\
&= \int \nabla \pi(\tau) r(\tau) d\tau \\
&= \int \pi(\tau) \nabla \log \pi(\tau) r(\tau) d\tau \\
\nabla \mathbb{E}_{\pi}\left[r(\tau)\right] &= \mathbb{E}_{\pi}\left[r(\tau) \nabla \log \pi(\tau)\right]
\end{aligned}
$$

How can we utilize the closed-form solution?

$$\pi^*_{\text{KL-RL}}(x) = \frac{1}{Z}\pi_0(x)\exp(r(x)/\beta),$$

$$p(y_1 \prec y_2 \mid x) = \sigma(r(x, y_2) - r(x, y_1))$$

$$\mathcal{J}(r) = \mathbb{E}_{(x,y^+,y^-)\sim D}\left[\log p(y^- \prec y^+ \mid x)\right]$$

Many alignment methods

- Direct preference optimization (DPO) from (x, y^-, y^+) triplets[1]

$$\max_{\pi} \mathbb{E}_{(x,y^+,y^-)\sim D}\left[\log\sigma\left(\beta\log\frac{\pi(y^+|x)}{p(y^+|x)} - \beta\log\frac{\pi(y^-|x)}{p(y^-|x)}\right)\right]$$

- Many more!!![2-6]

[1]Direct Preference Optimization: Your Language Model is Secretly a Reward Model (Rafailov et al., NeurIPS 2023).
[2]A General Theoretical Paradigm to Understand Learning from Human Preferences (Azar et al., AISTATS 2024).
[3]KTO: Model Alignment as Prospect Theoretic Optimization (Ethayarajh et al., ICML 2024).
[4]Controlled decoding from language models (Mudgal et al., ICML 2024).
[5]Robust Preference Optimization through Reward Model Distillation (Fisch et al., arxiv 2024)
[6]SimPO: Simple Preference Optimization with a Reference-Free Reward (Meng et al., arxiv 2024)

Controlled decoding (CD)

Theorem 1. *The optimal policy for the RL objective is unique and is given by*

$$\pi^{\star}_{\lambda}(z|[\mathbf{x}, y^t]) \propto p(z|[\mathbf{x}, y^t]) e^{\lambda V^{\star}([\mathbf{x}, y^t, z])}.$$

$$\mathcal{L}^{\star}(\boldsymbol{\theta}) = E_{\mathbf{x}\sim p_{\mathbf{x}}} E_{\mathbf{y}\sim p_{\mathbf{y}|\mathbf{x}}} \ell^{\star}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}),$$

$$\text{where } \ell^{\star}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}) = \frac{1}{2} \sum_{t \in [|\mathbf{y}|]} (V_{\boldsymbol{\theta}}([\mathbf{x}, y^t]) - V^{\star}([\mathbf{x}, y^t]))^2$$

CD-FUDGE

- Use an unbiased draw from the model as the target[1,2]

$$\mathcal{L}_F(\boldsymbol{\theta}) = E_{\mathbf{x}\sim p_{\mathbf{x}}} \ell_F(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}), \qquad \text{s.t. } \mathbf{y} \sim p,$$

$$\text{where } \ell_F(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}) = \frac{1}{2} \sum_{t \in [|\mathbf{y}|]} \left(V_{\boldsymbol{\theta}}([\mathbf{x}, y^t]) - r([\mathbf{x}, \mathbf{y}]) \right)^2$$

Theorem 3.1 (informal). *Under regularity assumptions, applying SGD on $\mathcal{L}_F$ converges to a stationary point of $\mathcal{L}^\star(\boldsymbol{\theta})$.*

[1]Controlled Text Generation With Future Discriminators (Yang & Klein, NAACL 2021).
[2]Controlled decoding from language models (Mudgal, Lee et al., ICML 2024).

CD-Q

- Bellman identity

$$V^\star([\mathbf{x}, y^t]) = \begin{cases} E_{z\sim p(\cdot|[x,y^t])} V^\star([\mathbf{x}, y^t, z]), & y_t \neq EOS \\ r([\mathbf{x}, y^t]), & y_t = EOS \end{cases}$$

- Train the value function similarly to DQN[1,2]

$$\mathcal{L}_Q(\boldsymbol{\theta}) = E_{\mathbf{x}\sim p_\mathbf{x}} \ell_Q(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}),$$

$$\text{where } \ell_Q(\mathbf{x}, y^t; \boldsymbol{\theta}) = \frac{1}{2} \sum_{t\in[|\mathbf{y}|]} \left(V_{\boldsymbol{\theta}}([\mathbf{x}, y^t]) - \dot{v}_t\right)^2,$$

$$v_t = \begin{cases} \sum_{z\in\mathcal{Y}} p(z|[x, y^t]) V_{\boldsymbol{\theta}}([\mathbf{x}, y^t, z]) & y_t \neq EOS \\ r([\mathbf{x}, y^t]) & y_t = EOS \end{cases}$$

[1]Reinforcement Learning: An Introduction (Sutton & Barto,2018).
[2]Playing Atari with Deep Reinforcement Learning (Mnih et al.,2013).

Token-wise control using CD

Theorem 2.1. *The optimal policy for the RL objective is unique and is given by*

$$\pi^{\star}_{\lambda}(z|[\mathbf{x}, y^t]) \propto p(z|[\mathbf{x}, y^t])e^{\lambda V^{\star}([\mathbf{x}, y^t, z])}.$$

Will this paper get accepted?

This paper will be

	LM likelihood	sentiment prefix score	aligned score
liked	average	very high	high
disliked	high	low	average
reviewed	high	average	average
hated	very high	very low	average

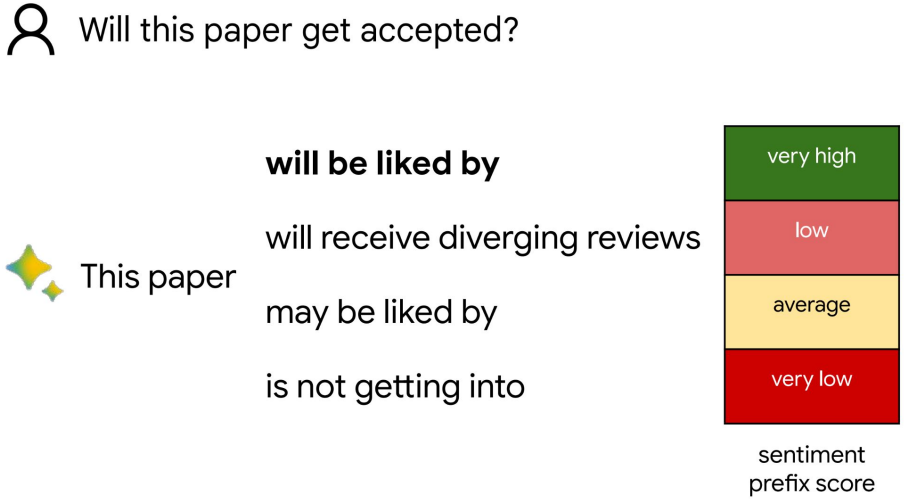
Blockwise control using CD (best-of-n++)

- Draw K blocks of length M tokens

$$\left\{z^M_{(k)}\right\}_{k\in[K]} \overset{\text{i.i.d.}}{\sim} p(z^M|[\mathbf{x}, y^t])$$

- Accept the continuation with the highest prefix score:

$$z^M := \arg \max_{\left\{z^M_{(k)}\right\}_{k\in[K]}} V_{\boldsymbol{\theta}}([\mathbf{x}, y^t, z^M_{(k)}])$$



Advantages over best-of-n:
- Limits the user-facing latency to the decoding time of a single block.
- Makes a large effective n practically feasible.

Reward-KL tradeoffs (length reward)

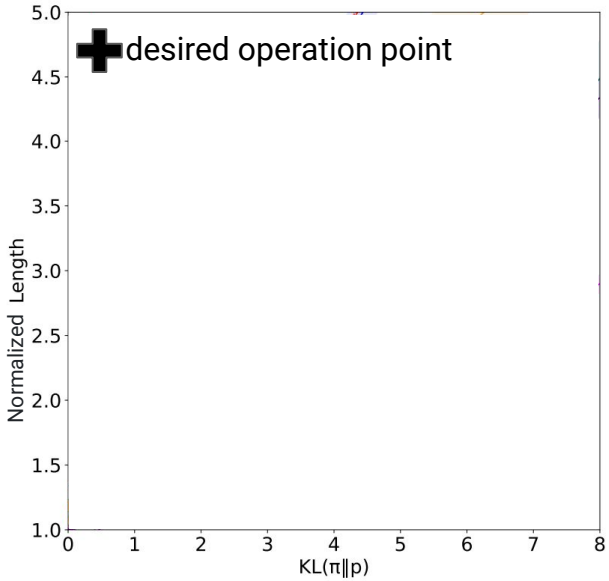


Figure 3: Length vs. KL divergence for different length alignment methods.

Reward-KL tradeoffs (length reward)

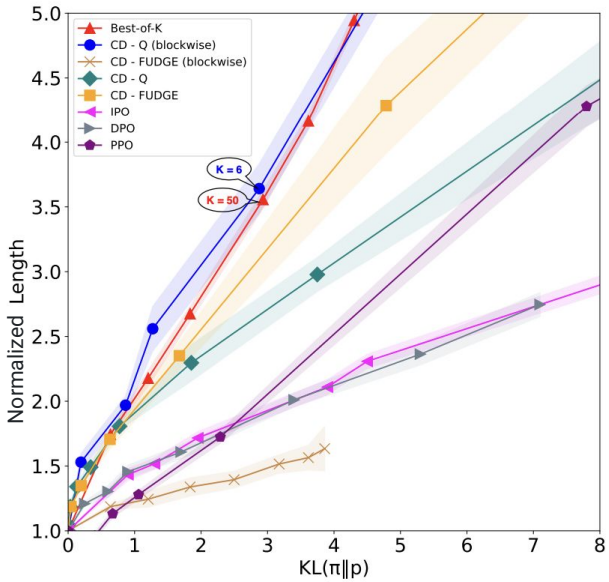


Figure 3: Length vs. KL divergence for different length alignment methods. CD-Q (blockwise) outperforms RL techniques such as IPO & PPO and is on par with best-ok-K while being much more efficient as it requires far fewer samples (e.g. 6 vs 50).

- Best-of-n is better than state-of-the-art RL methods
- Blockwise CD bridges the gap between tokenwise control and best-of-n
- Token-wise CD is a good contender for token-wise control

Reward-KL tradeoffs (helpfulness & harmlessness)

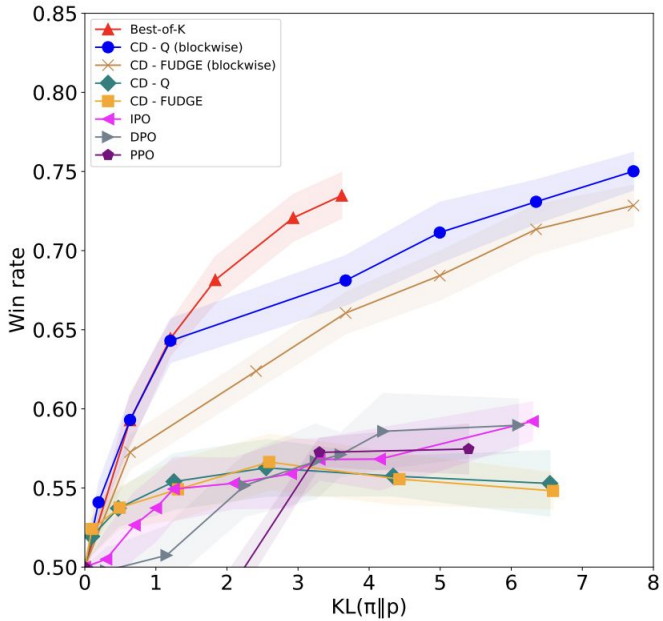
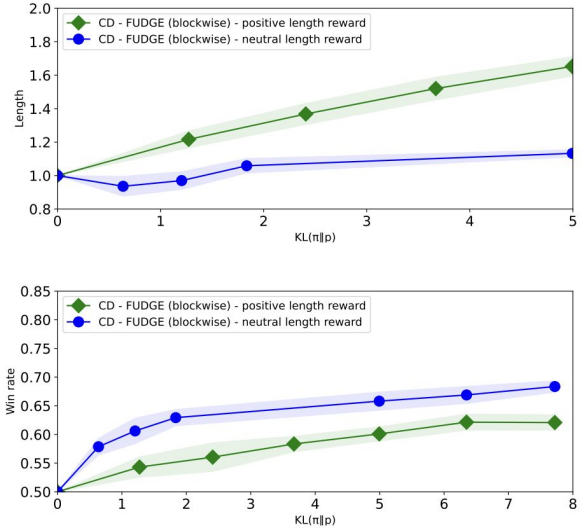


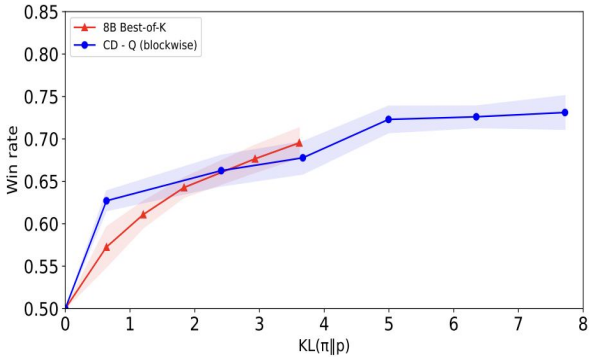
Figure 4: Win rate vs. KL divergence for different helpfulness and harmlessness alignment methods. CD-Q (blockwise) vastly outperforms RL techniques such as IPO & PPO.

- Similar observations for more realistic rewards

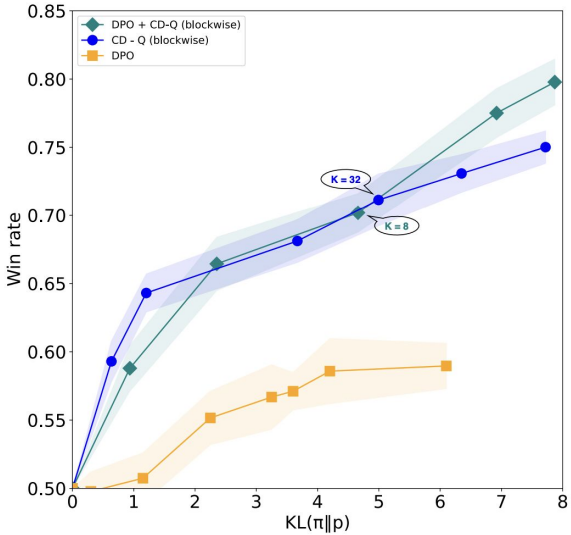
Modularity of CD for the win!



Multi-objective alignment



generalization to a new model



Integrating CD and DPO

Optimal reward-KL tradeoff

- KL-regularized RL solution is optimal for reward-KL tradeoff

$$\max_{\pi} \underbrace{\mathbb{E}_{\substack{x\sim\rho \\ y\sim\pi}}[r(x,y)]}_{\text{Linear in } \pi} - \underbrace{\lambda \mathrm{KL}(\pi \| p)}_{\text{Strongly convex in } \pi}$$

Optimal reward-KL tradeoff

- KL-regularized RL solution is optimal for reward-KL tradeoff

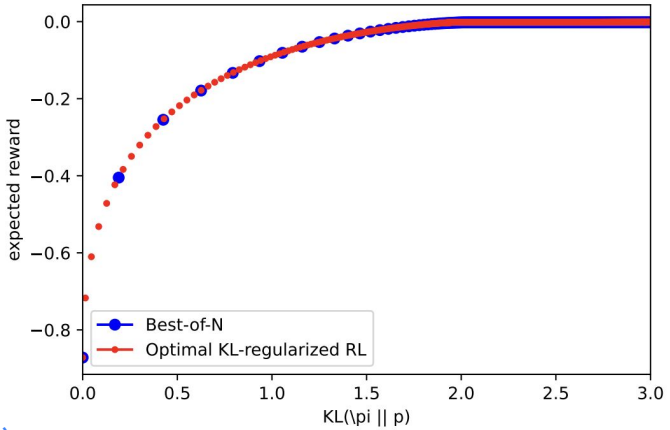
$$\max_{\pi} \underbrace{\mathbb{E}_{\substack{x\sim\rho \\ y\sim\pi}}[r(x,y)]}_{\text{Linear in } \pi} - \underbrace{\lambda \mathrm{KL}(\pi \| p)}_{\text{Strongly convex in } \pi}$$

- Empirically, best-of-n is strikingly close to the optimal trade-off

TH1.R1.1: ASYMPTOTICS OF LANGUAGE MODEL ALIGNMENT

Thu, 11 Jul, 09:45 - 10:05

Joy Yang, University of Sydney, Australia; Salman Salamatian, MIT, United States; Ziteng Sun, Ananda Theertha Suresh, Ahmad Beirami, Google, Australia



Asymptotics of Language Model Alignment (Yang et al., 2024).

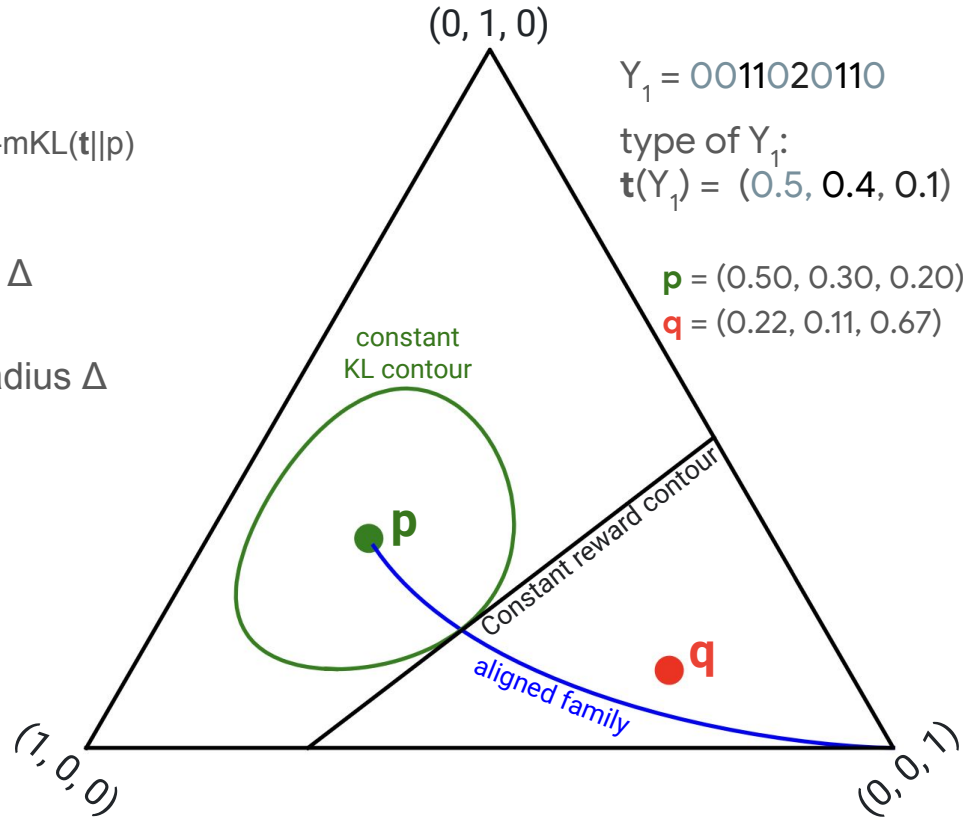
Why does best-of-n work so well?

- Let's revisit the example
- The probability of type **t** is given by $e^{-mKL(\mathbf{t}||p)}$
- Let $n = e^{m\Delta}$, then
  - Lemma: Any type **t** in the KL ball of radius Δ is sampled almost surely
  - Lemma: No type **t** outside the KL ball of radius Δ is sampled almost surely

Theorem 2. *Let ϕ_Δ be the optimal solution to Definition 2, and π_N^m be the distribution of the best-of-N, with $N = \exp(m\Delta)$. Under Assumption 1, we have that for all $\boldsymbol{x}$,*

$$\lim_{m\to\infty} \frac{1}{m} D_{\mathrm{KL}}(\pi_N^m(\cdot|\boldsymbol{x}) \| \phi_\Delta^m(\cdot|\boldsymbol{x})) = 0. \qquad (5)$$

RL solution w/ KL constraint of Δ



Asymptotics of Language Model Alignment (Yang et al., 2024).

Further considerations

- Recall the objective

$$\max_{\pi} \mathbb{E}_{\substack{x\sim\rho \\ y\sim\pi}}[r(x,y)] - \lambda \mathrm{KL}(\pi \| p)$$

- Ensemble rewards for more robust preference optimization.[1]
- Distill preference optimization from an ensemble.[2]
- Reusing English reward can help kickstart multi-lingual preference optimization.[3]
- Red teaming can help find diverse prompts that trigger the model.[4]
- Train rewards from a handful of loss patterns.[5]

[1]Helping or Herding? Reward Model Ensembles Mitigate but do not Eliminate Reward Hacking (Eisenstein,et al., 2024).
[2]Robust Preference Optimization through Reward Model Distillation (Fisch et al, 2024)
[3]Reuse Your Rewards: Reward Model Transfer for Zero-Shot Cross-Lingual Alignment (Wu et al.,2024).
[4]Gradient-Based Language Model Red Teaming (Wichers et al., EACL 2024)
[5]Improving Few-shot Generalization of Safety Classifiers via Data Augmented Parameter-Efficient Fine-Tuning (Balashankar et al., 2024)

Takeaways (alignment recipe)

- Step 1: Perform Best-of-n and make sure it works as desired.
  - Inspect a few responses and verify that the ranking induced by reward makes sense.
  - Best-of-n essentially gives the best tradeoffs you can hope for so if best-of-n doesn't work for your problem, no other fancy method will!
  - You'd also be able to debug best-of-n much faster.

- Step 2: Only then train your favorite alignment method.
  - Track KL($\boldsymbol{\pi}$ || $\mathbf{p}$) throughout training
    - KL > 100 The results are unlikely to be any useful!
    - KL > 15 Inspect the outcomes for reward hacking!
    - KL < 8 You are probably OK!

Efficient language model inference

Recap: how to infer from language models

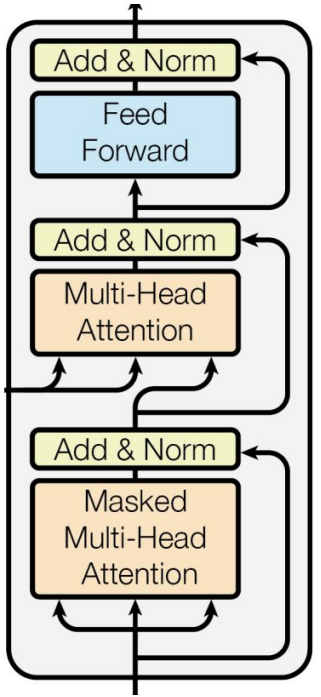
Transformer models are next token predictors

Given $x_1, x_2, \ldots, x_n$ transformer model computes

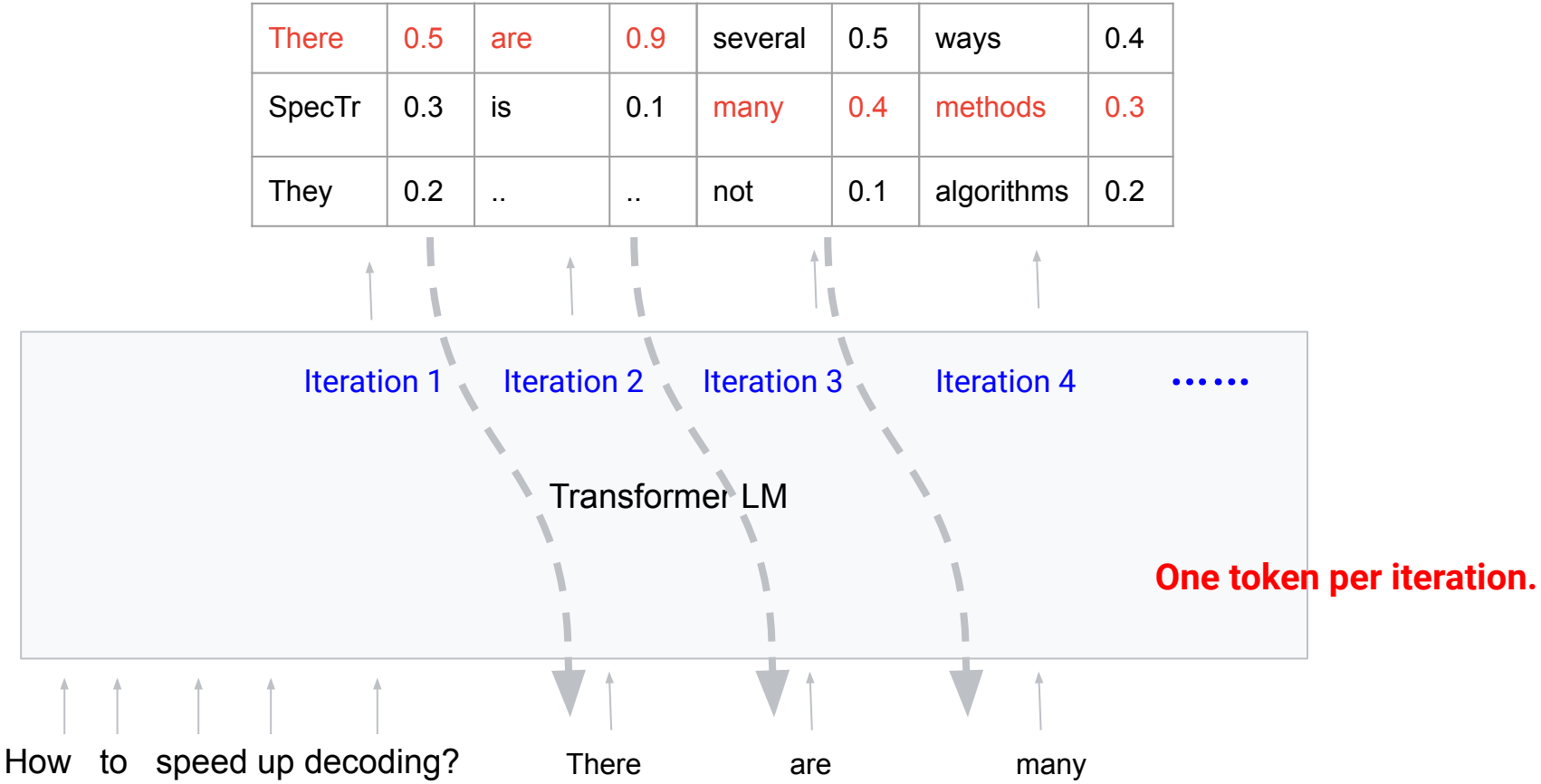
$$p(x_{n+1} \mid x_1, x_2, \ldots, x_n)$$

Next token is computed via autoregressive sampling

$$x_{n+1} \sim p(x \mid x_1, x_2, \ldots, x_n)$$

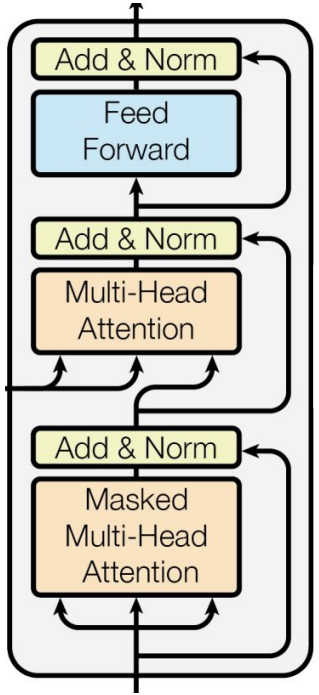


Autoregressive language models



Transformer language models

- Repeated layers of transformer block
- Billions of parameters
- Tokens are generated one at a time
- Each generation requires going over all parameters

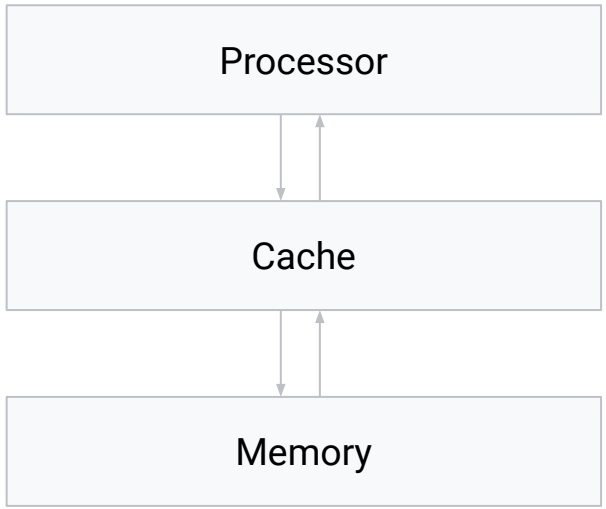


This process can be slow

A typical computing system

Language model inference is typically done on accelerators

- GPUs and TPUs



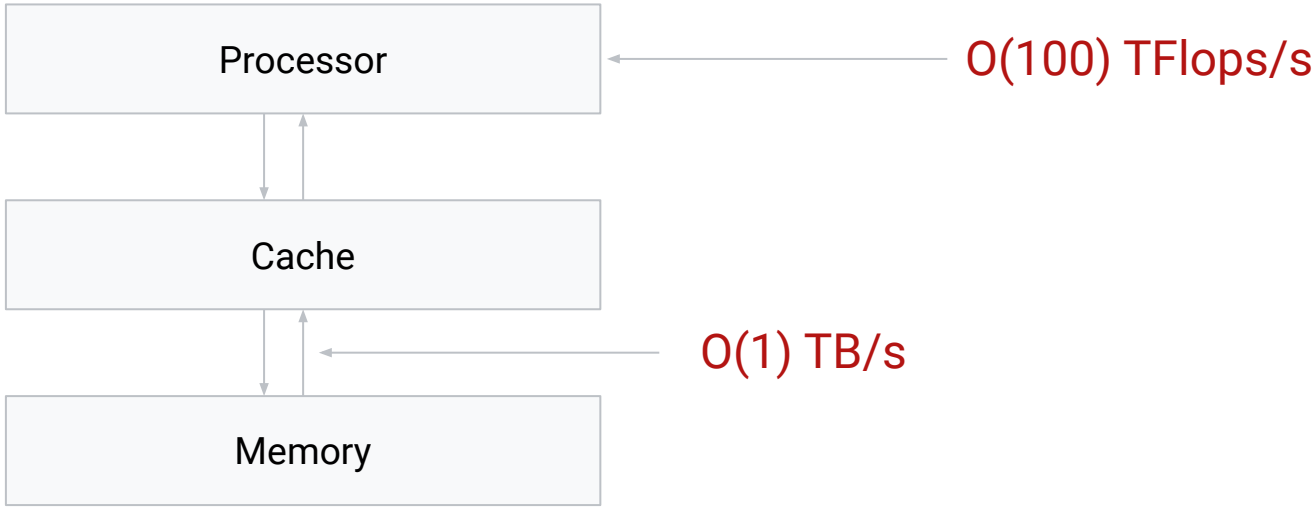
O(10) MB storage ← May store a layer

1024 x 1024 matrix in float = 4 MB

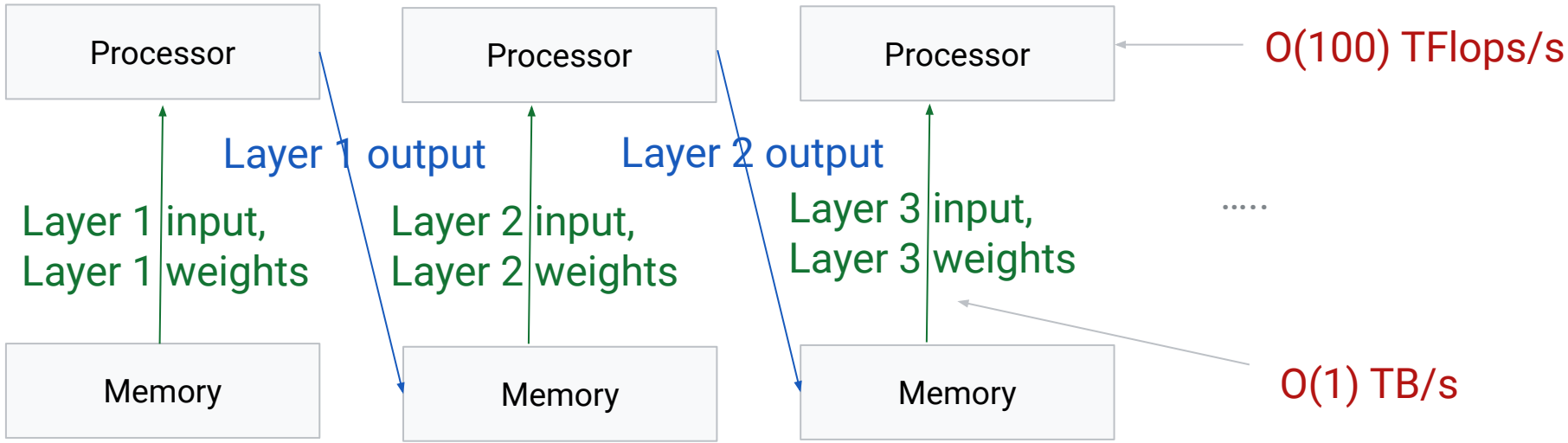
O(10) GB storage ← May store a small LM

1B parameters in float = 4 GB

Computing power and memory bandwidth



One forward pass



For decoding one token from a 1B parameter model:

- Total memory transfer: O(num_parameters + num_activations) = O(1) + o(1) GB
- Total compute= O(1) GFlops

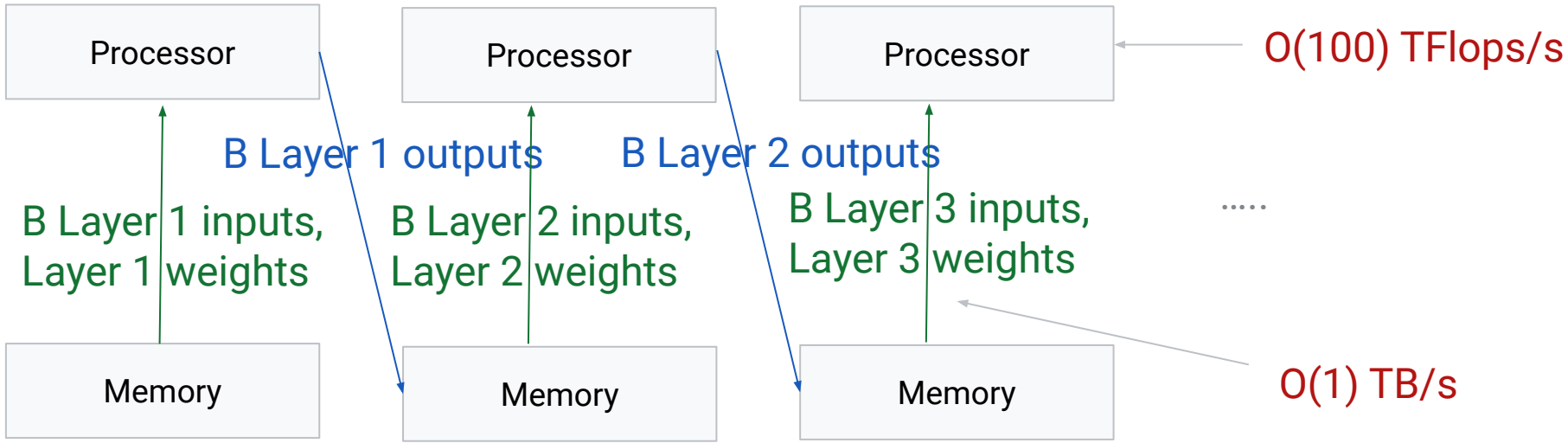
To decode 1000 tokens autoregressively: O(1) TB memory transfer, O(1) TFlops

Efficiency landscape

- Overcome memory bandwidth bottleneck
  - Increase the batch size ← This talk
  - Multihead attention vs multi-query attention
  - Flash attention
  - Compression ← This talk
  - Speculative decoding ← This talk
  - ....

- Reduce the amount of compute
  - Compression
  - ....

One forward pass : batch size B > 1



For decoding one token from a 1B parameter model:

- Total memory transfer: O(num_parameters + activations) = O(1) + o(B) GB
- Total compute= O(B) GFlops

To decode 1000 tokens autoregressively: O(1) + o(B) TB memory transfer, O(B) TFlops

Compression

Compression : the ideal approach

- Assume the parameters come from some distribution
  - The model parameters do not come from any distribution
- Distortion metric = change in model quality
  - Hard to analyze theoretically
- Compute rate distortion function and a coding scheme to achieve it
  - Unclear if compressing parameters and decompressing them is efficient

Compression : current approaches

- Pruning
  - Instead of storing all parameters, store just few important ones

- Quantization
  - Instead of using float32 representation for each parameter, use succinct representation with fewer bits

Magnitude based pruning

- Prune all weights with magnitudes close to zero
  - Finetune the model on non-pruned weights to ensure the quality loss is minimal
- Accelerators are not compute-friendly for pruned models
  - Structured pruning

To prune, or not to prune: exploring the efficacy of pruning for model compression, Zhu and Gupta, 2017

Standard post-training quantization

- Clip the outliers
- Quantize all values to equally placed bins
- 8 bits of quantization

Original tensor

-1.2	-1.1	0.5
-100	0	0.2
1.8	1.6	30

Clip all outliers to [-2, 2]

-1.2	-1.1	0.5
-2	0	0.2
1.8	1.6	2

Quantize to one of five values: -2, -1, 0, 1, 2

-1	-1	1
-2	0	0
2	2	2

ZeroQuant: Efficient and Affordable Post-Training Quantization for Large-Scale Transformers, Yao et al., 2022

A more systematic approach: Optimal brain surgeon

- Parameters to be pruned / quantized: W
- Dataset used to train the model: X
- Loss function that we care about: $L(W, X)$
- Let W^* be a minimum

$$W^* = \arg\min_W L(W, X)$$

- Hence

$$\nabla L(W^*, X) = 0$$

Optimal brain damage, LeCun et al., 1989

Optimal brain surgeon and general network pruning, Hassibi et al., 1993

A more systematic approach: Optimal brain surgeon

- Taylor series expansion around W^*

$$L(W, X) = L(W^*, X) + \frac{1}{2}(W - W^*)H(W - W^*) + O(\|W - W^*\|^3)$$

- Find the best coordinate i to prune

$$\min_W (W - W^*)H(W - W^*) \quad \text{s.t.} \quad e_i^T W = 0$$

- Solution is given by

$$p = \arg\min_p \frac{1}{[H^{-1}]_{p,p}} \cdot (W_p^*)^2$$

- The weights are updated as

$$w \leftarrow w - H^{-1}_{:,p} \frac{1}{[H^{-1}]_{p,p}} \cdot w_p$$

- If the Hessian is identity, this is same as magnitude based pruning

Optimal brain surgeon for pruning and quantization

- The approach is based on Taylor approximation
  - Allows us to select a single matrix row to be modified
  - Need to apply iteratively
- Computing Hessian is complex
  - Apply OBS layer by layer with a simplified loss

$$\min_{\hat{W}_\ell} \| W_\ell X_\ell - \hat{W}_\ell X_\ell \|_2^2$$

- Time complexity of computing Hessian-inverse is $O(d^6)$ for a d x d matrix

Optimal brain compression

Pruning

$$
\begin{aligned}
&M = \{1, \ldots, d_{\text{col}}\} \\
&\textbf{for } i = 1, \ldots, k \textbf{ do} \\
&\quad p \leftarrow \text{argmin}_{p \in M} \frac{1}{[\mathbf{H}^{-1}]_{pp}} \cdot w_p^2 \\
&\quad \mathbf{w} \leftarrow \mathbf{w} - \mathbf{H}^{-1}_{:,p} \frac{1}{[\mathbf{H}^{-1}]_{pp}} \cdot w_p \\
&\quad \mathbf{H}^{-1} \leftarrow \mathbf{H}^{-1} - \frac{1}{[\mathbf{H}^{-1}]_{pp}} \mathbf{H}^{-1}_{:,p} \mathbf{H}^{-1}_{p,:} \\
&\quad M \leftarrow M - \{p\} \\
&\textbf{end for}
\end{aligned}
$$

Quantization

$$
\begin{aligned}
&M = \{1, \ldots, d_{\text{col}}\} \\
&\textbf{for } i = 1, \ldots, k \textbf{ do} \\
&\quad p \leftarrow \text{argmin}_{p \in M} \frac{1}{[\mathbf{H}^{-1}]_{pp}} \cdot (q(w_p) - w_p)^2 \\
&\quad \mathbf{w} \leftarrow \mathbf{w} - \mathbf{H}^{-1}_{:,p} \frac{1}{[\mathbf{H}^{-1}]_{pp}} \cdot (w_p - q(w_p)) \\
&\quad \mathbf{H}^{-1} \leftarrow \mathbf{H}^{-1} - \frac{1}{[\mathbf{H}^{-1}]_{pp}} \mathbf{H}^{-1}_{:,p} \mathbf{H}^{-1}_{p,:} \\
&\quad M \leftarrow M - \{p\} \\
&\textbf{end for}
\end{aligned}
$$

Optimal Brain Compression: A Framework for Accurate Post-Training Quantization and Pruning, Frantar et al., 2023

Recent results on quantization

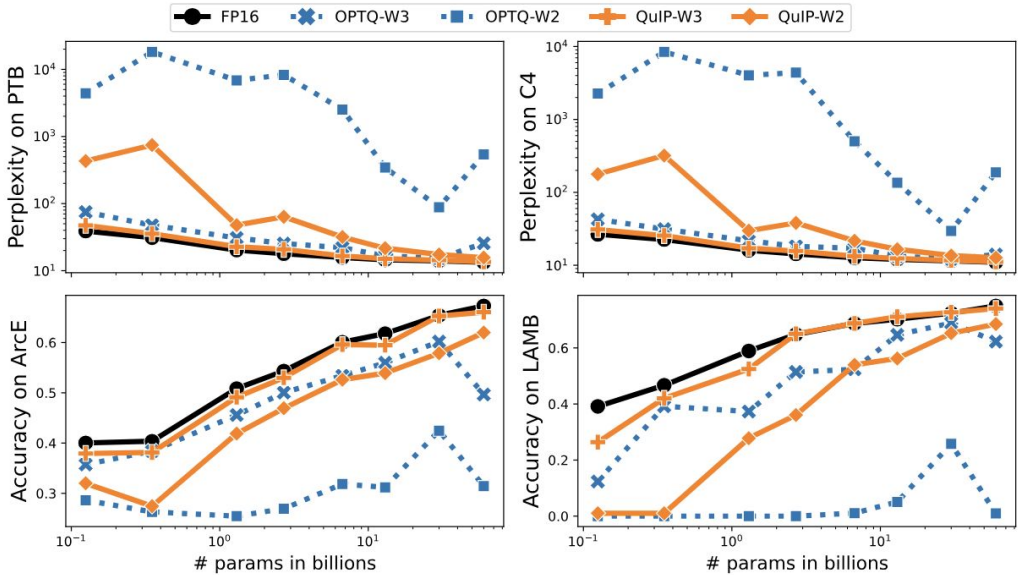


Figure 5: Quantizing OPT models up to 66B parameters. Our method QuIP is the first PTQ procedure to achieve good quantization at 2 bits per weight, across a variety of model sizes and evaluation tasks.

QuIP: 2-Bit Quantization of Large Language Models With Guarantees, Chee et al., 2023

Recent results on pruning

OPT	Sparsity	2.7B	6.7B	13B	30B	66B	175B
Dense	0%	12.47	10.86	10.13	9.56	9.34	8.35
Magnitude	50%	265.	969.	1.2e4	168.	4.2e3	4.3e4
SparseGPT	50%	**13.48**	**11.55**	**11.17**	**9.79**	**9.32**	**8.21**

SparseGPT: Massive Language Models Can be Accurately Pruned in One-Shot, Frantar and Alistarh, 2023

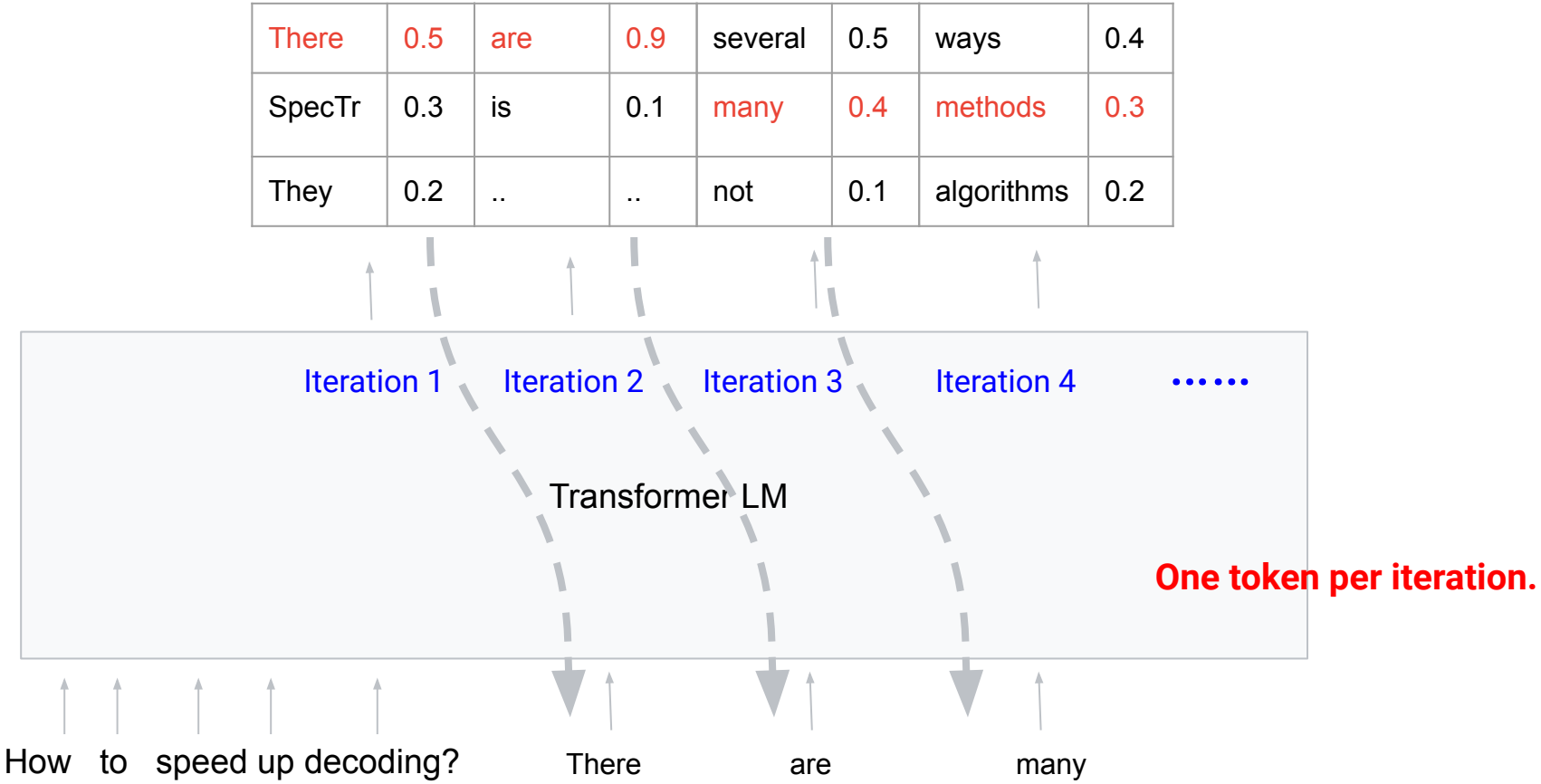
Speculative decoding

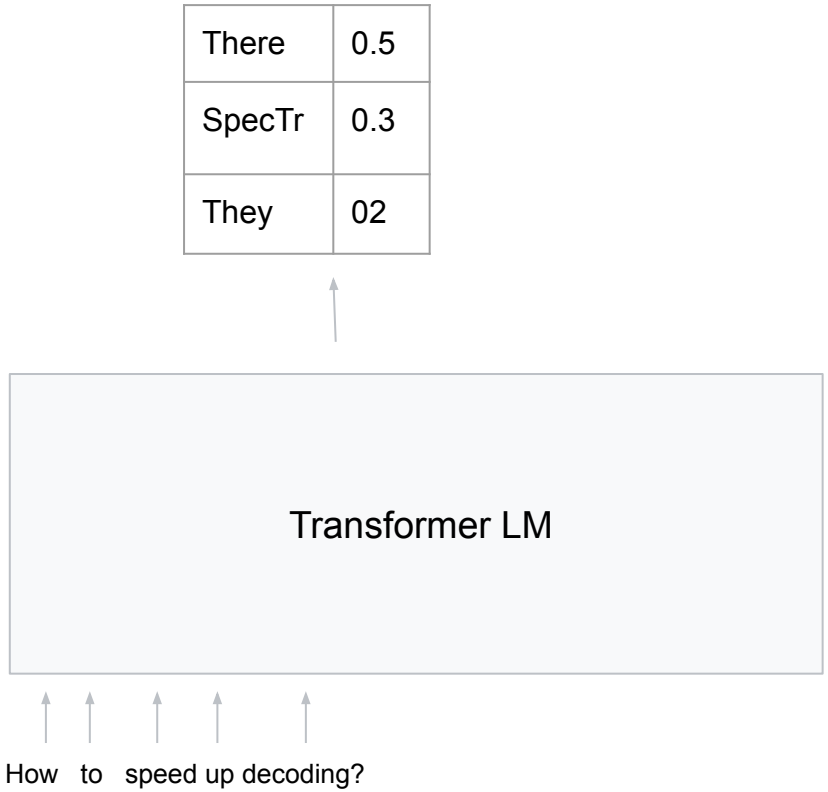
Fast Inference from Transformers via Speculative Decoding, Levianthan et al., 2023

Accelerating Large Language Model Decoding via Speculative Decoding, Chen et al., 2023

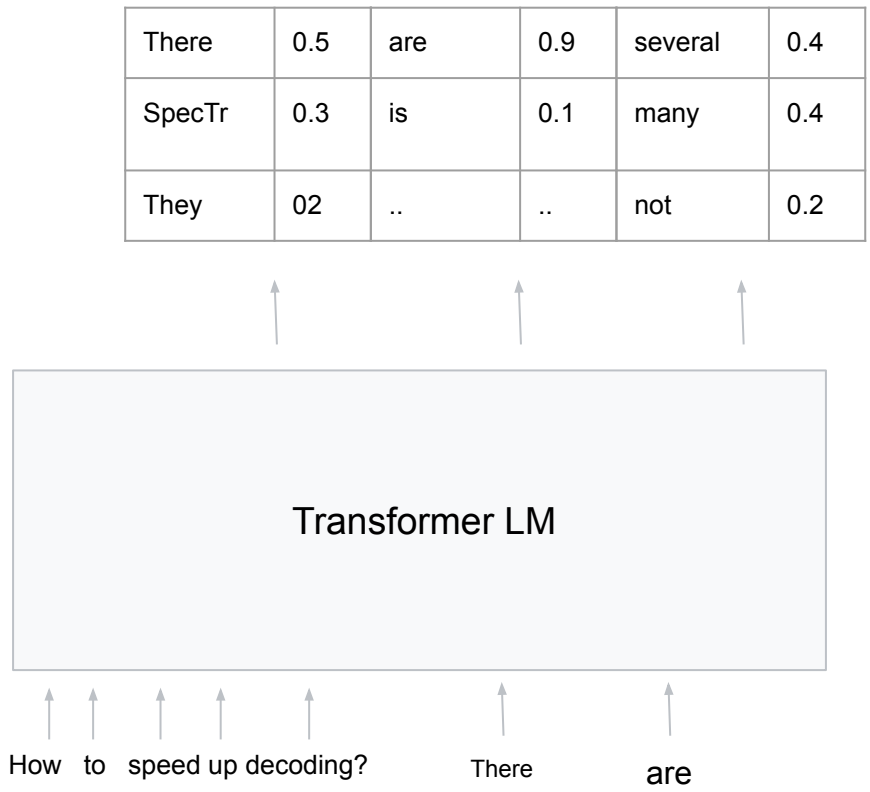
Block Verification Accelerates Speculative Decoding, Sun et al., 2024

Autoregressive language models





Standard inference: Only final prediction



Parallelization along time: predict for every time step

Parallelization along time and batch: predict for every time step and every element of the batch

A simplified computation model

Assumption: in a **memory-bound setting** (inference time is bottlenecked by GPU/TPU memory bandwidth)

Standard inference ≃ parallel inference (time)
≃ parallel inference (batch and time)

Relative latency	batch $= 1$	batch $= 2$	batch $= 4$	batch $= 8$
length $= 4$	1.00 ± 0.16	1.01 ± 0.15	1.06 ± 0.10	1.10 ± 0.16
length $= 8$	1.01 ± 0.18	1.09 ± 0.25	1.10 ± 0.09	1.42 ± 0.4

Goal: improve the speed of generating tokens by exploiting parallelization

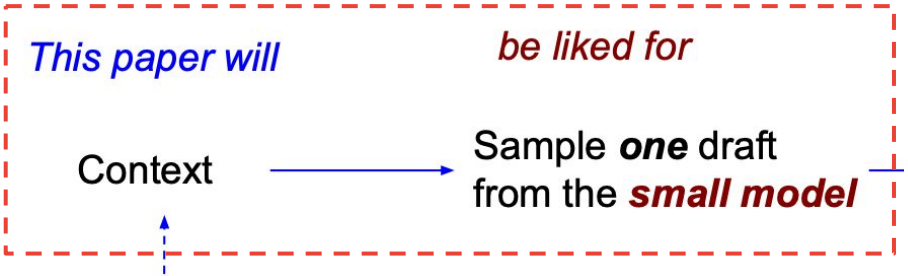
Notations

- The primary model of interest M_b
  - A large transformer model
- A secondary model M_s
  - Any small autoregressive model
- Use the smaller model to generate tokens
- Use the larger model to validate them

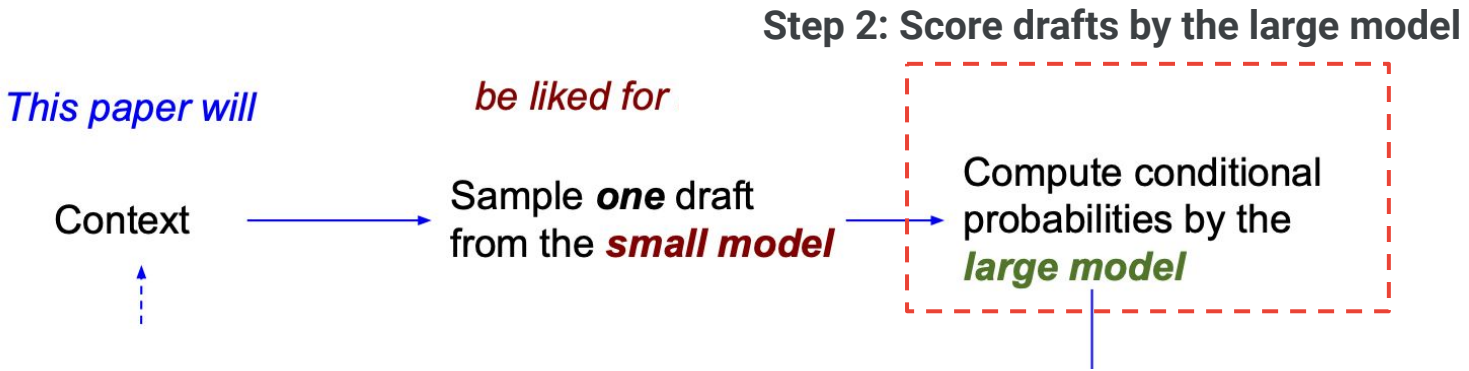
Speculative decoding: one iteration

Recall: small model size << large model size

Step 1: Draft construction

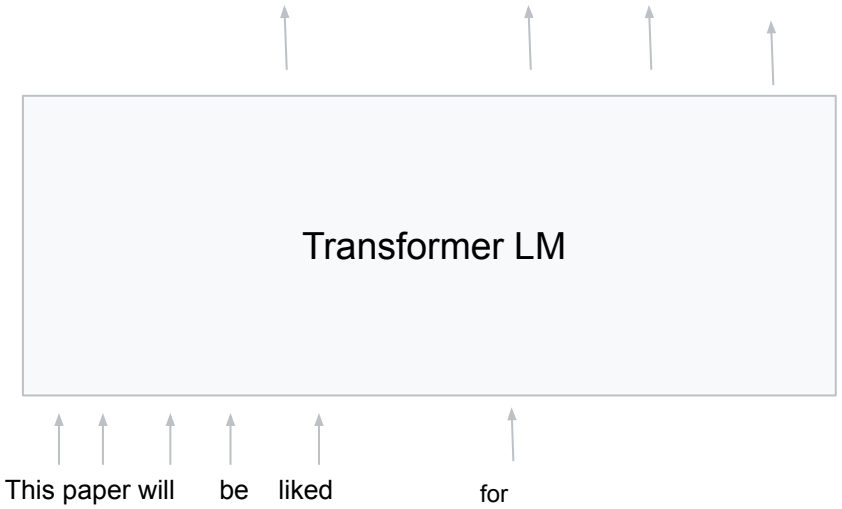


Speculative decoding: one iteration



Step 2: score drafts by the large model

be	0.5	liked	0.8
not	0.3	accepted	0.1
receive	02	rejected	0.2



Parallelization along time: predict for every time step

$$M_b(\cdot \mid \text{This paper will})$$

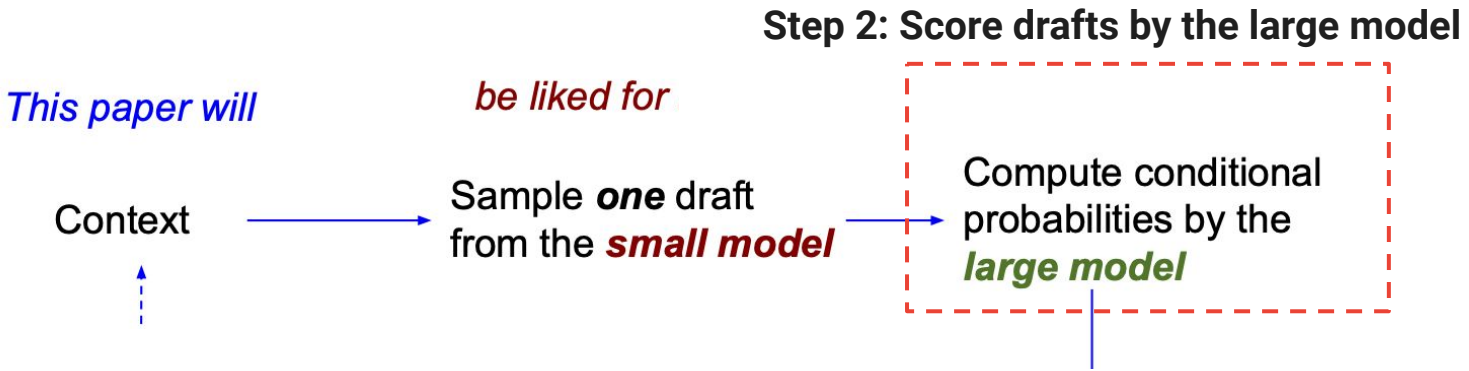
$$M_b(\cdot \mid \text{This paper will be})$$

$$M_b(\cdot \mid \text{This paper will be liked})$$

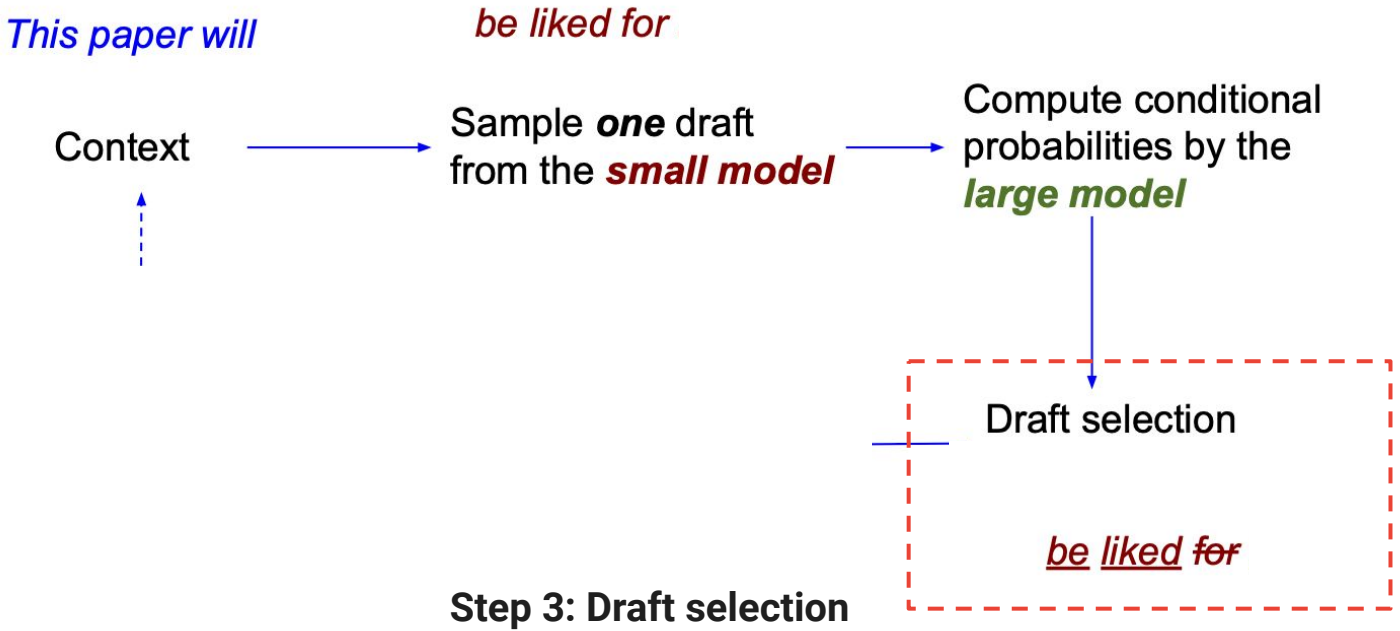
$$M_b(\cdot \mid \text{This paper will be liked for})$$

Speculative decoding: one iteration

Recall: Parallelization along time axis won't increase the latency

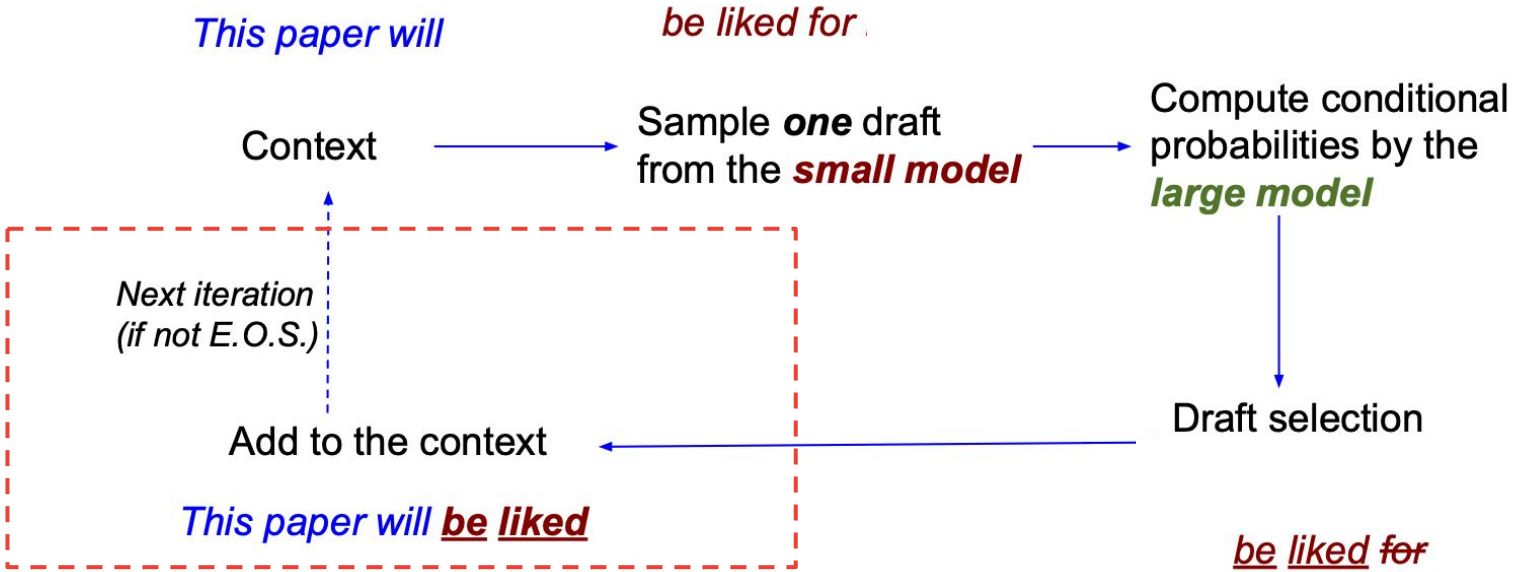


Speculative decoding: one iteration



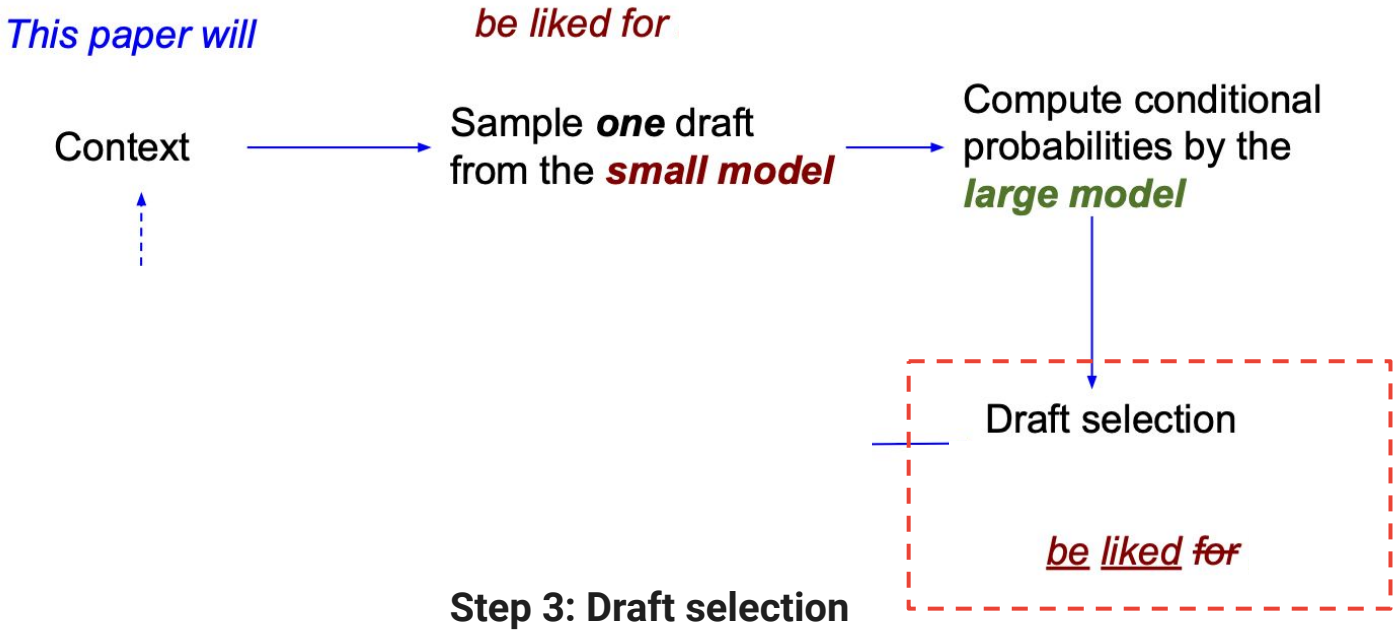
Goal: make sure that the distribution of selected samples follow the large model distribution

Speculative decoding: one iteration



Decoded *two* tokens with *one* call to the large model.

Speculative decoding: one iteration



Goal: make sure that the distribution of selected samples follow the large model distribution

The sampling question

Goal: make sure that the distribution of selected samples follow the large model distribution

- Small model conditional distribution $p = M_s(\cdot \mid x_1, x_2, \ldots, x_n)$
- Large model condition distribution $q = M_b(\cdot \mid x_1, x_2, \ldots, x_n)$
- Given a sample from p : X
- Goal 1: get a sample from such that $Y \sim q$
- Goal 2: maximize speedup i.e maximize the probability of $X = Y$

Attempt 1: independent sampling

Obtain independent sample from q

$$\begin{aligned}\Pr(X = Y) &= \sum_x \Pr(X = Y = x) \\ &= \sum_x \Pr(X = x) \cdot \Pr(Y = x) \\ &= \sum_x p(x) \cdot q(x)\end{aligned}$$

Can we do better?

Attempt 2: towards formalizing

- Suppose we have a joint distribution π over $\mathcal{X} \times \mathcal{X}$ such that

$$\sum_x \pi(x, y) = q(y)$$

$$\sum_y \pi(x, y) = p(x)$$

- A sampling scheme would be, given $p, q, \pi, X \sim p$, sample according to

$$\pi\left(Y \mid X\right)$$

- Question: what is the best joint distribution that maximizes

$$\Pr(X = Y)$$

Attempt 2: maximal coupling

- Suppose we have a joint distribution π over $\mathcal{X} \times \mathcal{X}$ such that

$$\sum_x \pi(x, y) = q(y)$$

$$\sum_y \pi(x, y) = p(x)$$

- Question: what is the best joint distribution π that maximizes $\Pr(X = Y) = \sum_x \pi(x, x)$
- A linear program in π
- Solution:

$$\max_\pi \pi(X = Y) = \sum_x \min(p(x), q(x)) = 1 - d_{TV}(p, q)$$

- Recall: for independent sampling

$$\Pr(X = Y) = \sum_x p(x)q(x) < \sum_x \min(p(x), q(x))$$

Draft selection: token-level maximal coupling

Algorithm 1 Token-level maximal coupling

Input: Distributions p, q, Draft sample $X \sim p$.

1: Compute the *residual* distribution p^{res} where $\forall x \in \Omega, p^{\text{res}}(x) = \frac{q(x) - \min\{p(x), q(x)\}}{1 - \sum_{x'} \min\{p(x'), q(x')\}}$.

2: Sample $\eta \sim U(0, 1)$.

3: **if** $\eta \leq \min\left(1, \frac{q(X)}{p(X)}\right)$ **then**

4: **Return** $Y = X$. {*Accept the draft token.*}

5: **end if**

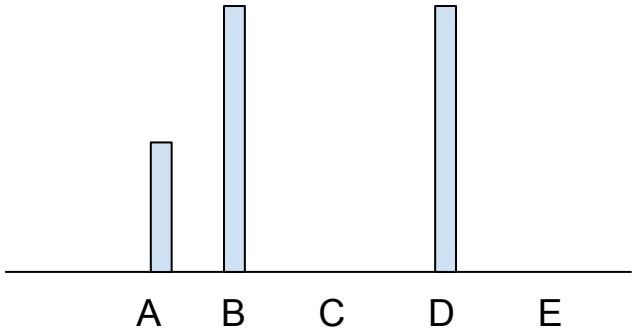
6: **Return** $Y \sim p^{\text{res}}$. {*Sample a corrected token from the residual distribution.*}

$$\Pr(X = Y) = \sum_{x} \min(p(x), q(x))$$

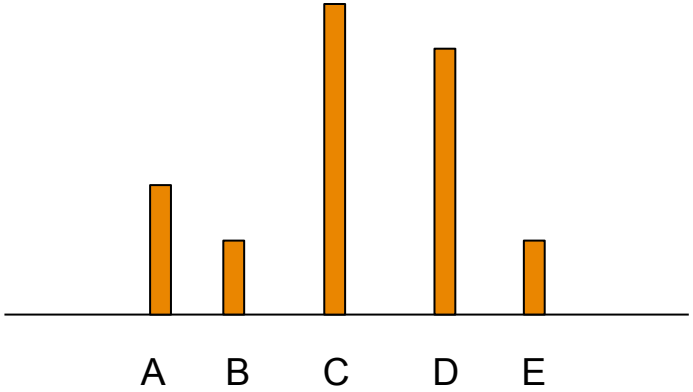
Fast Inference from Transformers via Speculative Decoding, Leviathan et al., 2023

Accelerating Large Language Model Decoding via Speculative Decoding, Chen et al., 2023

Speculative sampling (proof by pictures)

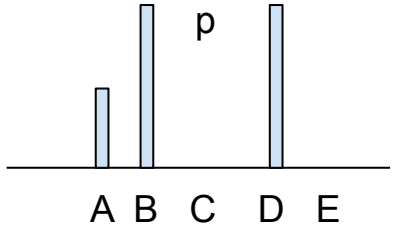


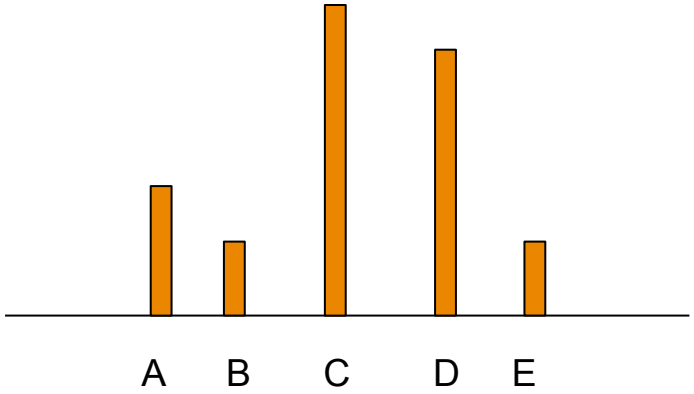
p



q

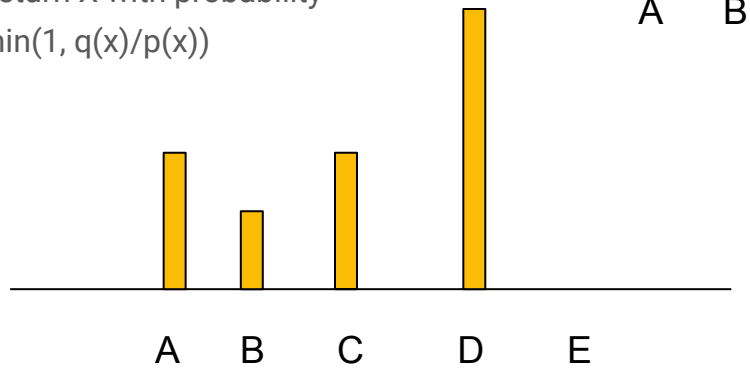
Speculative sampling (proof by pictures)





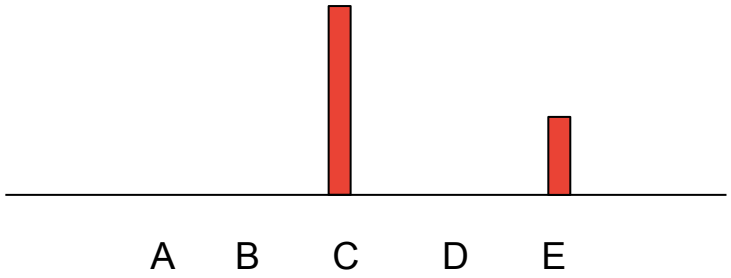
q

Return X with probability
min(1, q(x)/p(x))



+

With remaining probability draw a sample from the residual distribution $(q-p)_+$



min(p, q): conditional distribution of output, if we keep the original sample

$(q-p)_+$: Residual distribution from which we sample if we don't accept draft sample

Speculative sampling (proof sketch)

- Let A denote the event that the sample from p is accepted

$$
\begin{aligned}
\Pr(Y = x) &= \Pr(Y = x, A) + \Pr(Y = x, A^c) \\
&= p(x) \min\left(\frac{q(x)}{p(x)}, 1\right) + \Pr(Y = x, A^c) \\
&= \min\big(q(x), p(x)\big) + \Pr(Y = x, A^c) \\
&= \min\big(q(x), p(x)\big) + \Pr(Y = x \mid A^c)\Pr(A^c) \\
&= \min\big(q(x), p(x)\big) + \frac{\max(q(x) - p(x), 0)}{\sum_z \max(q(z) - p(z), 0)}\Pr(A^c) \\
&= \min\big(q(x), p(x)\big) + \max(q(x) - p(x), 0) \\
&= q(x)
\end{aligned}
$$

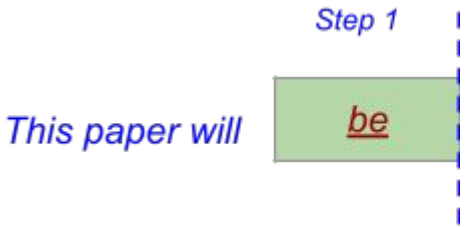
Draft selection: token-level maximal coupling

$$p = M_s(\cdot \mid \text{This paper will}), \quad q = M_b(\cdot \mid \text{This paper will}), \quad X = \text{be}$$

Algorithm 1 Token-level maximal coupling

Input: Distributions p, q, Draft sample $X \sim p$.

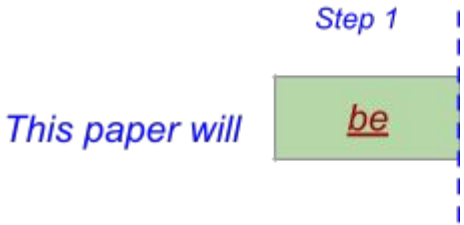
1: Compute the *residual* distribution p^{res} where $\forall x \in \Omega, p^{\text{res}}(x) = \frac{q(x) - \min\{p(x), q(x)\}}{1 - \sum_{x'} \min\{p(x'), q(x')\}}$.
2: Sample $\eta \sim U(0, 1)$.
3: **if** $\eta \leq \min\left(1, \frac{q(X)}{p(X)}\right)$ **then**
4: **Return** $Y = X$. {*Accept the draft token.*}
5: **end if**
6: **Return** $Y \sim p^{\text{res}}$. {*Sample a corrected token from the residual distribution.*}



Draft selection: token-level maximal coupling

$p = M_s(\cdot \mid \text{This paper will}), \quad q = M_b(\cdot \mid \text{This paper will})$

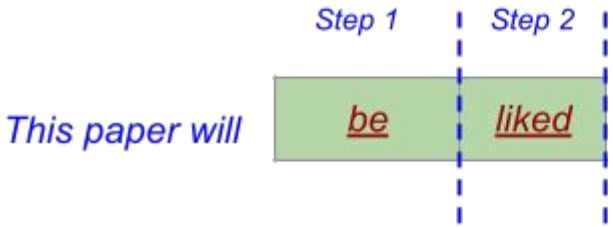
$X = \text{be}$



Draft selection: token-level maximal coupling

$p = M_s(\cdot \mid \text{This paper will be}), \quad q = M_b(\cdot \mid \text{This paper will be})$

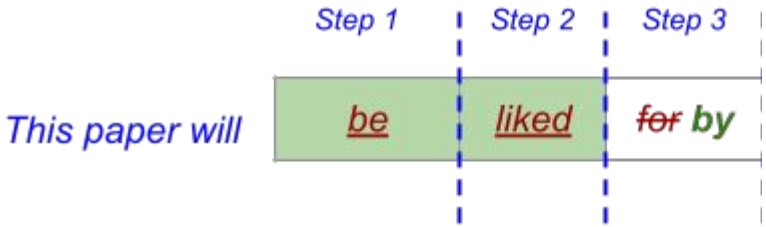
$X = \text{liked}$



Draft selection: token-level maximal coupling

$p = M_s(\cdot \mid \text{This paper will be liked}), \quad q = M_b(\cdot \mid \text{This paper will be liked})$

$X = \text{liked}$



sampled from the residual distribution

Guarantees

- (Validity) The output sequence distribution always follows the large model distribution
- (Speedup) If the small model and large model distributions are similar, decoding is faster

$$\max_{\pi} \pi(X = Y) = \sum_{x} \min(p(x), q(x)) = 1 - d_{TV}(p, q)$$

Lossless acceleration, provably no performance drop.

Optimal in an information theoretic sense

Experiments with speculative decoding

Table 2. Empirical results for speeding up inference from a T5-XXL 11B model.

TASK	M_q	TEMP	γ	α	SPEED
ENDE	T5-SMALL ★	0	7	0.75	**3.4X**
ENDE	T5-BASE	0	7	0.8	2.8X
ENDE	T5-LARGE	0	7	0.82	1.7X
ENDE	T5-SMALL ★	1	7	0.62	**2.6X**
ENDE	T5-BASE	1	5	0.68	2.4X
ENDE	T5-LARGE	1	3	0.71	1.4X
CNNDM	T5-SMALL ★	0	5	0.65	**3.1X**
CNNDM	T5-BASE	0	5	0.73	3.0X
CNNDM	T5-LARGE	0	3	0.74	2.2X
CNNDM	T5-SMALL ★	1	5	0.53	**2.3X**
CNNDM	T5-BASE	1	3	0.55	2.2X
CNNDM	T5-LARGE	1	3	0.56	1.7X

# of tokens guessed

$$\sum_x \min(p(x), q(x))$$

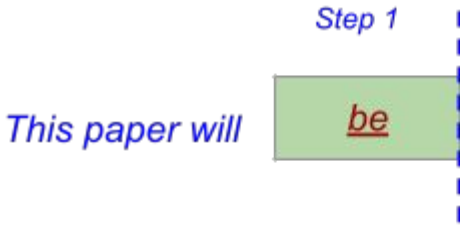
Fast inference from transformers via speculative decoding, Leviathan et al., 2023

Can we improve the speculative decoding algorithm?

Recap: token-level draft selection (k=1)

$p = M_s(\cdot \mid \text{This paper will}), \quad q = M_b(\cdot \mid \text{This paper will})$

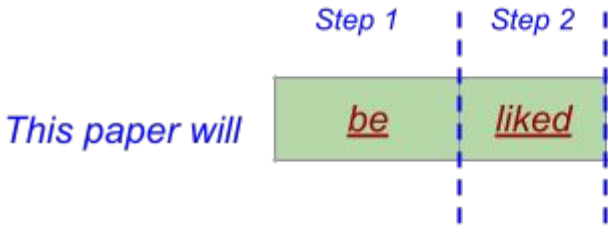
$X = \text{be}$



Recap: token-level draft selection (k=1)

$p = M_s(\cdot \mid \text{This paper will be}), \quad q = M_b(\cdot \mid \text{This paper will be})$

$X = \text{liked}$



So far we have been validating tokens one by one, what if we validate entire block at once?

Improvement over speculative decoding?

Given $X_1, X_2, \ldots, X_L \sim p^L$ generate $Y_1, Y_2, \ldots, Y_L \sim q^L$

$$\beta(x^L, y^L) \triangleq \max_{\ell \leq L} \{ \forall i \leq \ell, x_i = y_i \}$$

Optimal transport problem

$$\max_{\pi} \mathbb{E}_{X^L, Y^L \sim \pi} \left[\beta(X^L, Y^L) \right]$$

Subject to conditions

$$\sum_{y^L} \pi(x^L, y^L) = p^L(x^L), \quad \forall x^L$$

$$\sum_{x^L} \pi(x^L, y^L) = q^L(y^L), \quad \forall y^L$$

Challenges in solving it

We don't know the constraints as it requires knowing $p^L(x^L)\ \forall x^L$ and $q^L(x^L)\ \forall x^L$

$$\sum_{y^L} \pi(x^L, y^L) = p^L(x^L), \quad \forall x^L$$

$$\sum_{x^L} \pi(x^L, y^L) = q^L(y^L), \quad \forall y^L$$

Even if we know these constraints, there are exponential number of them

Result

Theorem 2. *For $i > 0$, let $N(i)$ be the number of decoded tokens after i iterations in Algorithm 3. For any valid draft verification algorithm* VERIFY *in Definition 1, we have $\forall \boldsymbol{c}, \mathcal{M}_s, \mathcal{M}_b, \gamma$, and i,*

$$\mathbb{E}_{\text{BLOCKVERIFY}}[N(i)] \geq \mathbb{E}_{\text{VERIFY}}[N(i)],$$

Algorithm

Algorithm 1 Token Verification

Input: Draft block X^γ; small model distributions $\forall i < \gamma, \mathcal{M}_s(\cdot \mid \boldsymbol{c}, X^i)$; large model distributions $\forall i \leq \gamma, \mathcal{M}_b(\cdot \mid \boldsymbol{c}, X^i)$.

Sample $\eta_1, \ldots, \eta_\gamma \sim U(0,1)$.
Set $\tau = 0$.
for $i = 1, \ldots \gamma$ **do**
  Set $h_i = \min\{\frac{\mathcal{M}_b(X_i|\boldsymbol{c}, X^{i-1})}{\mathcal{M}_s(X_i|\boldsymbol{c}, X^{i-1})}, 1\}$.
  if $\eta_i \leq h_i$ **then**
    Set $\tau = i$.
  else
    break.
  end if
end for
if $\tau = \gamma$ **then**
  Sample Y from $\mathcal{M}_b(\cdot \mid \boldsymbol{c}, X^\gamma)$.
else
  Sample Y from $p_{\text{res}}^{\text{token}}(\cdot \mid \boldsymbol{c}, X^\tau)$ as in Equation (2).
end if
Return X^τ, Y.

Algorithm 2 Block Verification

Input: Draft block X^γ; small model distributions $\forall i < \gamma, \mathcal{M}_s(\cdot \mid \boldsymbol{c}, X^i)$; large model distributions $\forall i \leq \gamma, \mathcal{M}_b(\cdot \mid \boldsymbol{c}, X^i)$.

Sample $\eta_1, \ldots, \eta_\gamma \sim U(0,1)$.
Set $\tau = 0$, $p_0 = 1$.
for $i = 1, \ldots \gamma$ **do**
  Set $p_i = \min\{p_{i-1}\frac{\mathcal{M}_b(X_i|\boldsymbol{c}, X^{i-1})}{\mathcal{M}_s(X_i|\boldsymbol{c}, X^{i-1})}, 1\}$.
  Set h_i as in Equation (4).
  if $\eta_i \leq h_i$ **then**
    Set $\tau = i$.
  else
    continue.
  end if
end for
if $\tau = \gamma$ **then**
  Sample Y from $\mathcal{M}_b(\cdot \mid \boldsymbol{c}, X^\gamma)$.
else
  Sample Y from $p_{\text{res}}^{\text{block}}(\cdot \mid \boldsymbol{c}, X^\tau)$ as in Equation (3).
end if
Return X^τ, Y.

Probability of accepting i consecutive tokens

Acceptance probability in Algorithm 2 (Line 5): $h_\gamma = p_\gamma$, and when $i < \gamma$,

$$h_i = \frac{\sum_{x' \in \mathcal{X}} \max\{p_i \cdot \mathcal{M}_b(x' \mid \boldsymbol{c}, X^i) - \mathcal{M}_s(x' \mid \boldsymbol{c}, X^i), 0\}}{\sum_{x' \in \mathcal{X}} \max\{p_i \cdot \mathcal{M}_b(x' \mid \boldsymbol{c}, X^i) - \mathcal{M}_s(x' \mid \boldsymbol{c}, X^i), 0\} + 1 - p_i}. \quad (4)$$

Experiments

Dataset	Block efficiency			Wall clock time speedup over baseline		
	TokenV	BlockV	Improve. ↑ %	TokenV	BlockV	Improve. ↑ %
LM1B	3.21 ± 0.01	3.49 ± 0.02	8.68 ± 0.79	2.17 ± 0.01	2.32 ± 0.01	6.85 ± 0.74
GPT Prompt	3.41 ± 0.04	3.76 ± 0.02	10.06 ± 1.66	2.30 ± 0.02	2.48 ± 0.01	8.14 ± 1.55
WebQA	3.44 ± 0.01	3.70 ± 0.01	7.53 ± 0.24	2.32 ± 0.00	2.45 ± 0.01	5.75 ± 0.22
PIQA	3.40 ± 0.02	3.68 ± 0.00	8.30 ± 0.62	2.29 ± 0.01	2.44 ± 0.00	6.52 ± 0.58
ShareGPT	3.34 ± 0.01	3.62 ± 0.03	8.45 ± 0.98	2.25 ± 0.01	2.40 ± 0.02	6.68 ± 0.91
XSum	3.49 ± 0.02	3.76 ± 0.01	7.63 ± 0.94	2.35 ± 0.01	2.49 ± 0.01	5.82 ± 0.88
GSM8K	3.81 ± 0.01	4.15 ± 0.03	8.74 ± 0.56	2.55 ± 0.01	2.73 ± 0.02	6.84 ± 0.51
WMT-DeEn	3.19 ± 0.01	3.41 ± 0.02	7.00 ± 0.78	2.15 ± 0.01	2.27 ± 0.01	5.36 ± 0.73
Average	3.41	3.70	8.30	2.30	2.45	6.49

Metrics:

- **Block efficiency**: average number of decoded tokens per iteration (*ideal speedup*).
- **Wall clock speedup**: includes the time for drafting samples, the increase in scoring time due to parallelization, and other overheads.

Summary

- Background on language models
- Language model inference
- User-intended language model inference
- Efficient language model inference

Thanks!

Acknowledgements:
We thank Jonathan Berant, Jacob Eisenstein, Brian Roark, and Ziteng Sun for feedback and helpful pointers.