CSE 493s/599s
Lecture 4. Sampling and in-context learning

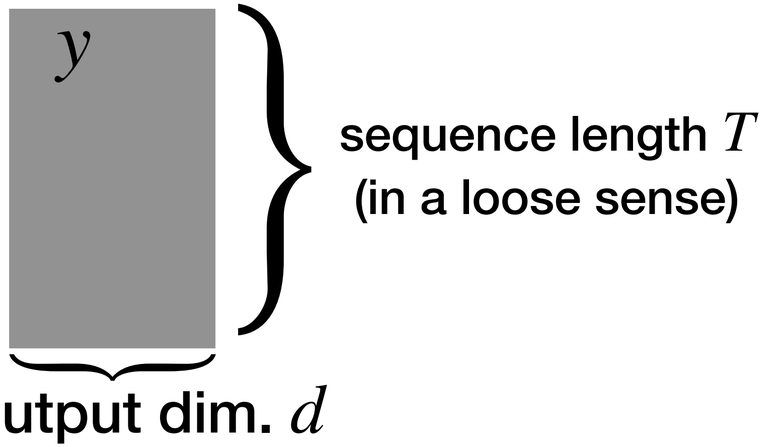
Sewoong Oh

Outline

- Language models
- General LLM framework
  - Token processing
  - Sequence mixing
  - Prediction

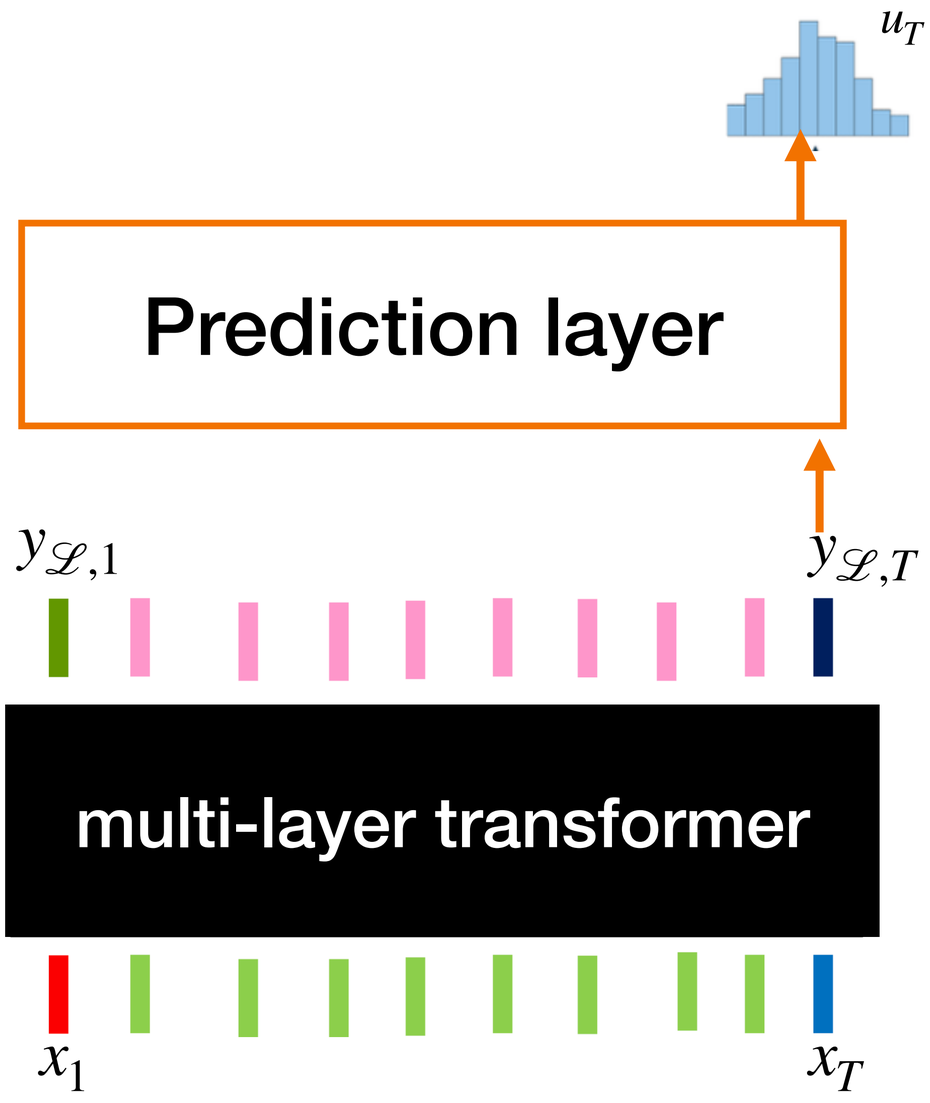
Prediction

- Next token prediction involve outputting a distribution over the vocab and sampling from the distribution.



- The last transformer block outputs the T embeddings each d dimension: $y_{\mathscr{L}} \in \mathbb{R}^{T \times d}$.

- The prediction layer takes the output representation of the last word, $y_{\mathscr{L},T}$, to predict the distribution u_T of the next token w_{T+1},

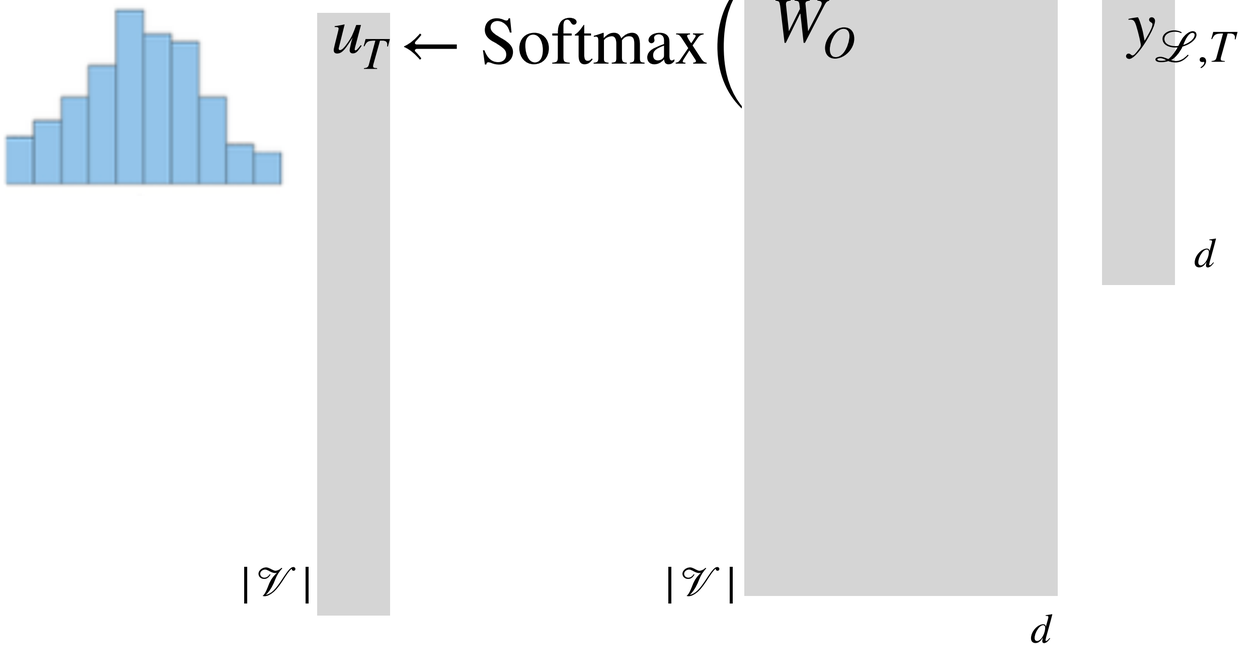


- The last transformer block outputs the T embeddings each d dimension: $y_{\mathscr{L}} \in \mathbb{R}^{T \times d}$.

- The prediction layer takes the output representation of the last word, $y_{\mathscr{L},T}$, to predict the next token, with a **learnable parameter** $W_O \in \mathbb{R}^{|\mathscr{V}| \times d}$, which may or may not be sharing weights with the input token embedding matrix.

$$u_T \leftarrow \text{Softmax}(W_O y_{\mathscr{L},T})$$

$$u_{T,i} = \mathbb{P}(i\text{-th token}) = \frac{e^{W_{O,i} y_{\mathscr{L},t}}}{\sum_{j=1}^{|\mathscr{V}|} e^{W_{O,j} y_{\mathscr{L},t}}}$$

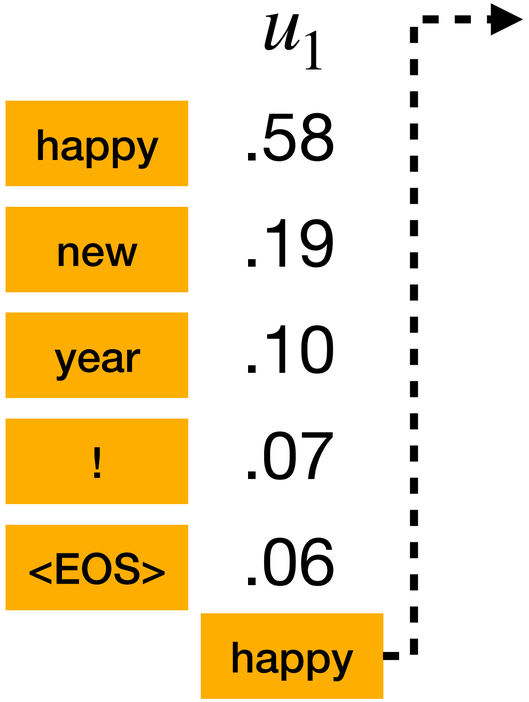


- Given the (conditional) token distribution $\{u_t\}_{t=1}^{T}$, there are many ways to sample tokens, auto-regressively.

- Auto-regressive sampling uses the chain rule to break the distribution on the sentence into:

$$\mathbb{P}(S) = \mathbb{P}(w_{1:T}) = \mathbb{P}(w_1)\mathbb{P}(w_2 \mid w_1)\mathbb{P}(w_3 \mid w_1, w_2)\cdots\mathbb{P}(w_T \mid w_{1:T-1})$$

- **Random sampling** from this conditional distribution generates token-by-token from the distribution u_t, each time until <EOS>.

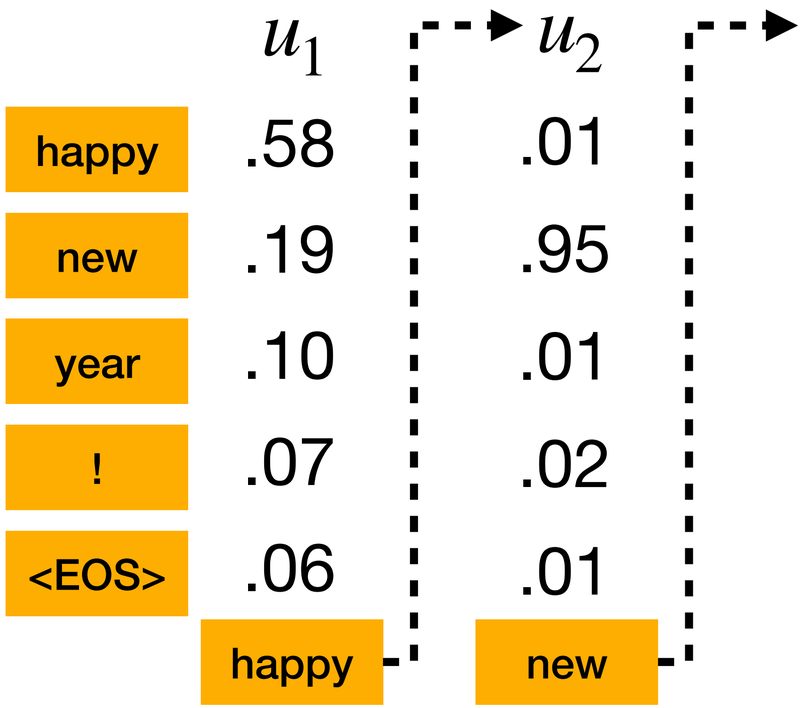


- Given the (conditional) token distribution $\{u_t\}_{t=1}^{T}$, there are many ways to sample tokens, auto-regressively.

- Auto-regressive sampling uses the chain rule to break the distribution on the sentence into:

$$\mathbb{P}(S) = \mathbb{P}(w_{1:T}) = \mathbb{P}(w_1)\mathbb{P}(w_2 \mid w_1)\mathbb{P}(w_3 \mid w_1, w_2)\cdots\mathbb{P}(w_T \mid w_{1:T-1})$$

- **Random sampling** from this conditional distribution generates token-by-token from the distribution u_t, each time until <EOS>.

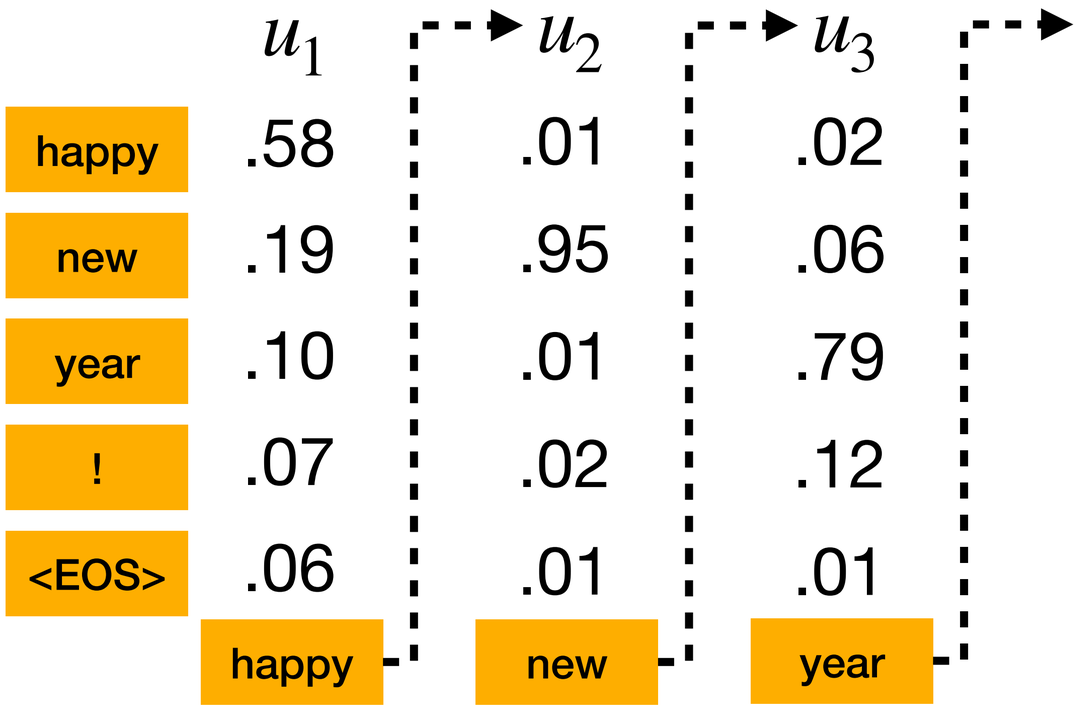


- Given the (conditional) token distribution $\{u_t\}_{t=1}^{T}$, there are many ways to sample tokens, auto-regressively.

- Auto-regressive sampling uses the chain rule to break the distribution on the sentence into:

$$\mathbb{P}(S) = \mathbb{P}(w_{1:T}) = \mathbb{P}(w_1)\mathbb{P}(w_2 \mid w_1)\mathbb{P}(w_3 \mid w_1, w_2)\cdots\mathbb{P}(w_T \mid w_{1:T-1})$$

- **Random sampling** from this conditional distribution generates token-by-token from the distribution u_t, each time until <EOS>.

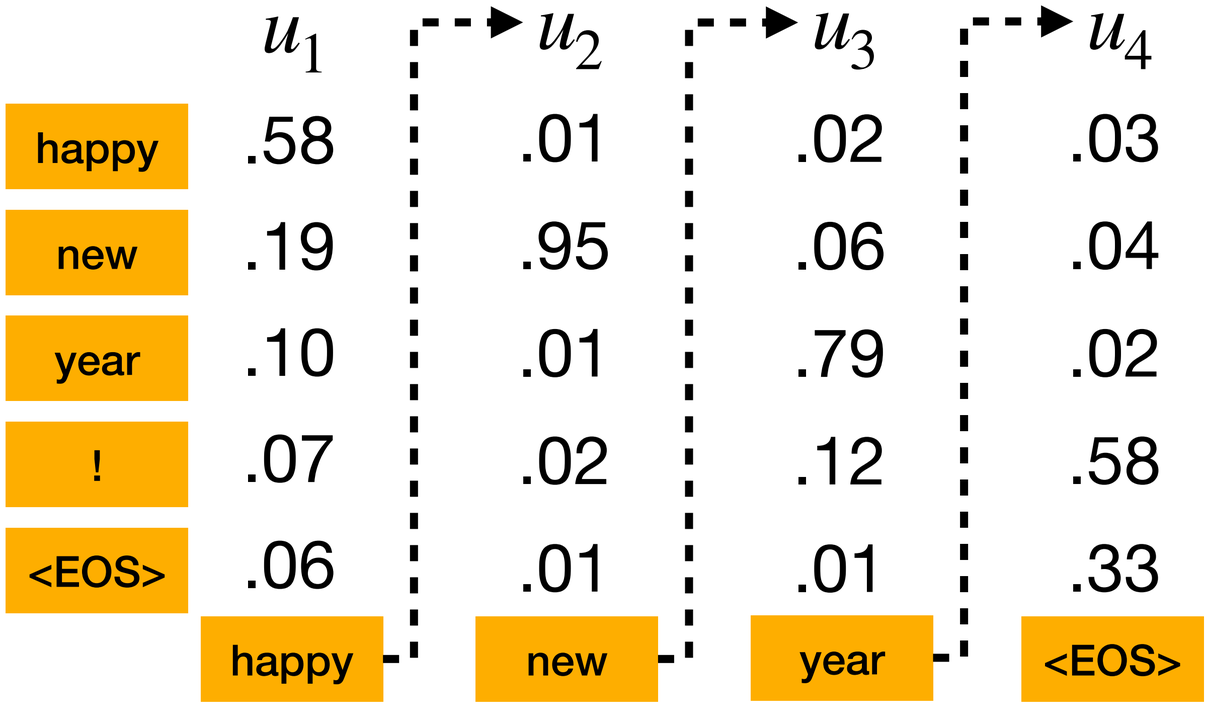


- Given the (conditional) token distribution $\{u_t\}_{t=1}^{T}$, there are many ways to sample tokens, auto-regressively.

- Auto-regressive sampling uses the chain rule to break the distribution on the sentence into:

$$\mathbb{P}(S) = \mathbb{P}(w_{1:T}) = \mathbb{P}(w_1)\mathbb{P}(w_2 \mid w_1)\mathbb{P}(w_3 \mid w_1, w_2)\cdots\mathbb{P}(w_T \mid w_{1:T-1})$$

- **Random sampling** from this conditional distribution generates token-by-token from the distribution u_t, each time until <EOS>.



- What could go wrong?

- Given the (conditional) token distribution $\{u_t\}_{t=1}^{T}$, there are many ways to sample tokens, auto-regressively.

- Auto-regressive sampling uses the chain rule to break the distribution on the sentence into:

$$\mathbb{P}(S) = \mathbb{P}(w_{1:T}) = \mathbb{P}(w_1)\mathbb{P}(w_2 \mid w_1)\mathbb{P}(w_3 \mid w_1, w_2)\cdots\mathbb{P}(w_T \mid w_{1:T-1})$$

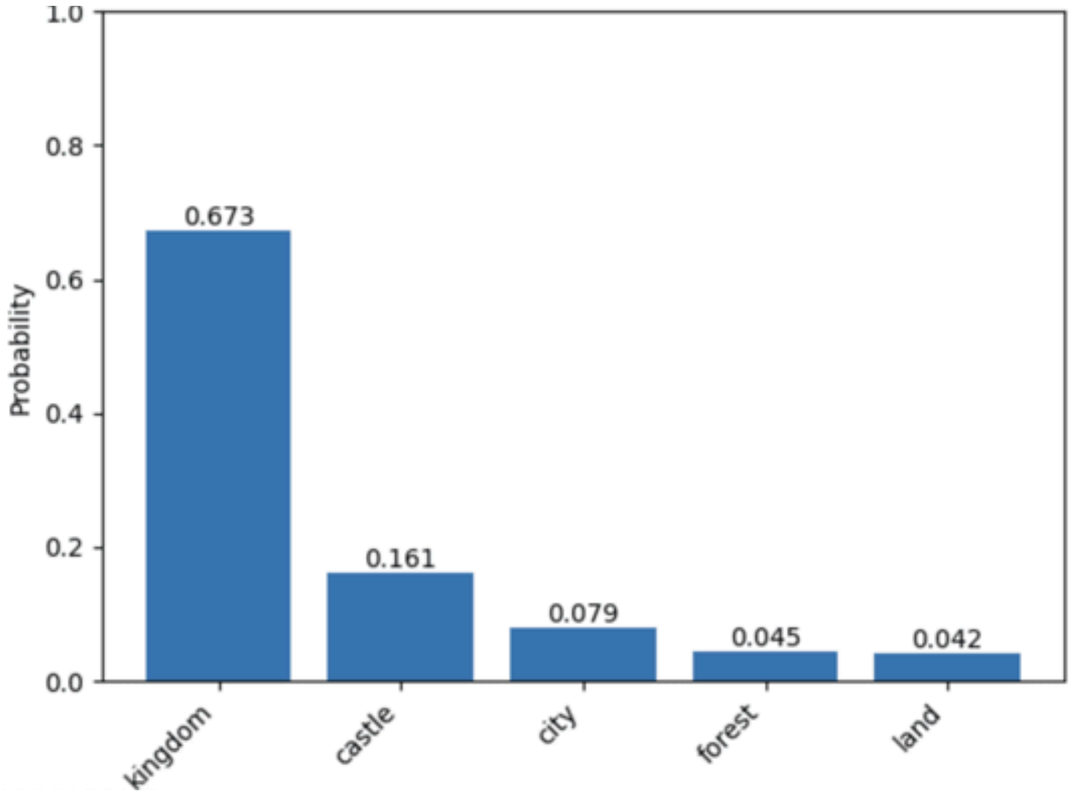
- **Random sampling** from this conditional distribution generates token-by-token from the distribution u_t, each time until <EOS>.

    - **Benefit**: if the LM predicts an accurate and **calibrated *conditional distribution***, then random sampling auto-regressively provides an exact sample from the ***joint distribution*** on the sentence.

    - **Weaknesses**:

        - can generate out-of-distribution samples, resulting in **hallucination** and non-grammatical sentences.
        - LMs distributions are **not well calibrated** since they are trained as classifiers with cross-entropy.

- Given the token distribution, there are many ways to sample tokens, auto-regressively.
  - **Random sampling** samples a token from the distribution u_t, each time until <EOS>, but can generate out-of-distribution samples, resulting in hallucination and non-grammatical sentences.
  - On the other extreme is **Greedy sampling,** which samples the highest probability token, deterministically. Can get stuck repeating highly likely phrases.



Beam Search (Greedy sampling)

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

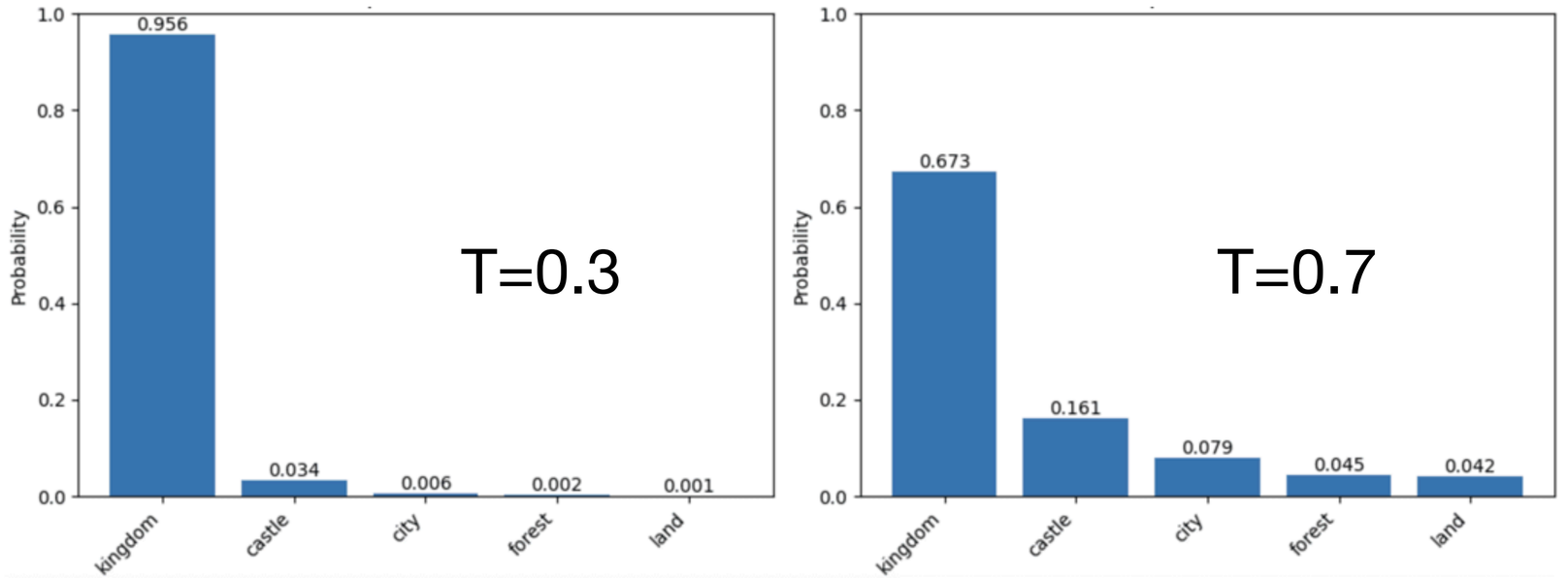
Random Sampling

They were cattle called Bolivian Cavalleros; they live in a remote desert uninterrupted by town, and they speak huge, beautiful, paradisiacal Bolivian linguistic thing. They say, 'Lunch, marge.' They don't tell what the lunch is," director Professor Chuperas Omwell told Sky News. "They've only been talking to scientists, like we're being interviewed by TV reporters. We don't even stick around to be interviewed by TV reporters. Maybe that's how they figured out that they're cosplaying as the Bolivian Cavalleros."

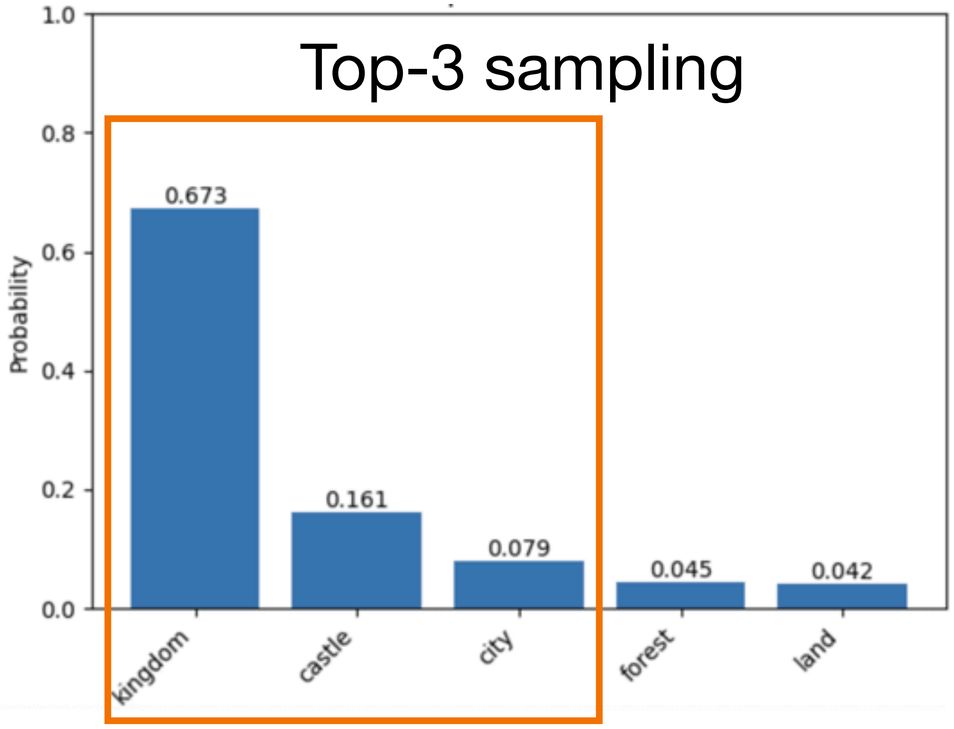
- Given the token distribution, there are many ways to sample tokens, auto-regressively.
  - In practice, one balances the two ends by **Sampling with temperature** T, usually between 0 and 1, which samples from an adjusted Softmax$\left((1/T)\, W_O y_{\mathscr{L},t} \right)$:

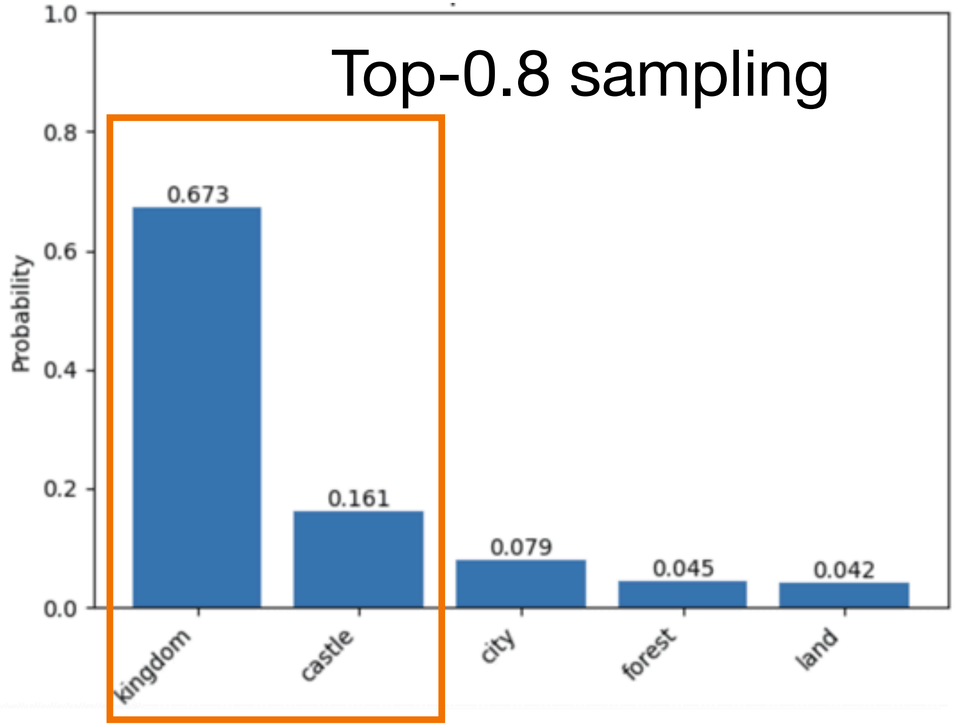
$$u_{t,i} = \mathbb{P}(i\text{-th token}) = \frac{e^{\frac{W_{O,i} y_{\mathscr{L},t}}{T}}}{\sum_{j=1}^{|\mathscr{V}|} e^{\frac{W_{O,j} y_{\mathscr{L},t}}{T}}},$$

where $T = 1$ recovers the original random sampling,
T=0 recovers the greedy sampling,
0<T<1 balances the two. T is tuned like a hyper-parameter at inference time.

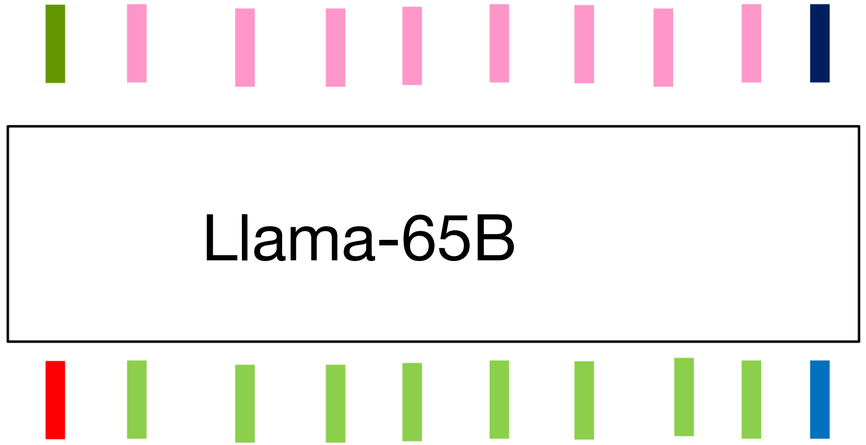


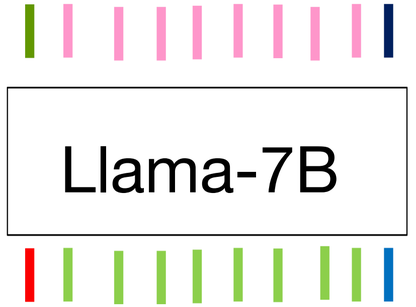
- Given the token distribution, there are many ways to sample tokens, auto-regressively.
  - **Top-k sampling** samples from u_t but only among the top-k tokens.
    - Some top-3 are very likely, some are not, so performance varies.
  - **Nucleus sampling** (also known as **top-p sampling**) [Holtzman et al. 2020] samples from u_t but only among the smallest set whose cumulative probability exceeds $p \in (0,1)$
    - In practice, **Nucleus sampling** strikes the right balance and usually performs the best.



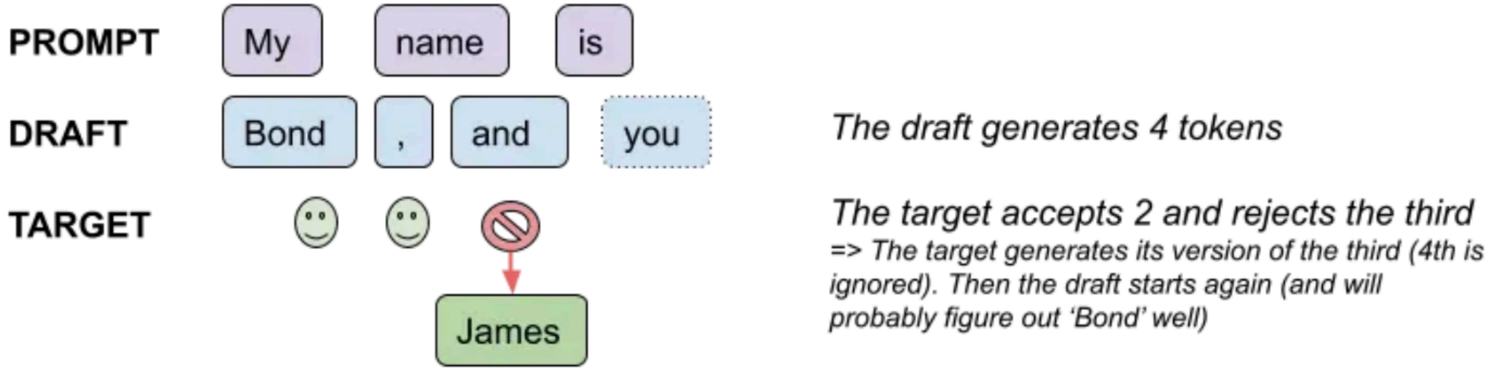


- A major issue in LLMs: **auto-regressive sampling** is sequential and slow.
- **Speculative decoding** [Chen et al. 2024] is inspired from the hypothesis: maybe LLMs can write faster with a help of smaller language models.
- Suppose you have two models of differing sizes, e.g., Llama-7B and Llama-65B, and a sampling scheme of choice, e.g., random with temperature T.
  - How would you speed up sampling?
  - Hint: it is motivated by how people use LLMs.

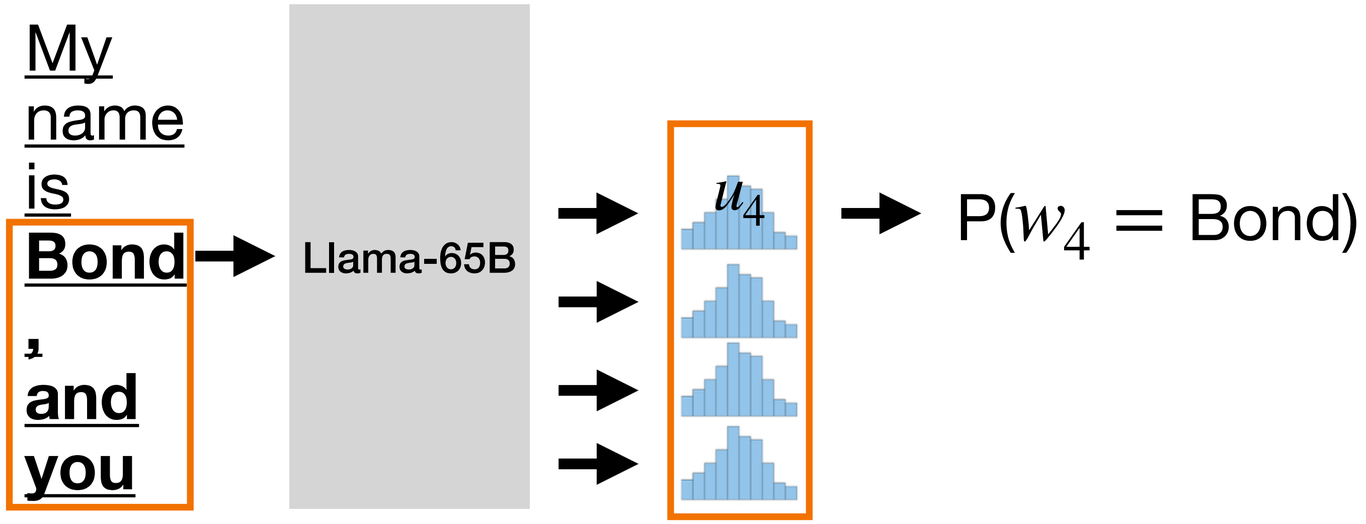




- **Speculative decoding** [Chen et al. 2024] performs at each step,
  - use ***small model*** to generate the next 4 tokens, e.g., “Bond , and you”

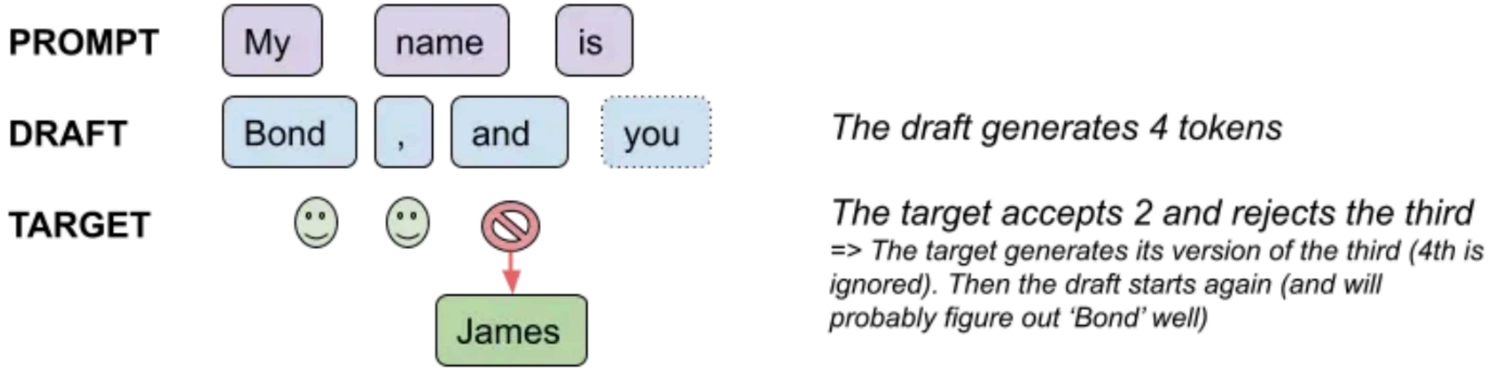


  - pass all [prompt, generated 4 tokens]=“ My name is **Bond, and you** ” to the ***large*** model, and produce the probability distribution of the all 4 tokens,

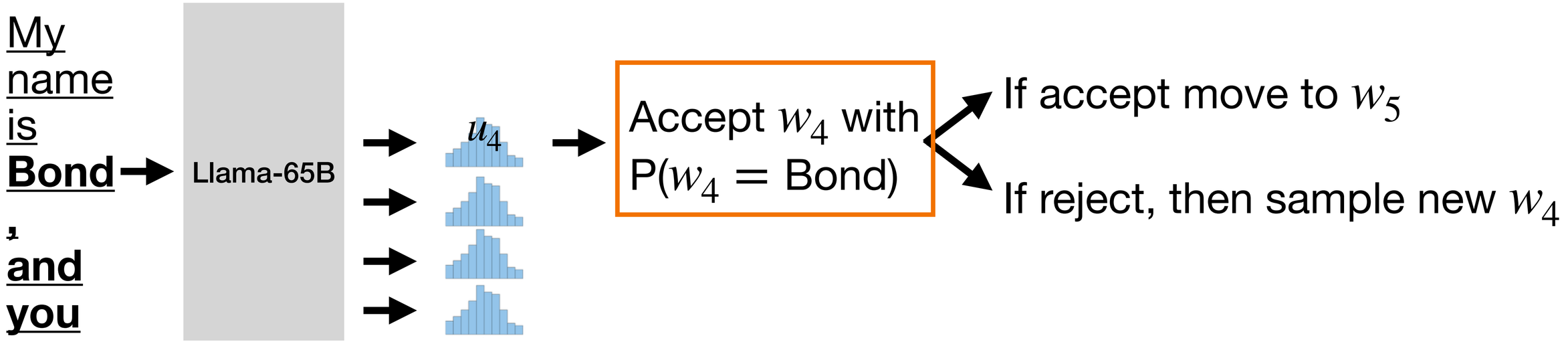


What do we do with it?

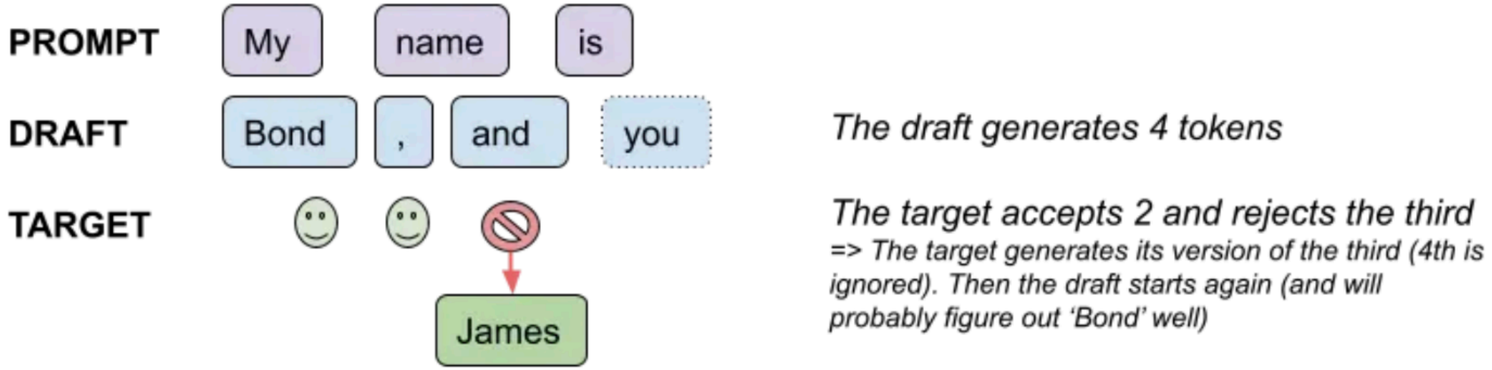
- **Speculative decoding** [Chen et al. 2024] performs at each step,
  - use ***small*** model to generate the next 4 tokens, e.g., “Bond , and you”



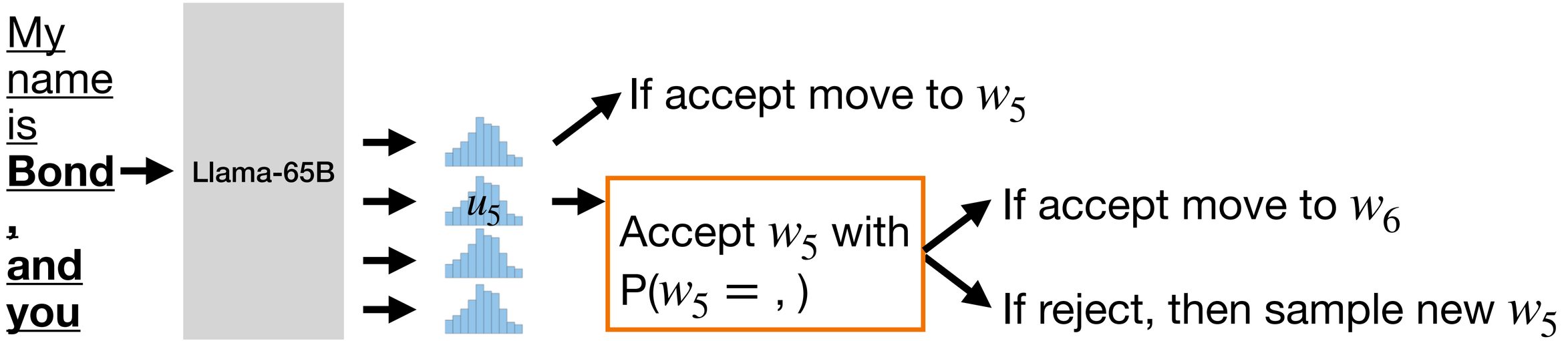
  - pass all [prompt, generated 4 tokens]=“ My name is **Bond, and you** ” to the ***large*** model, and produce the probability distribution of the all 4 tokens,



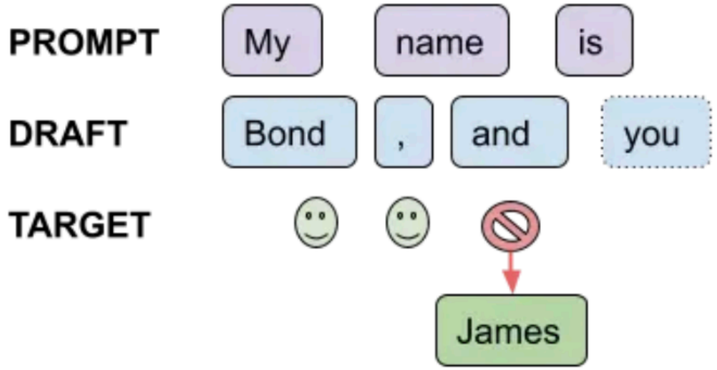
- **Speculative decoding** [Chen et al. 2024] performs at each step,
  - use ***small*** model to generate the next 4 tokens, e.g., “Bond , and you”



  - pass all [prompt, generated 4 tokens]=“ My name is **Bond, and you** ” to the ***large*** model, and produce the probability distribution of the all 4 tokens,



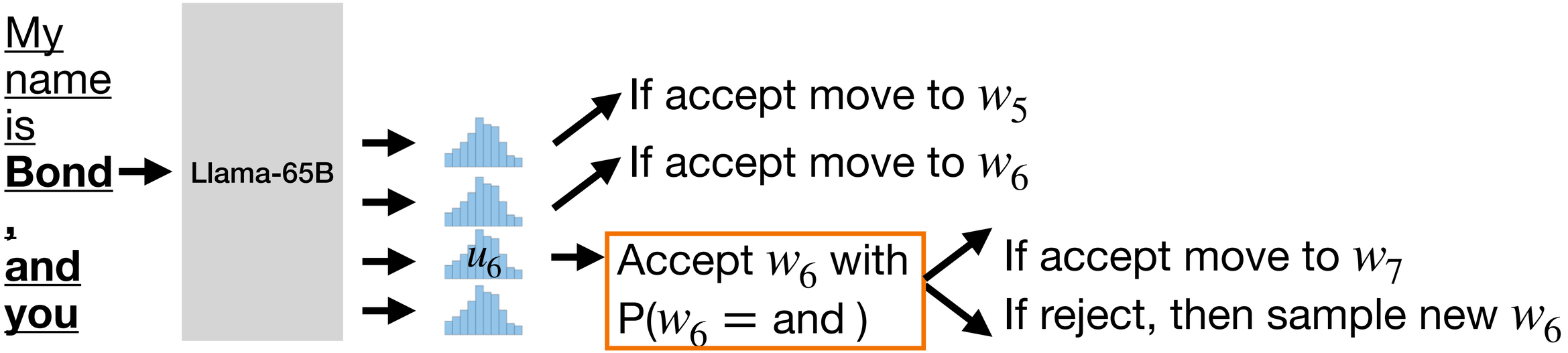
- **Speculative decoding** [Chen et al. 2024] performs at each step,
  - use ***small*** model to generate the next 4 tokens, e.g., “Bond , and you”



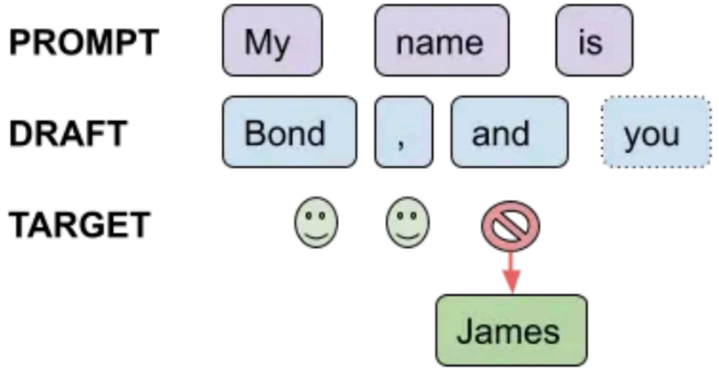
The draft generates 4 tokens

The target accepts 2 and rejects the third
=> The target generates its version of the third (4th is ignored). Then the draft starts again (and will probably figure out ‘Bond’ well)

  - pass all [prompt, generated 4 tokens]=“ My name is **Bond, and you** ” to the ***large*** model, and produce the probability distribution of the all 4 tokens,



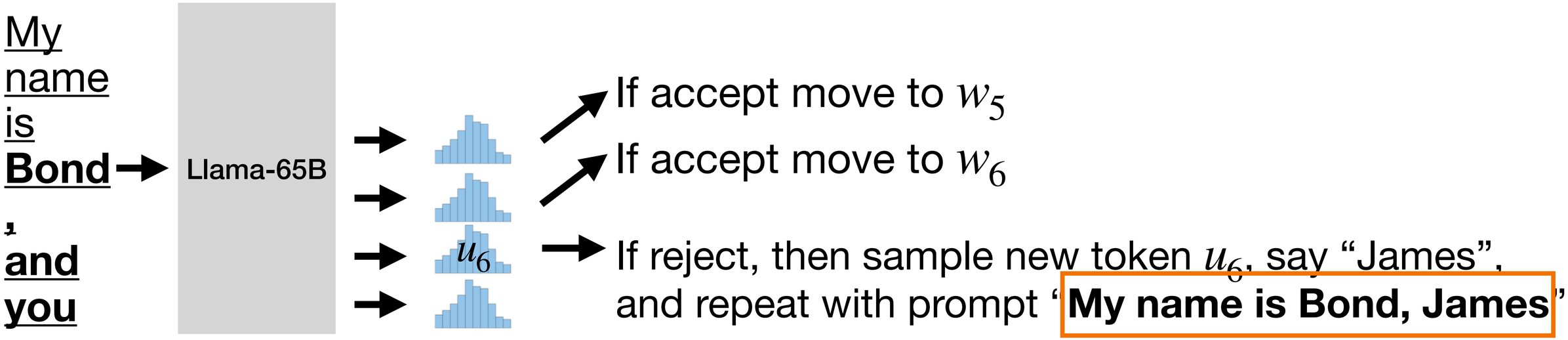
- **Speculative decoding** [Chen et al. 2024] performs at each step,
  - use ***small*** model to generate the next 4 tokens, e.g., “Bond , and you”



The draft generates 4 tokens

The target accepts 2 and rejects the third
=> The target generates its version of the third (4th is ignored). Then the draft starts again (and will probably figure out ‘Bond’ well)

  - pass all [prompt, generated 4 tokens]=“ My name is **Bond, and you** ” to the ***large*** model, and produce the probability distribution of the all 4 tokens,



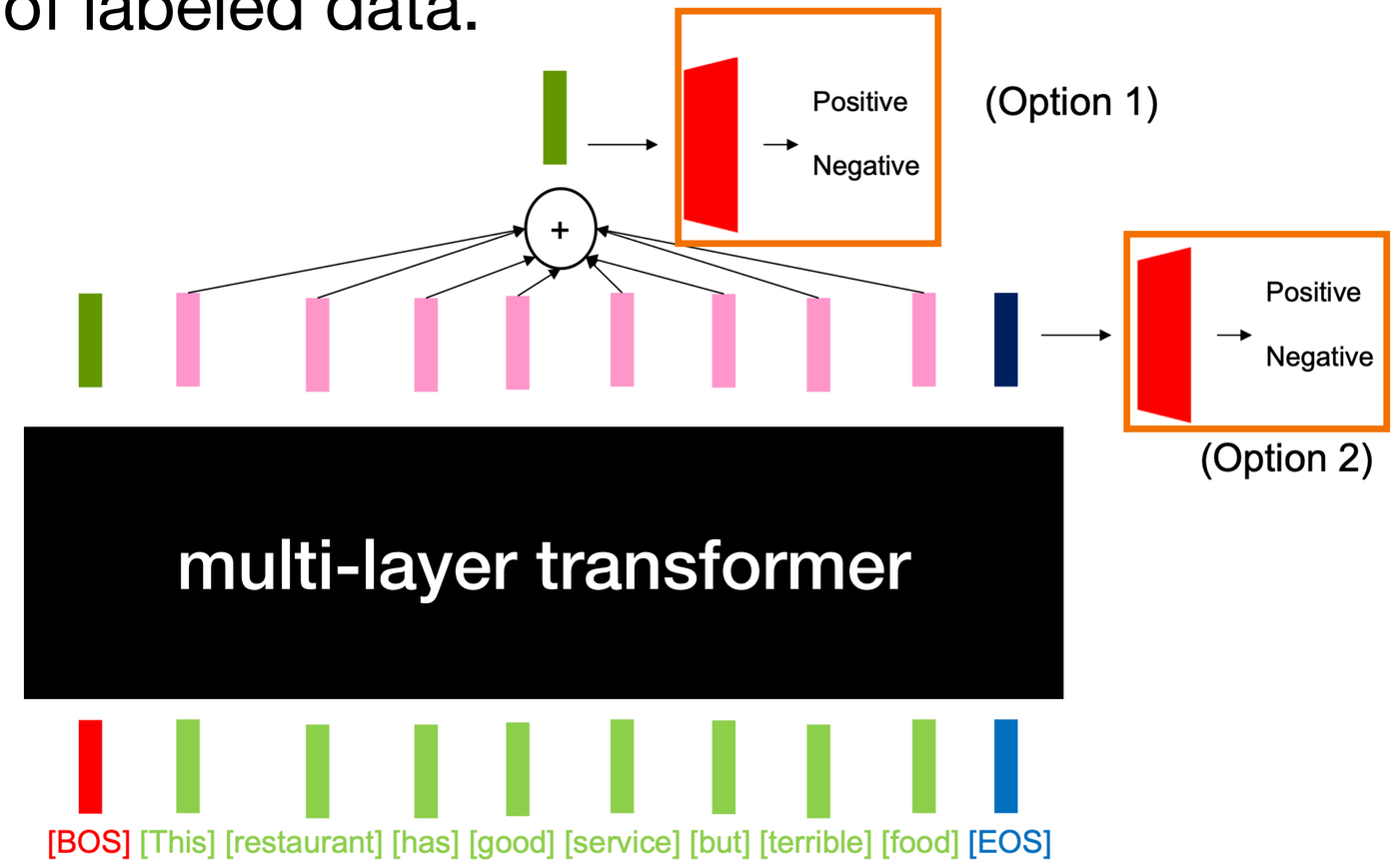
- **Speculative decoding** [Chen et al. 2024] performs at each step,
  1. use ***small*** model to **generate** the next 4 tokens, e.g., “Bond , and you”
  2. pass all [prompt, generated 4 tokens]=“ My name is **Bond, and you** ” to the ***large*** model, and **check** the probability distribution of the all 4 tokens,
  3. **modified rejection sampling**: accept each token one by one with probability that you would have accepted it had you generated each from your sampling scheme (and the large model),
     - if a token is rejected, generate a new token from the already computed distribution.
- Thanks to the modified rejection sampling step, this is **always exactly** the same probability of generating the sequence as using the large model alone.
- It is potentially 4 times faster, since **checking can be done simultaneously**, whereas generating is sequential.
- This is **never slower** than using large model alone (almost), because no large model inference is wasted.

- It is common to **fine-tuning** a language model for downstream tasks, i.e., further train (part of) the model on typically small number of samples specific to a domain or a task.

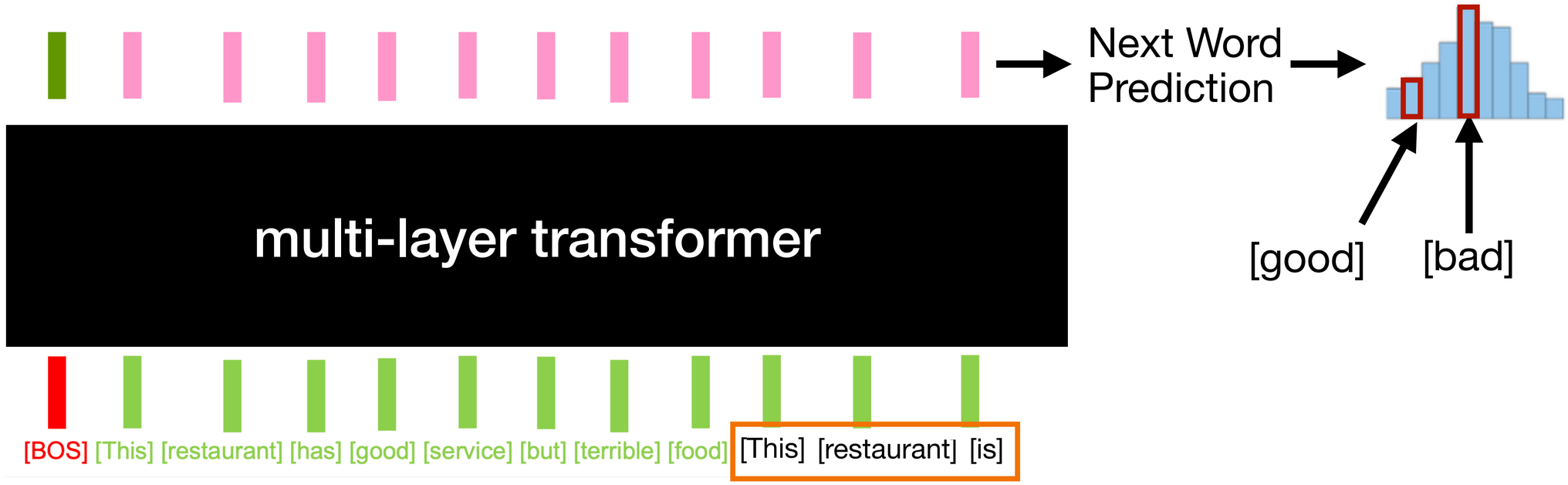
- For document/sentence **classification with FFN**, FFN is appended to (part of) the output representation of the last transformer layer, and this FFN is fine-tuned on a small number of labeled data.

Given fine-tuning training data

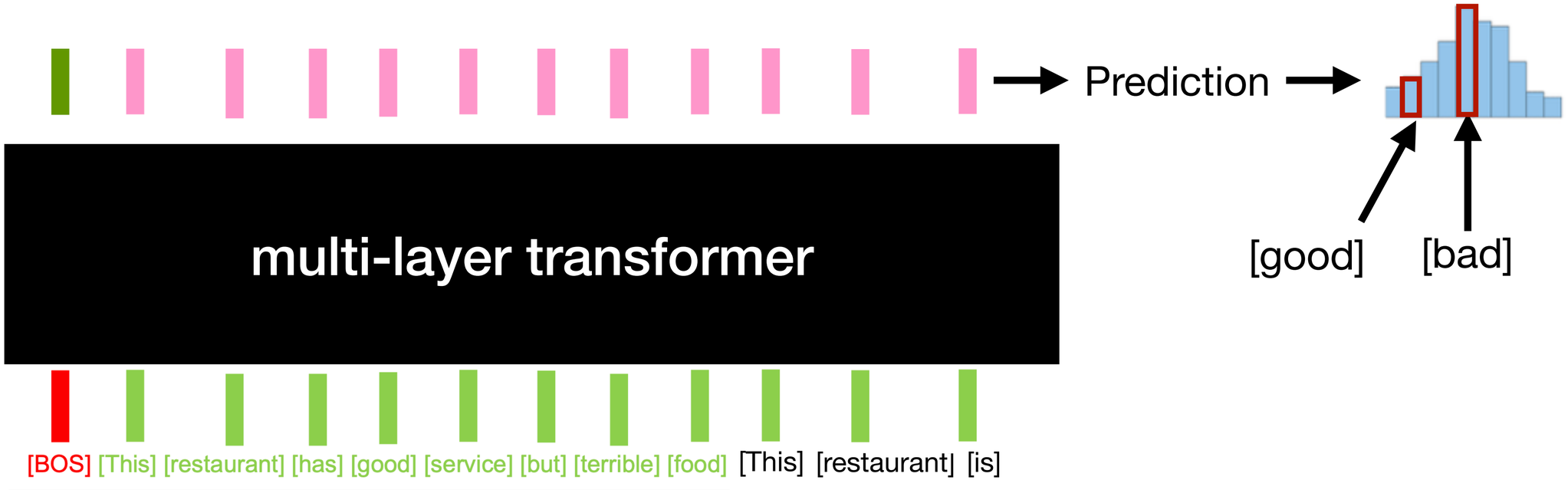
{(review, label $\in \{0,1\}$)}



- Alternatively, one can make classification into a **next token prediction** task, which works significantly better, while still fine-tuning the parameters of the LM on classification training data.



- Classification as next-token prediction improves over fine-tuning a classifier.
[Gao et al. 2021]



Template	Label words	Accuracy
SST-2 (positive/negative)		mean (std)
$<S_1>$ It was [MASK] .	great/terrible	**92.7 (0.9)**
$<S_1>$ It was [MASK] .	good/bad	92.5 (1.0)
$<S_1>$ It was [MASK] .	cat/dog	91.5 (1.4)
$<S_1>$ It was [MASK] .	dog/cat	86.2 (5.4)
$<S_1>$ It was [MASK] .	terrible/great	83.2 (6.9)
Fine-tuning	-	81.4 (3.8)

In-Context Learning

In-Context Learning

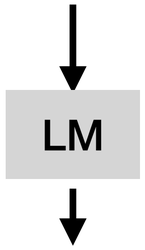
- Now, consider **zero-shot prompting** (without any parameter tuning) for question/answer tasks**,** where we are given a single unlabeled input x (i.e., question) and want to predict its label y (i.e., answer).

  $x =$ "The movie's acting could've been better, but the visuals and directing were top notch."

- **Zero-shot** means you are not given any training examples.

- One way to do it is to wrap x in a template we call a **verbalizer** v

  "Review: The movie's acting could've been better, but the visuals and directing were top notch. Out of positive, negative, or neutral, this review is"



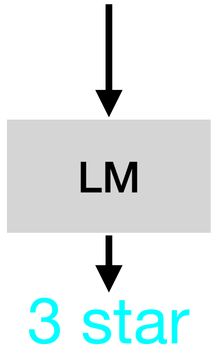
neutral

- Now, consider **zero-shot prompting** (without any parameter tuning) for question/answer tasks**,** where we are given a single unlabeled input x (i.e., question) and want to predict its label y (i.e., answer).

  $x =$ "The movie's acting could've been better, but the visuals and directing were top notch."

- One way to do it is to wrap x in a template we call a **verbalizer** v

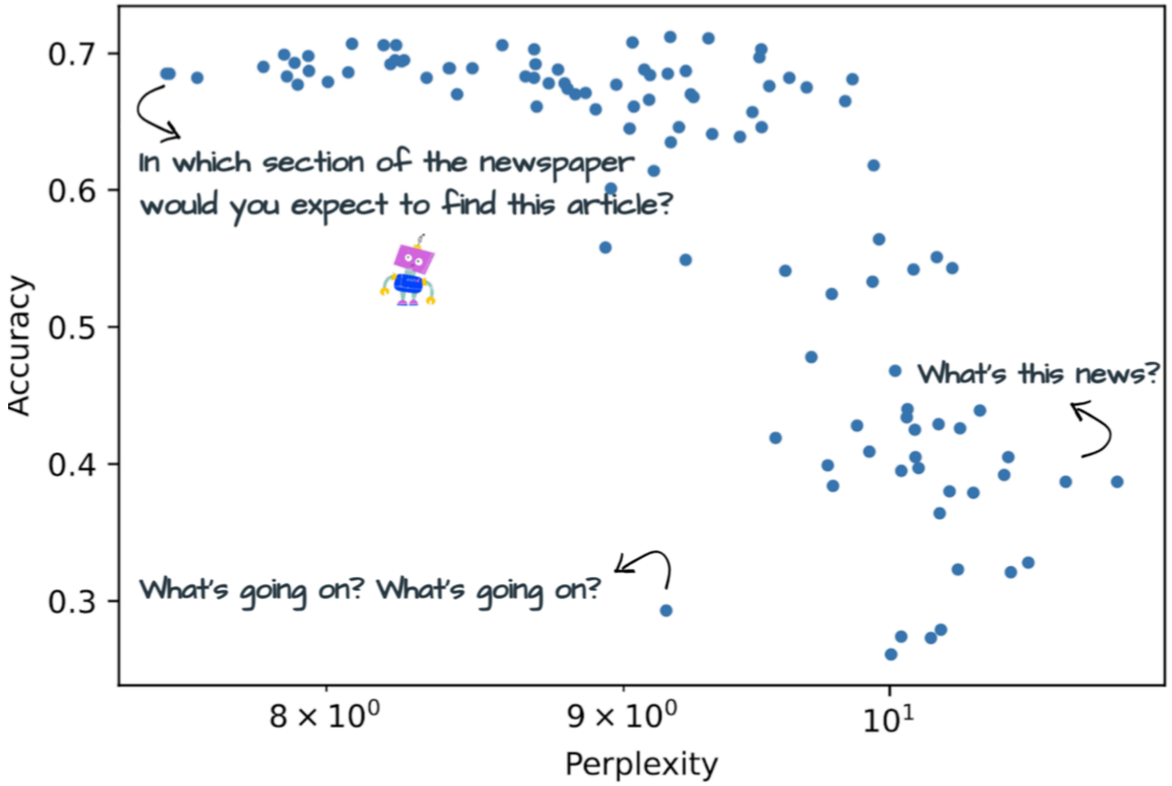
  Review: The movie's acting could've been better, but the visuals and directing were top notch. Out a 1 to 5 start scale, the reviewer would probably give this



- What could go wrong?

- Downsides to zero-shot prompting:
  - If we just use what the model outputs, it might not be of the right format.
  - It is preferred to compare the probability of the valid options, e.g., compare [P(neutral | context), P(negative | context), P(positive | context)]
  - Performance varies based on how we **prompt** it, and it is an art choosing or engineering the right prompt.

- On a large number of prompts for news classification produced by manual writing, paraphrasing, and backtranslation, more natural prompts give less variability.
- **Perplexity** measures how natural the prompt is (lower the more natural)
- **Accuracy** is the classification performance.



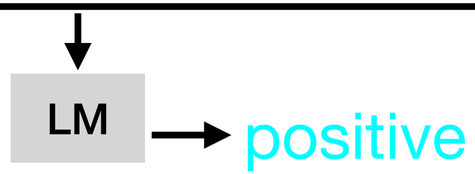
[Gonen et al. 2022]

- A number of prompt optimization techniques use gradients on the prompts or black-box optimization techniques. (note that we do not train/optimize the model, but optimize the prompt in this scenario: **zero-shot prompting**)
- Usually, not much better than manual “optimization” via trial-and-error.
- Instead, consider a scenario where we are given a few in-domain samples: **few-shot prompting** or **in-context learning**

Review: The cinematography was stellar; great movie. Sentiment (positive or negative): positive

Review: The plot was boring and the visuals were subpar. Sentiment (positive or negative): negative

Review: The movie’s acting could’ve been better, but the visuals and directing were top notch. Sentiment (positive or negative):

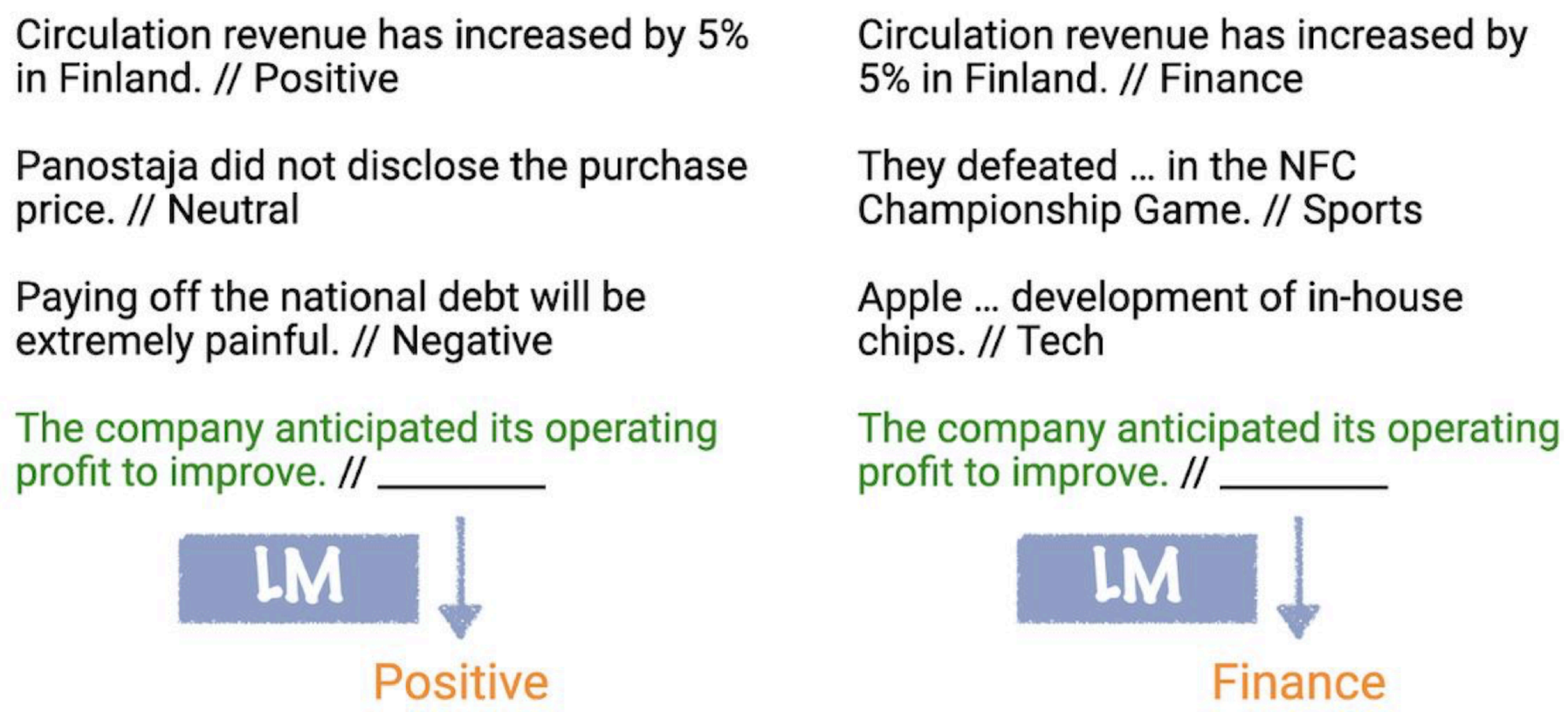


- In-context learning is much more efficient than fine-tuning with the **few-shot** examples, and gives significant improvements on downstream tasks, comparable with parameter fine-tuned models.

[Min et al. 2022]

- How does in-context learning work?

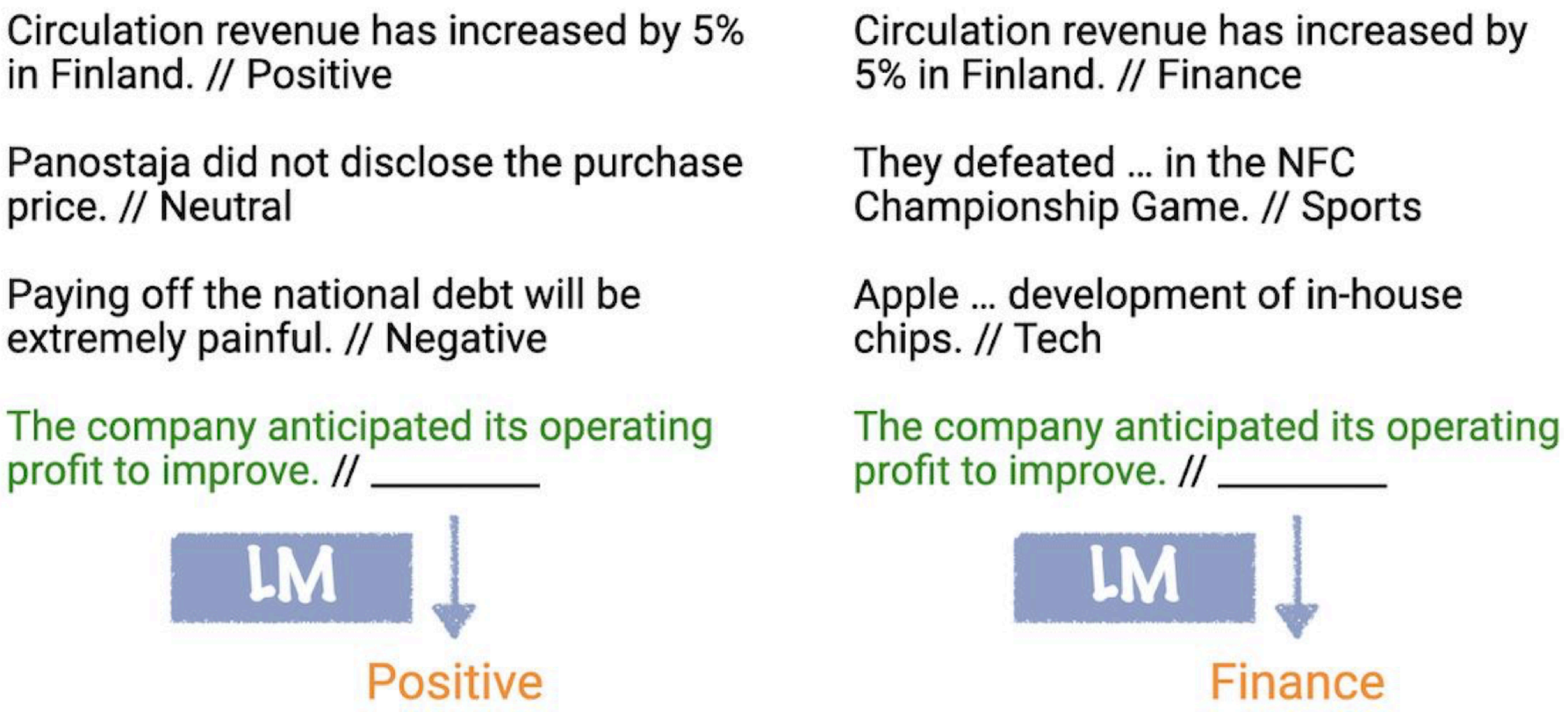
Visual taken from: https://ai.stanford.edu/blog/understanding-incontext/



- Model needs to figure out:

- How does in-context learning work?

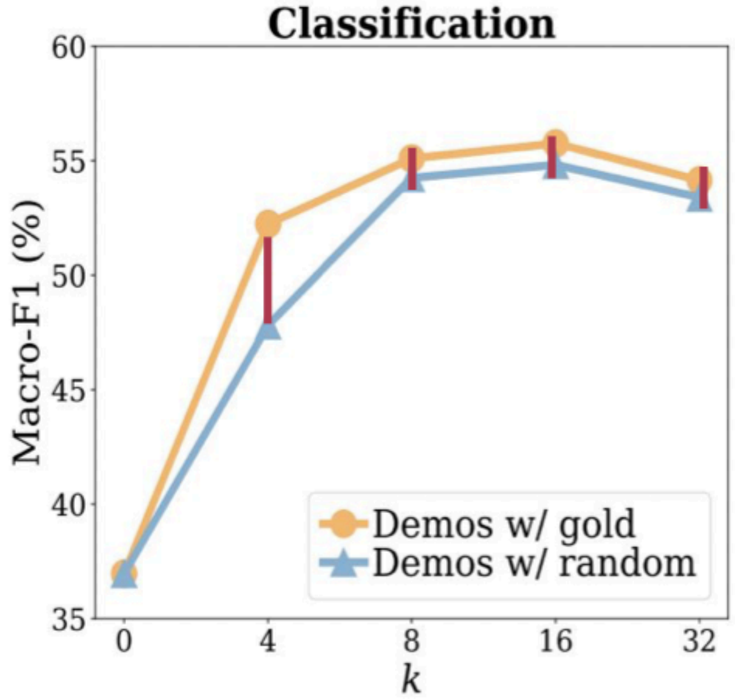
Visual taken from: https://ai.stanford.edu/blog/understanding-incontext/

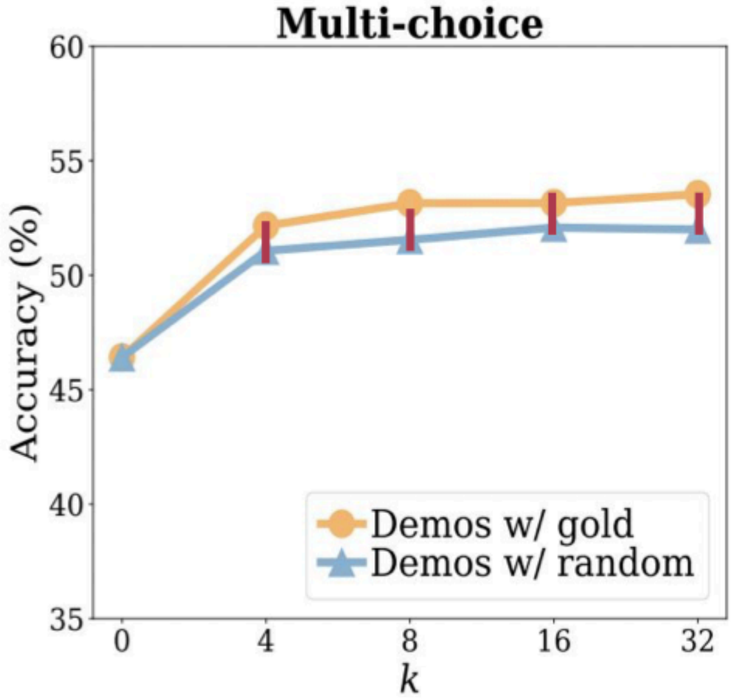


- Model needs to figure out:
  - **input (Question) distribution** (I): financial assessment or news topic
  - **label distribution/format** (L): positive/neutral/negative or topic
  - **formatting** (F)
  - **input-output mapping** (M): $\mathbb{P}(\text{answer} \mid \text{question})$

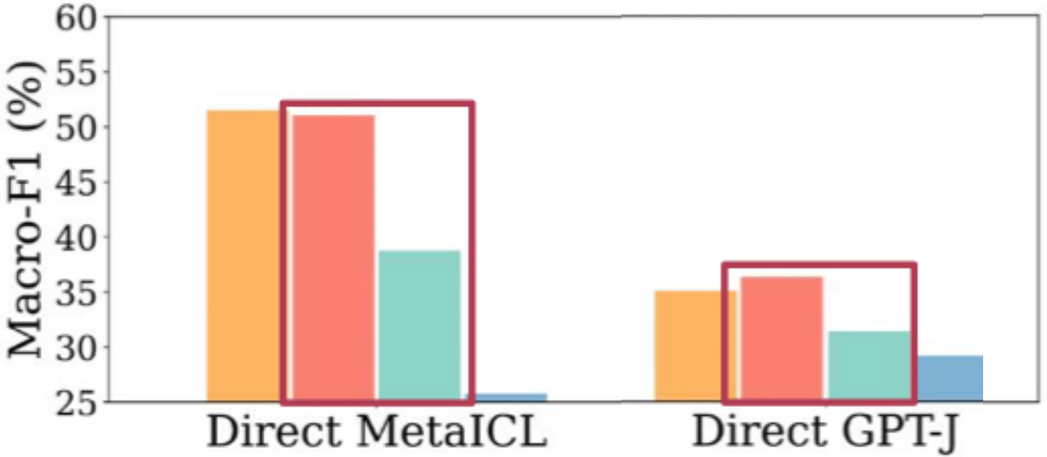
- Surprisingly, the **quality of the label** or **Input-output mapping** in the in-context learning is not as relevant.
- It is the **format/domain** of the demonstration examples that matters.

- One can generate randomly labelled examples to improve performance.





- But seeing the correct label space is important.



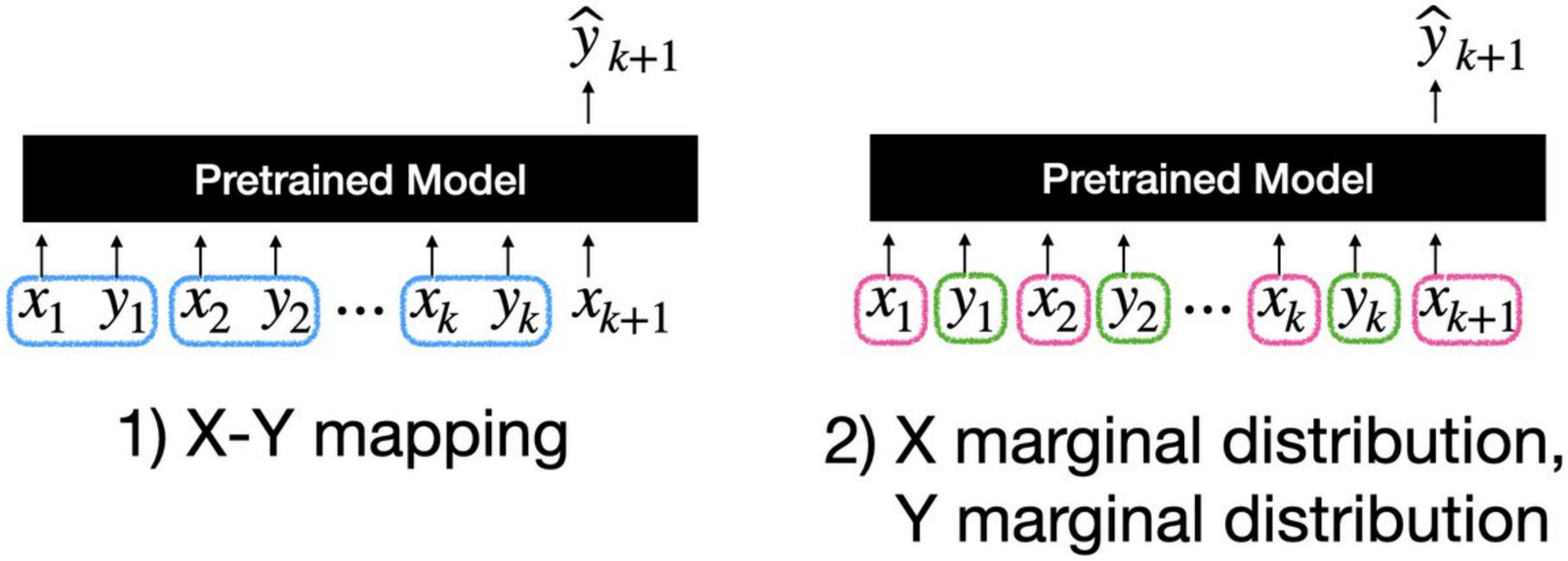
	F	*L*	*I*	*M*
Gold labels	✓	✓	✓	✓
Random labels	✓	✓	✓	✗
Random English words	✓	✗	✓	✗
No demonstrations	✗	✗	✗	✗

F: Format
L: Label space
I: Input distribution
M: Input-Label Mapping

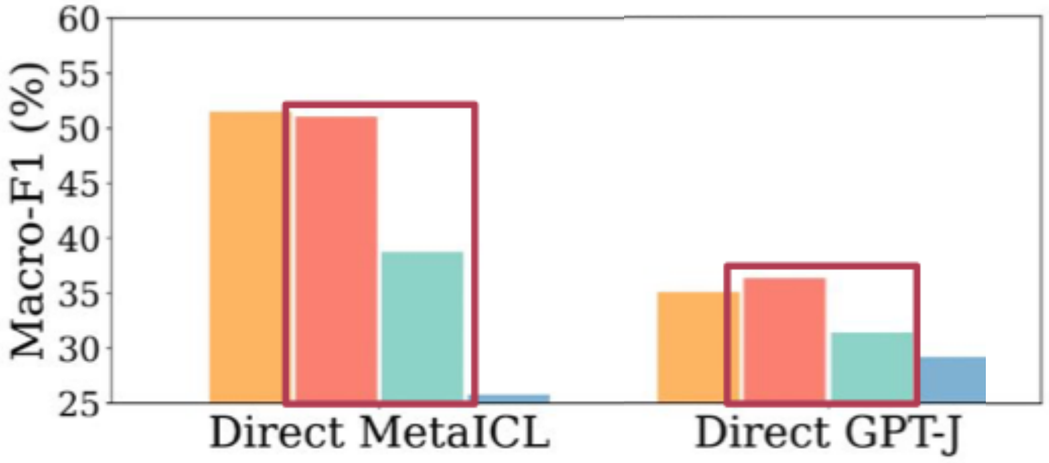
- LM pretaining as **topic modeling** [Xie et al. 2022].
- Hypothesis:
  - Pretraining documents have long-coherence, i.e., sentences in the same document share the same topic/concept.
  - During pretraining: LM is trained to
    - **infers the topic** from previous sentences as evidence, and then
    - **predict the tokens** conditioned on the topic
  - During In-context learning:
    - **infers the topic** shared by the in-context examples, and then
    - **predict the tokens** conditioned on the topic

$$\mathbb{P}(\text{answer} \mid \text{ICL prompt}) = \sum_{\text{topics}} \underbrace{\mathbb{P}(\text{answer} \mid \text{topic, ICL prompt})}_{\text{LM pretaining}} \times \underbrace{\mathbb{P}(\text{topic} \mid \text{ICL prompt})}_{\text{at inference-time, ICL provides proper weights}}$$

- In-context learning **retrieves the topic/concept** of the domain from a few examples, and **focuses on the skill** needed in that domain
- In the Bayesian setting, this can be done using just the **marginal distributions of x and y :** (I) and (L) in our terminology.
- In particular, this explains why random input-output mapping still gives gain



- On the other hand, if we are given many samples for ICL, the LM could learn the input-output mapping from the samples: (M) in our terminology.

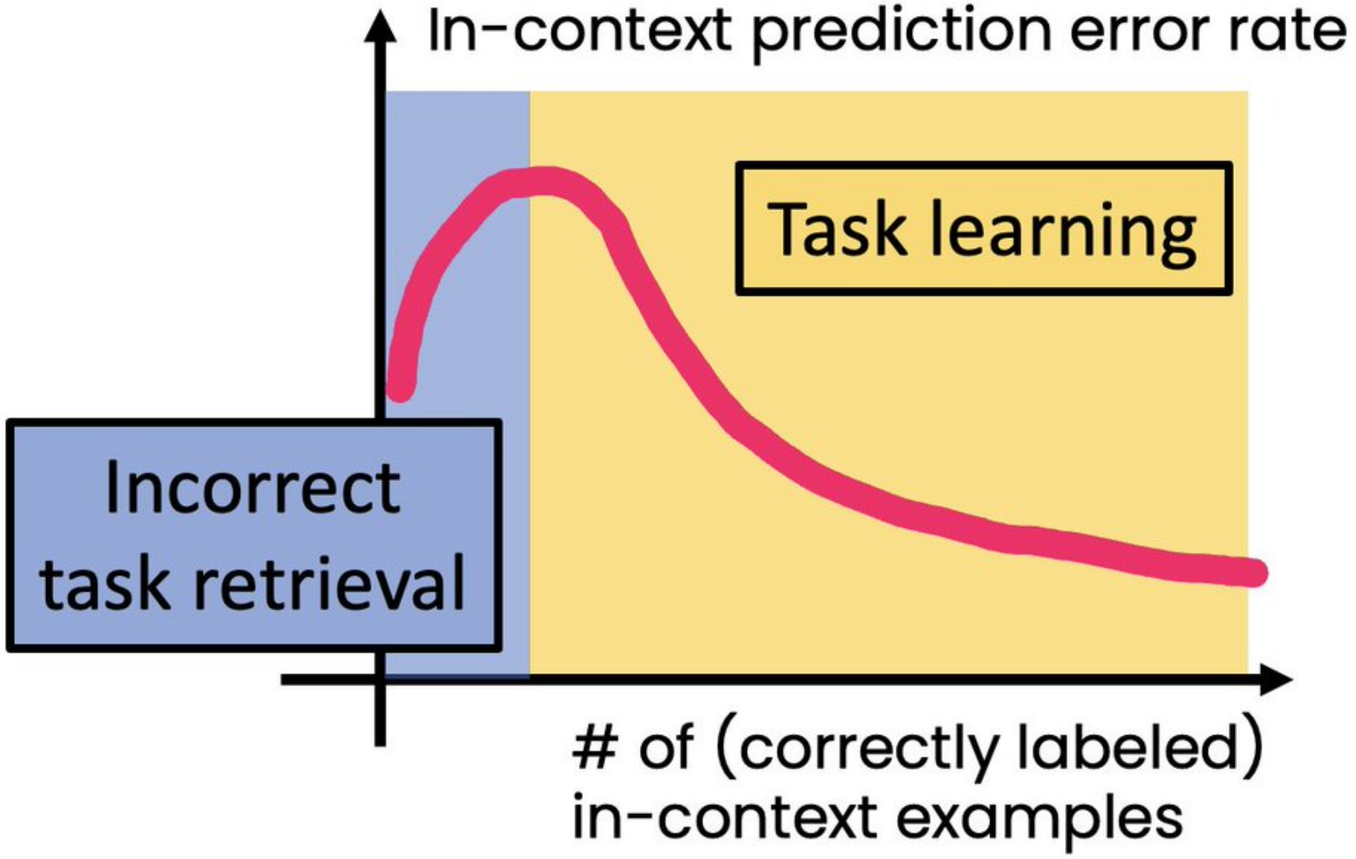


	F	*L*	*I*	*M*	
Gold labels	✓	✓	✓	✓	$I + L + M + F$
Random labels	✓	✓	✓	✗	$I + L + F$
Random English words	✓	✗	✓	✗	$I + F$
No demonstrations	✗	✗	✗	✗	

F: Format
L: Label space
I: Input distribution
M: Input-Label Mapping

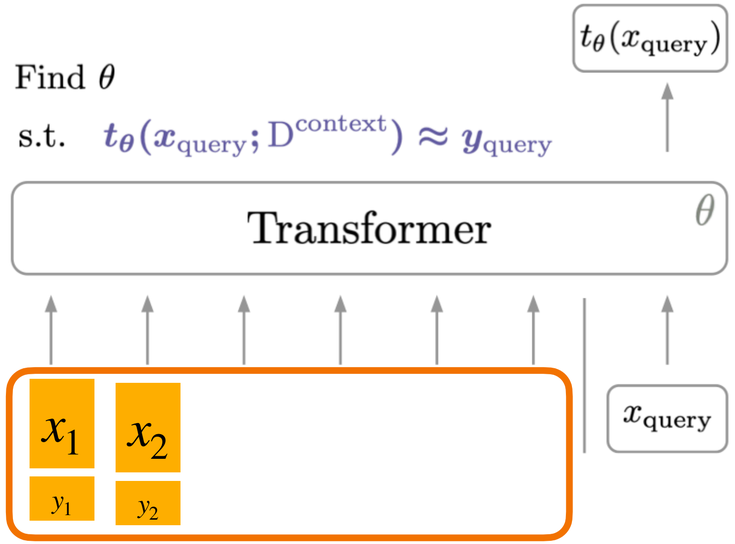
[Lin and Lee, 2024]

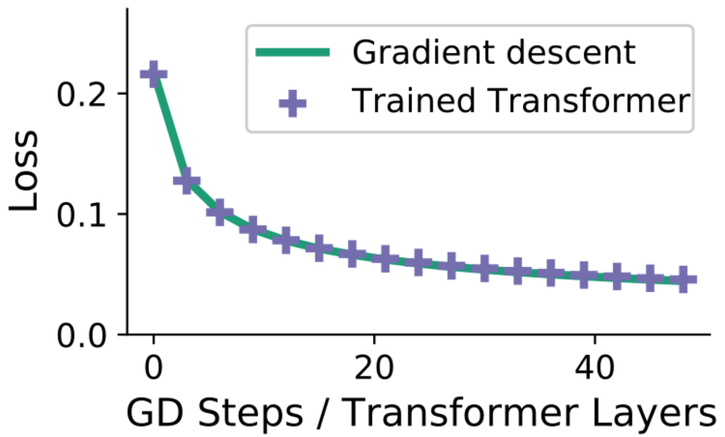
- The **Early ascent phenomena** is common in **In-context Learning**.
- With small number of samples, the marginal distributions of x and y can lead to wrong concept being retrieved.
- This can be fixed if given more samples, where the task is learned from the given samples.



[Lin and Lee, 2024]

- Toy example to understand in-context learning: **linear regression**





- Several theoretical explanations under linear models.
- Typically, explores equivalence between, for example,
  - GD update on linear model and regression loss;
  - In-context Learning with (specific) single attention layer.

[van Oswald et al. 2022] [Akyürek et al. 2022]

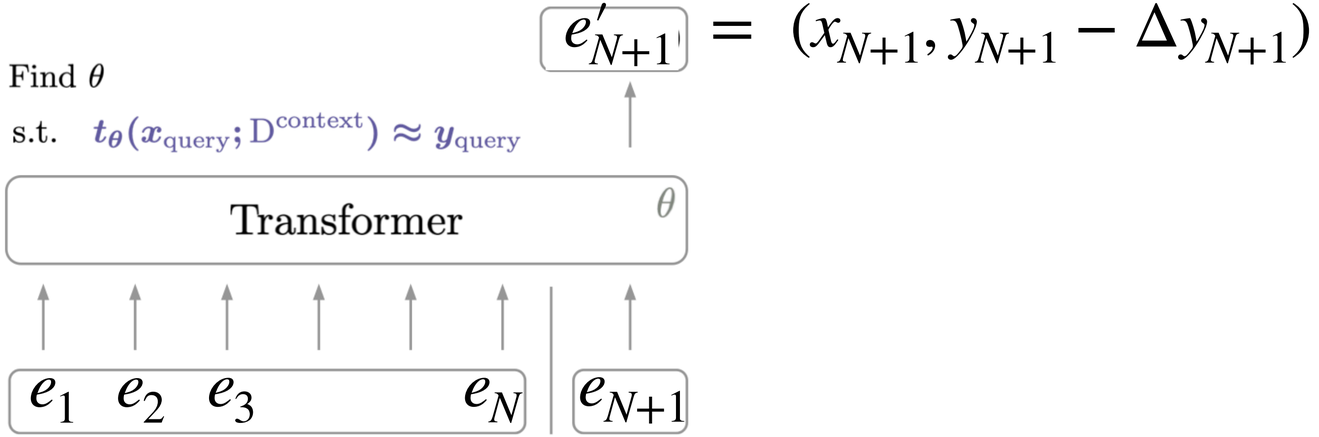
Assume N training pairs $\{(x_i, y_i)\}_{i=1}^N$ where $x_i \in \mathbb{R}^{N_x}$ and $y_i \in \mathbb{R}^{N_y}$ with a single test query (x_{test}, y_{test}). We can merge each pair into a single query:

$$e_i = (x_i, y_i), \text{ for } i = (1, \ldots, N) \in \mathbb{R}^{N_x + N_y}$$

and for query token, we set:

$$e_{N+1} = (x_{test}, -W_0 x_{test})$$

Here W_0 is the "initial" weight matrix (often set to zero) before performing the gradient descent updates.

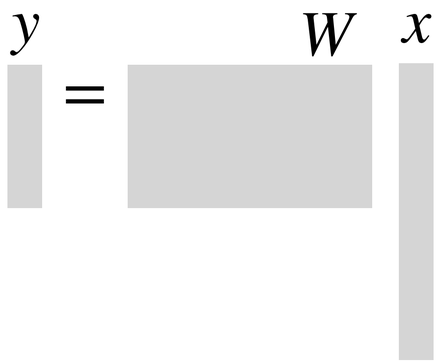


Proposition 1. *Given a 1-head linear attention layer and the tokens $e_j = (x_j, y_j)$, for $j = 1, \ldots, N$, one can construct key, query and value matrices W_K, W_Q, W_V as well as the projection matrix P such that a Transformer step on every token e_j is identical to the gradient-induced dynamics $e_j \leftarrow (x_j, y_j) + (0, -\Delta W x_j) = (x_i, y_i) + PVK^T q_j$ such that $e_j = (x_j, y_j - \Delta y_j)$. For the test data token (x_{N+1}, y_{N+1}) the dynamics are identical.*

- **multi-dimensional** Linear regression recap: Gradient Descent

Given a linear model as:

$$y = Wx$$

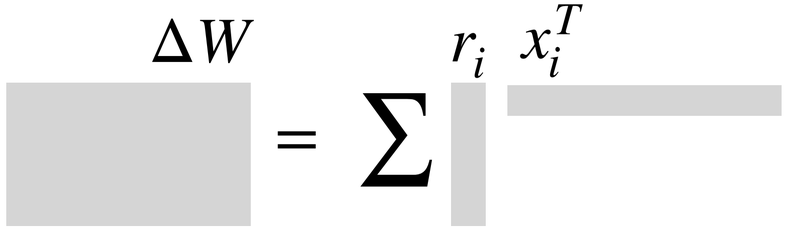


under the MSE regression loss:

$$L(W) = \frac{1}{2N} \sum_{i=1}^{N} ||Wx_i - y_i||_2^2$$

A single-step of gradient with learning rate η updates W:

$$\Delta W = -\eta \nabla_W L(W) = -\frac{\eta}{N} \sum_{i=1}^{N} (Wx_i - y_i) x_i^\top.$$



Hence, the new weights would be: $W_{new} = W + \Delta W$.

- Can we design self-attention matrices (K,Q,V) that matches this GD update?

Consider a single-head self-attention layer of the following form:

$$e_j \leftarrow e_j + PV(K^\top q_j),$$

for P being a linear projection matrix and each token will be mapped into learnable "query", "key", "value" matrices are typically called:

$$q_j = W_Q e_j,\ k_j = W_k e_j,\ v_j = W_v e_j.$$

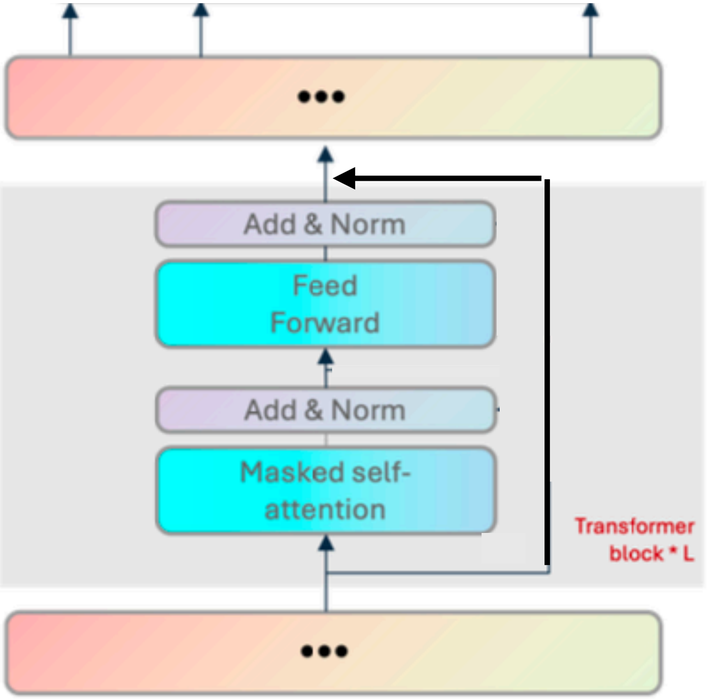
then combined in the usual self-attention formula.

We can write the self-attention formula as:

$$e_j \leftarrow e_j + P\sum_{i=1}^{N} v_i \otimes k_i q_j = e_j + PW_v \sum_{i=1}^{N} e_i \otimes e_i W_K^\top W_Q e_j.$$

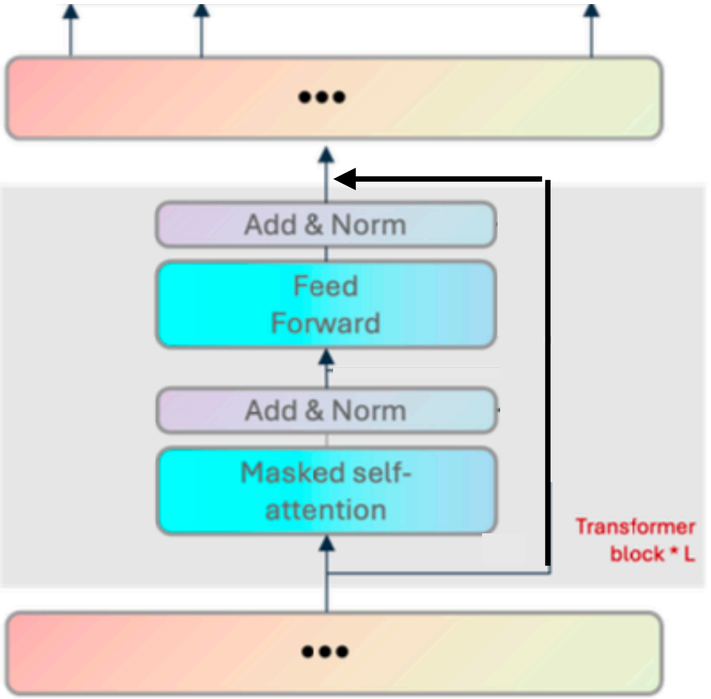
where $\otimes$ represents the Outer product between two vectors.

Skip connection



- We want to match $I - \frac{\eta}{N}\sum_{i=1}^{N}(Wx_i - y_i)x_i^\top$

  with $e_j \leftarrow e_j + P\sum_{i=1}^{N} v \otimes kq_j = e_j + PW_v \sum_{i=1}^{N} e_i \otimes e_i W_K^\top W_Q e_j.$



For instance, write each token e_j as $(x_i, y_i) \in \mathbb{R}^{N_X+N_Y}$. Then define:

$$W_K = W_Q = \begin{pmatrix} I_{N_x} & 0 \\ 0 & 0 \end{pmatrix},$$

As this ensures that the keys, and queries look into "x" or input part of each token.

We can choose the weights of values as:

$$W_V = \begin{pmatrix} 0 & 0 \\ W_0 & -I_{N_y} \end{pmatrix},$$

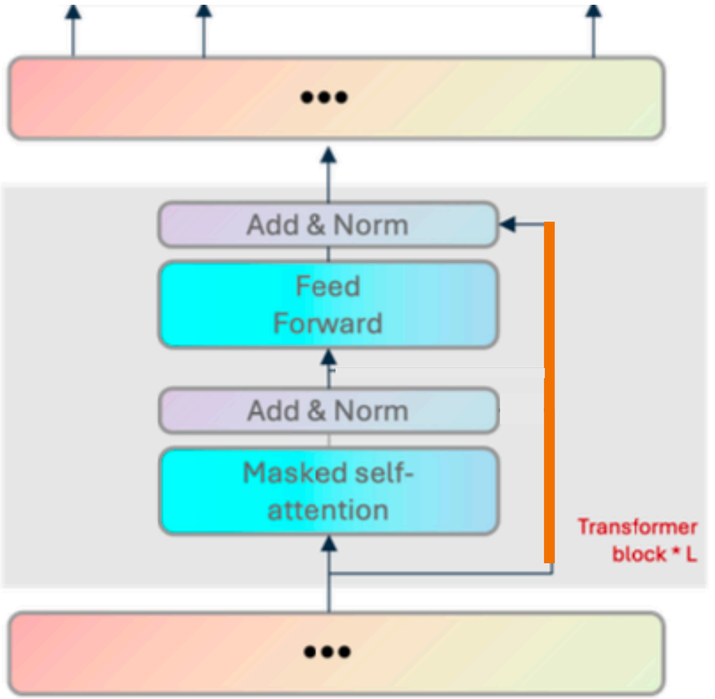
Here, W_0 is the initial linear model weight being finetuned and $-I_{N_y}$ subtracts the target from the value.

Moreover, we can define the projection matrix P as follows:

$$P = \frac{\eta}{N} I_{N_X+N_y}$$

- We want to match $I - \frac{\eta}{N}\sum_{i=1}^{N}(Wx_i - y_i)x_i^\top$

  with $e_j \leftarrow e_j + P\sum_{i=1}^{N} v \otimes kq_j = e_j + PW_v \sum_{i=1}^{N} e_i \otimes e_i W_K^\top W_Q e_j.$



Let us put the pieces together and put the construction of the key, value and query matrices together:

$$\begin{pmatrix} x_j \\ y_j \end{pmatrix} \leftarrow \begin{pmatrix} x_j \\ y_j \end{pmatrix} + \frac{\eta}{N} I \sum_i \left(\begin{pmatrix} 0 & 0 \\ W_0 & -I_y \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} \right) \otimes \left(\begin{pmatrix} I_x & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} \right) \begin{pmatrix} I_x & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_j \\ y_j, \end{pmatrix}$$

The left-hand side of this update function can be further simplified to:

$$\begin{pmatrix} x_j \\ y_j \end{pmatrix} \leftarrow \begin{pmatrix} x_j \\ y_j \end{pmatrix} + \frac{\eta}{N} I \sum_i \begin{pmatrix} 0 \\ W_0 x_i - y_i \end{pmatrix} \otimes \begin{pmatrix} x_i \\ 0 \end{pmatrix} \begin{pmatrix} x_j \\ 0 \end{pmatrix} = \begin{pmatrix} x_j \\ y_j \end{pmatrix} + \begin{pmatrix} 0 \\ -\Delta W x_j \end{pmatrix}.$$

Sources

- Other courses in LLMs that the lecture slides are based on
  - CSE493S/599S at UW by Ludwig Schmidt: https://mlfoundations.github.io/advancedml-sp23/
  - EE-628 at EPFL by Volkan Cevher: https://www.epfl.ch/labs/lions/teaching/ee-628-training-large-language-models/ee-628-slides-2025/
- Useful blog posts
  - https://azizbelaweid.substack.com/p/complete-summary-of-absolute-relative
  - https://blog.dust.tt/speculative-sampling-llms-writing-a-lot-faster-using-other-llms/
  - https://gordicaleksa.medium.com/eli5-flash-attention-5c44017022ad
- Dan Jurafsky and James H. Martin. Speech and Language Processing (3rd ed. draft). draft, third edition, 2023.
- Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. “Efficient estimation of word representations in vector space”, In International Conference on Learning Representations, 2013.
- Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "Glove: Global vectors for word representation”, Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014.
- Ofir Press, Noah A. Smith1,3 Mike Lewis2 , “Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation”, In International Conference on Learning Representations, 2022
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, “Attention Is All You Need”, In Neural Information Processing Systems, 2017
- Beitong Zhou, Cheng Cheng, Guijun Ma, and Yong Zhang. “Remaining useful life prediction of lithium-ion battery based on attention mechanism with positional encoding”, In IOP Conference Series: Materials Science and Engineering, 2020.
- Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the difficulty of training recurrent neural networks." In International Conference on Machine Learning, 2013

- Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory.” In Neural Computation, 9(8):1735–1780, 11 1997.
- Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. “Learning phrase representations using rnn encoder-decoder for statistical machine translation”, In ACL 2014
- Andrey Andreyevich Markov. “Essai d’une recherche statistique sur le texte du roman. ‘Eugene Onegin’ illustrant la liaison des epreuve en chain”. In: Izvistia Imperatorskoi Akademii Nauk (Bulletin de l’Académie Impériale des Sciences de St.-Pétersbourg). 6th ser, 7:153–162, 1913.
- Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, Yejin Choi, “The Curious Case of Neural Text Degeneration”, In *International Conference on Learning Representations*, 2020
- Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre and John Jumper, “Accelerating Large Language Model Decoding with Speculative Sampling”In, *ACL-findings*, 2024
- Sergey Ioffe, Christian Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", In *International Conference on Machine Learning*, 2015
- Shibani Santurkar∗ MIT shibani@mit.edu Dimitris Tsipras∗ MIT tsipras@mit.edu Andrew Ilyas∗ MIT ailyas@mit.edu Aleksander Madry, “How Does Batch Normalization Help Optimization?”, In *Advances in Neural Information Processing Systems* 31 (NeurIPS 2018)
- Jimmy Lei Ba, Jamie Ryan Kiros, Geoffrey E. Hinton, “Layer Normalization “, In 2016
- Tianyu Gao, Adam Fisch, Danqi Chen, “Making Pre-trained Language Models Better Few-shot Learners”, In *ACL*, 2021
- Sewon Min1,2 Xinxi Lyu1 Ari Holtzman1 Mikel Artetxe2 Mike Lewis2 Hannaneh Hajishirzi1,3 Luke Zettlemoyer, “rethinking the role of demonstrations what makes in conte…”
- Hila Gonen1,2 Srini Iyer2 Terra Blevins1 Noah A. Smith1,3 Luke Zettlemoyer1, “Demystifying Prompts in Language Models via Perplexity Estimation”
- E Akyürek, B Wang, Y Kim, J Andreas , “In-context language learning: Architectures and algorithms”, 2024
- What learning algorithm is in-context learning? Investigations with linear models Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, Denny Zhou, 2022
- Ziqian Lin, Kangwook Lee, “Dual Operating Modes of In-Context Learning”, 2024