L01-1 Lecture note (Introduction to transformer, LLM and Instruct GPT)

2023-09-20

School of EE at KU
Sanghoon Sull
sull@korea.ac.kr

Contents

- Transformer and LLMs
- ChatGPT: Improved LLM by human feedback

Instruct GPT

Transformer and LLMs

Transformer, NIPS 2017

NIPS 2017

88974 citations as of 2023-09-17

Applications: NLP, vision, image, ViT

protein folding and others

Alphafold

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

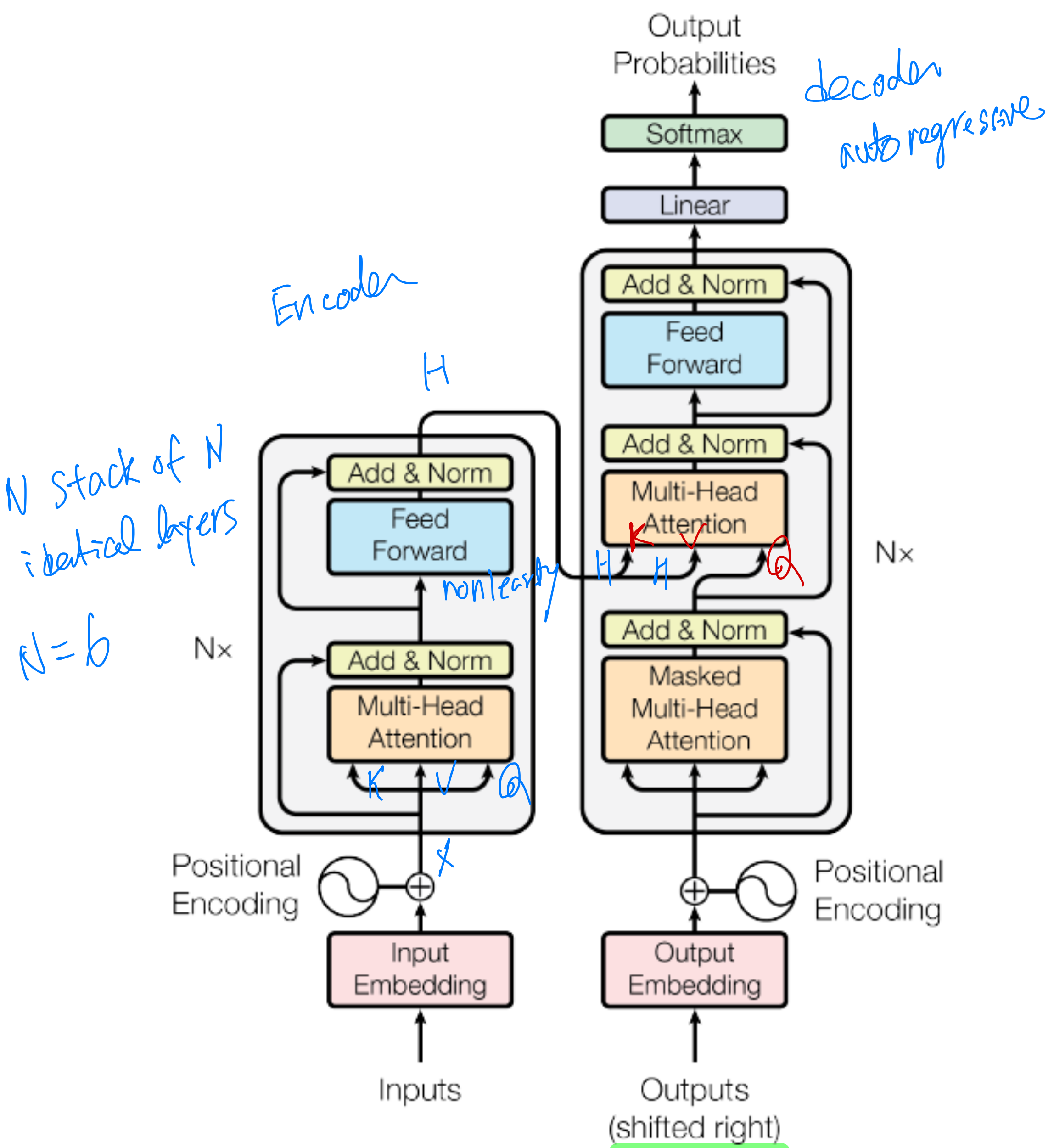
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

Model architecture

1. Positional encoding for order
2. Nonlinearities by FF
3. To enable parallelization during training of the decoder, mask the future in self-attn
4. Given a batch X of input vectors, compute XK, XQ and XV and then scaled dot-product (key-query-value) attn for all pairs.
5. Multi-head attn for diverse viewpoints
6. Residual connections for better training
7. Layer normalization for faster training
8. Quadratic complexity



Key-Query-Value attention

$\boxed{W^Q}$ $\boxed{\cdot}$ $\Rightarrow$ x_i^T

$1 \times d_m$

dot product: $x_i \cdot x_j^T$

$d_m \times 1$

Q: $W^Q \cdot x_i^T$

- Input: $x_1 \ldots x_T$ Output: $z_1 \ldots z_T$

A token in a transformer model is a single unit of text, such as a word, subword, or punctuation mark. Here assume token ≈ word.

$$X = \begin{bmatrix} x_1 \\ \ldots \\ x_T \end{bmatrix} \in R^{T \times d_m}$$

d model = 512 (base model)

$XW^K = K \in R^{T \times d_k}$	$W^K \in R^{d_m \times d_k}$	Key matrix
$XW^Q = Q \in R^{T \times d_k}$	$W^Q \in R^{d_m \times d_k}$	Query matrix
$XW^V = V \in R^{T \times d_v}$	$W^V \in R^{d_m \times d_v}$	Value matrix

Key, query and value matrices ≈ projection matrices

Single-head attention

Softmax $y_1 \; y_2 \; \cdots \; y_T$

$$XW^Q (XW^K)^T \cdot V$$

$T \times T$ $\quad$ $T \times d_v$

$$\frac{e^{y_1}}{e^{y_1} + \cdots + e^{y_T}}$$

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \qquad (1)$$

$XW^K = K \in R^{T \times d_k}$	$W^K \in R^{d_m \times d_k}$	Key weight matrix
$XW^Q = Q \in R^{T \times d_k}$	$W^Q \in R^{d_m \times d_k}$	Query weight matrix
$XW^V = V \in R^{T \times d_v}$	$W^V \in R^{d_m \times d_v}$	Value weight matrix

Key, query and value matrices: linear projection matrices

Use dot-product attention.

$$X \in R^{T \times d_m} \rightarrow Attention \in R^{T \times d_v}$$

T $\quad$ d_m $\qquad$ T $\quad$ d_v

Single-head attention

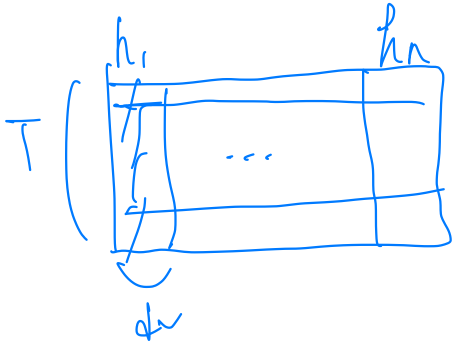
$$\underset{T\times d_k}{\overset{Q}{XW^Q}}\;\underset{d_k\times T}{\overset{K^T}{{W^K}^{\mathrm{T}}X^{\mathrm{T}}}} = \overset{QK^T}{XW^Q{W^K}^{\mathrm{T}}X^{\mathrm{T}}} \in \mathbb{R}^{T\times T}$$

All pairs of attention scores

$$softmax\left[\underset{T\times T}{XW^Q{W^K}^{\mathrm{T}}X^{\mathrm{T}}}\right]\underset{T\times d_v}{XW^V} = \begin{array}{c} 1 \\ 2 \\ 3 \\ \vdots \\ \vdots \\ \vdots \\ T \end{array}\begin{bmatrix} \\ \\ \\ \\ \\ \\ \\ \end{bmatrix}$$

output $\in \mathbb{R}^{T\times d_v}$

Multi-head attention



$T \times dm$ $\quad$ $dm \times dm$

$$MultiHead(Q,K,V) = \underbrace{Concat(head_1, \ldots, head_h)}_{T \times (hdv)\ =\ dm}W^O$$

where $\underbrace{head_i}_{T \times dv} = Attention(\underbrace{X}_{T \times dm}\underbrace{W_i^Q}_{dm \times dk}, XW_i^K, \underbrace{XW_i^V}_{T \times dv})$

Projections are parameter matrices:

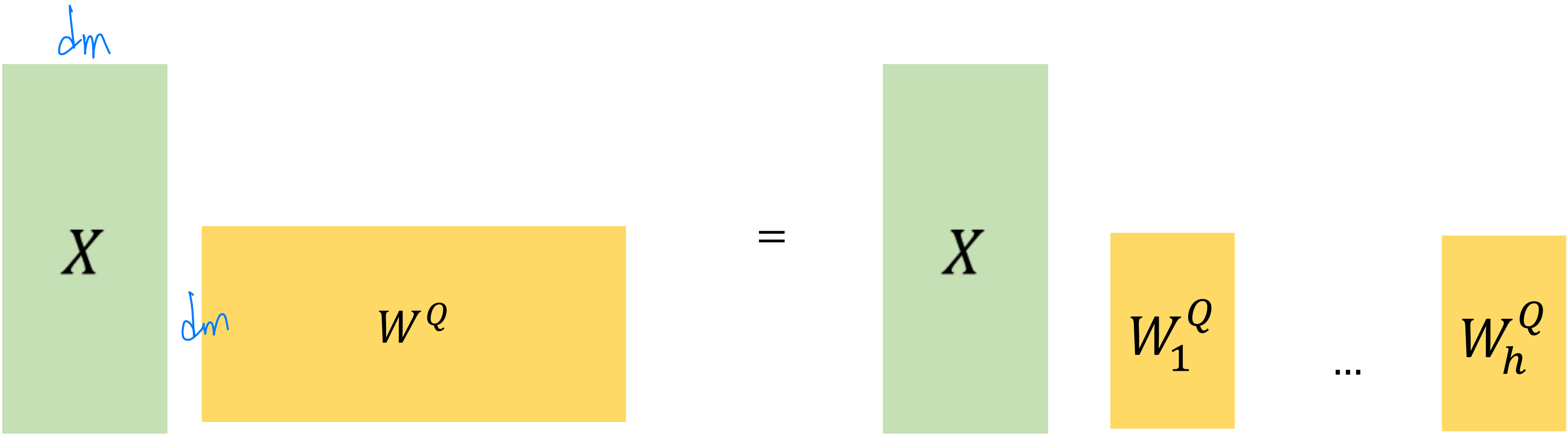
$W_i^Q \in R^{d_m \times d_k}$ $\quad W_i^K \in R^{d_m \times d_k}$ $\quad W_i^V \in R^{d_m \times d_V}$ $\quad W^O \in R^{h\,d\_v \times d_m}$

Assume $h = 8, d_k = d_v = \frac{d_m}{h} = 64$

$$X \in R^{T \times d_m} \rightarrow MultiHead \in R^{T \times d_m}$$

Multi-head attention

h: # of heads



$W^Q \in R^{d_m \times hd_k}$

$W_i^Q \in R^{d_m \times d_k}$

FFN

ReLU:

$$FFN(X) = \max(0, \overbrace{XW_1 + b_1}^{T\times d_{ff}})W_2 + b_2$$

$X \in R^{T\times d_m}$ $W_1 \in R^{d_m \times d_{ff}}$

$W_2 \in R^{d_{ff}\times d_m}$

# of parameters:

attention: $d_k = d_v = \frac{d_m}{h}$ $\Rightarrow$ $4 d_m^2$

FFN: $2 \times d_m \times d_{ff}$

$d_m = 512$ $d_k = 64$ $h = 8$ $d_{ff} = 2048$

Attn: 1M FFN: 2M

Transformer

Attention visualizations:

It is in this spirit that a majority of American governments have passed new laws since 2009 making the registration of voting process more difficult.

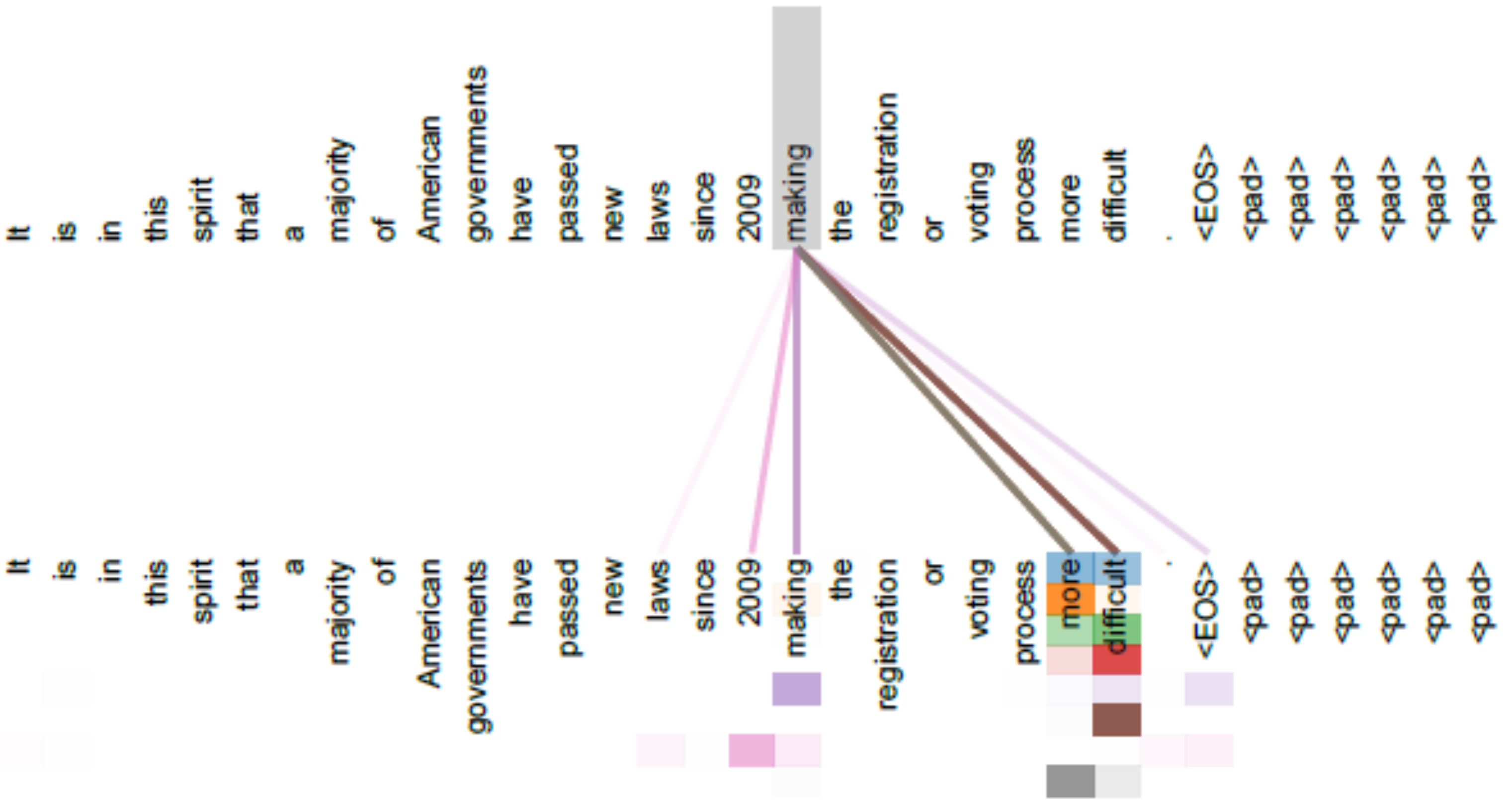
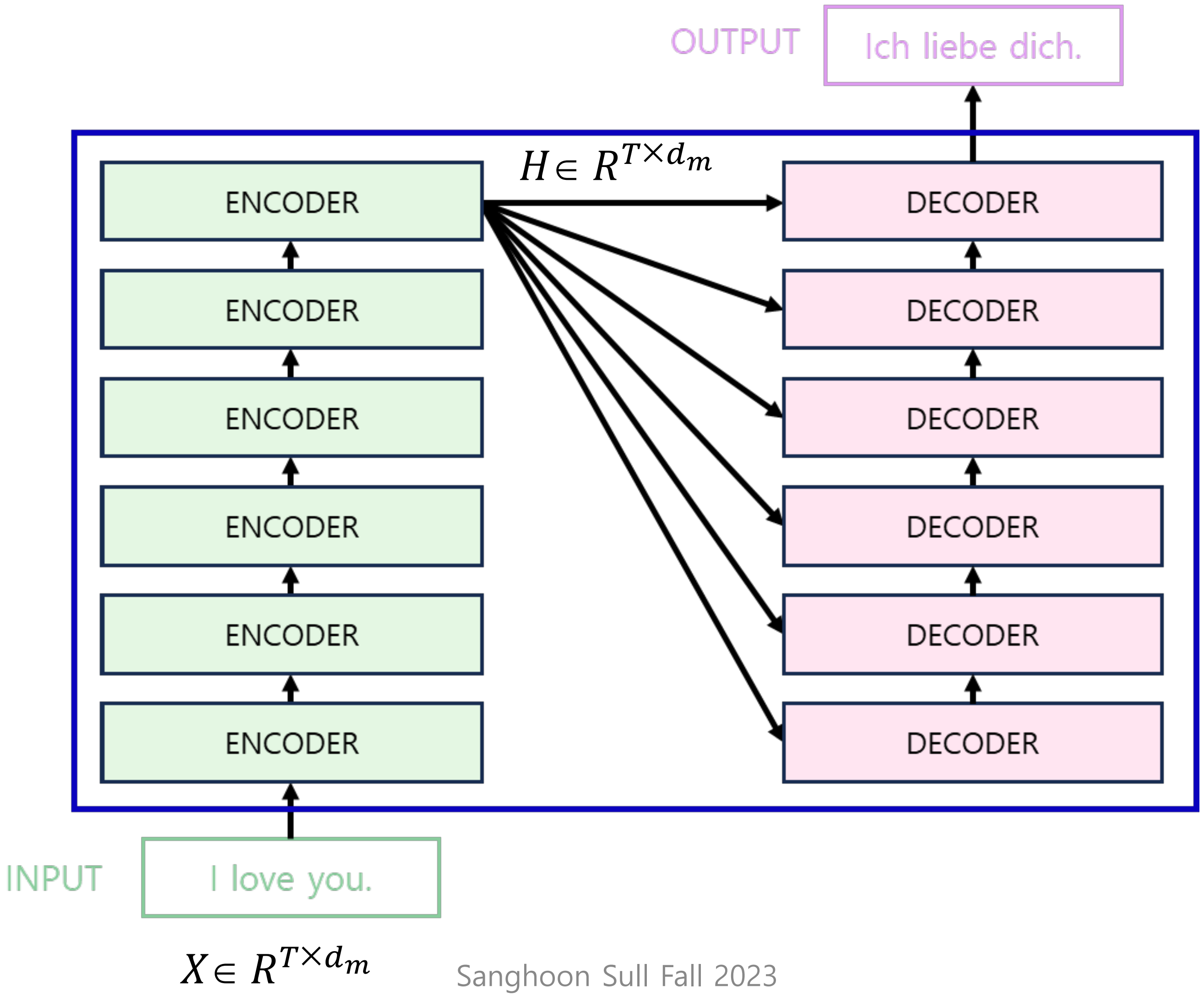
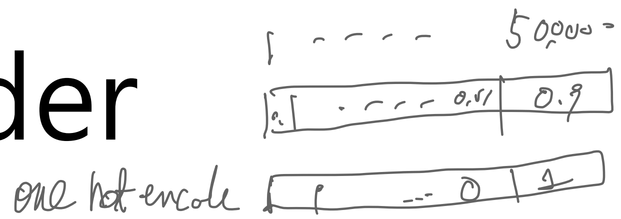


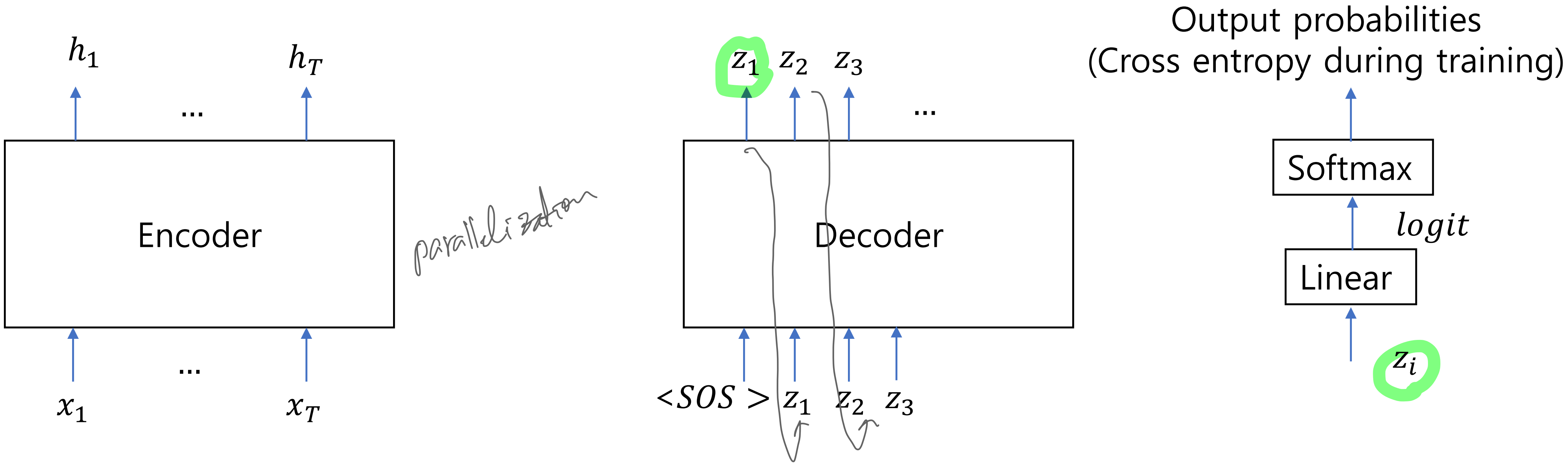
Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb 'making', completing the phrase 'making...more difficult'. Attentions here shown only for the word 'making'. Different colors represent different heads. Best viewed in color.

Transformer: Encoder and decoder



Transformer: Encoder and decoder





Input: $x_1, \ldots, x_T$

Output: $z_1, \ldots, z_T$

Use ground truth labels (teacher forcing) with masked attention during training for parallelization.
Decoder operates autoregressively during inference.
Application: machine translation

Masked attention

	Ich	liebe	dich
Ich		$-\infty$	$-\infty$
liebe			$-\infty$
dich			

Used to prevent the decoder from seeing future input tokens during training for parallelization

Also, used during inference to preserve the auto-regressive property

Set to –infinity after the dot-product and scaling prior to softmax

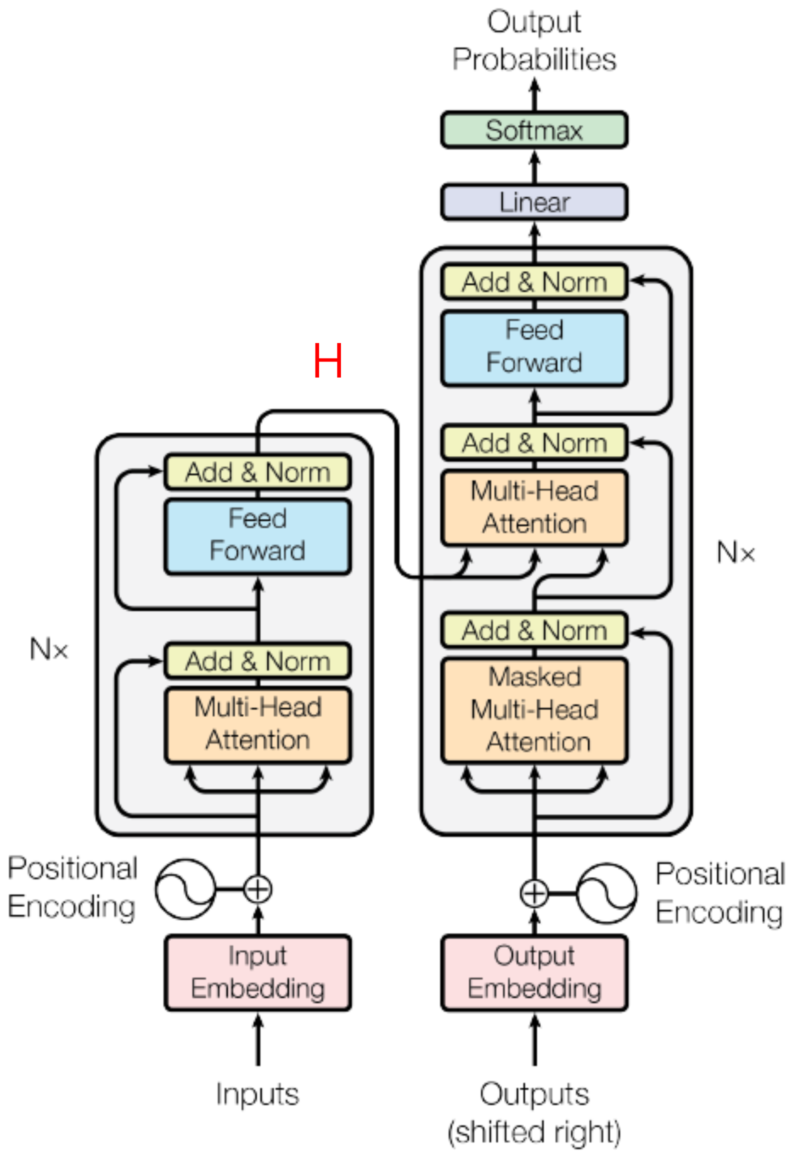
Cross attention in decoder

$H \in R^{T \times d_m}$ encoder vectors

$Z \in R^{T \times d_m}$ decoder vectors

Similar to self-attention,

1. dot-product,
2. Scale
3. (mask)
4. Softmax
5. MatMul

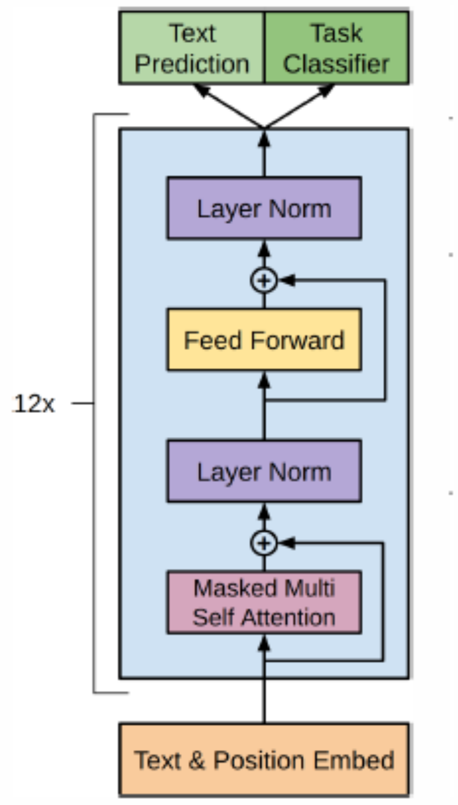


GPTs (Generative pre-trained transformer) from OpenAI

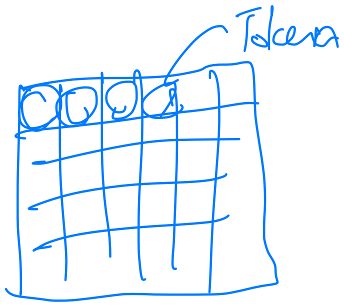
GPT-3 (GPT-2) based on transformer decoder only
(Google BERT: transformer encoder)

Recall:
Given data x and class c,
generative model: $p(x|c)$ likelihood
Discriminative model: $p(c|x)$ posterior

The model was trained to predict the next token in a sequence, given the previous tokens in the sequence. This type of training is common for language models, as it allows the model to learn the relationships between words and phrases.



Datasets used to train GPT-3.



Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

Table 2.2: Datasets used to train GPT-3. "Weight in training mix" refers to the fraction of examples during training that are drawn from a given dataset, which we intentionally do not make proportional to the size of the dataset. As a result, when we train for 300 billion tokens, some datasets are seen up to 3.4 times during training while other datasets are seen less than once.

GPT-3: 175 billion parameters
Training data of 570GB, roughly equivalent to 400 billion byte-pair-encoded tokens

ChatGPT: Improved LLM by human feedback

auto-regressive prediction loss

How to improve pre-trained LLMs?

- Use human feedback to improve/fine-tune LLMs.

- Approach
  - Define a reward model from human judgement
  - Fine-tune an LLM using reinforcement learning (RL) enabled by reward learning

https://www.cs.toronto.edu/~hinton/coursera/lecture1/lec1.pdf

Reinforcement learning (RL) (1/3)

- In RL, the output is an action or sequence of actions and the only supervisory signal is an occasional scalar reward.
  - The goal in selecting each action is to maximize the expected sum of the future rewards.
  - We usually use a discount factor for delayed rewards so that we don't have to look too far into the future.
- RL is difficult:
  - The rewards are typically delayed so its hard to know where we went wrong (or right).
  - A scalar reward does not supply much information.

Reinforcement learning (RL) (2/3)

$R(x, y_1) > R(x, y_2)$

$z_1 : (x, y_1)$ $R(x, y_2)$

$z_2 : (x, y_2)$

Generate an episode τ $t = 0,1, \ldots, T-1$

following policy $\pi_\theta(a_t|s_t)$: (s_0, a_0, r_0) ·□ $(s_{T-1}, a_{T-1}, r_{T-1})$

LLM

At each time t, the agent receives a state s_t and selects an action at from some set of possible actions A according to its policy π, where π is a mapping from states s_t to actions a_t. In return, the agent receives the next state s_{t+1} and receives a scalar reward r_t. The process continues until the agent reaches a terminal state after which the process restarts. The goal of agent is to maximize the expected return from each state s_t:

Define the return $R_t = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k}$

$\max_\theta E[R_t]$

Maximize $E[R_t]$ with respect to the policy parameters θ

By a long derivation, $\nabla_\theta log\pi_\theta(a_t|s_t)(R_t - b_t(s_t))$

Policy optimization in RL (3/3)

$\theta \quad f(x_1, x_2)$

f is growing fast in the direction

Vanilla Policy Gradient

Algorithm 1 "Vanilla" policy gradient algorithm

Initialize policy parameter θ, baseline b — To reduce variance

for iteration$=1, 2, \dots$ **do**

Collect a set of trajectories by executing the current policy

At each timestep in each trajectory, compute

the *return* $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and

the *advantage estimate* $\hat{A}_t = R_t - b(s_t)$.

Re-fit the baseline, by minimizing $\|b(s_t) - R_t\|^2$,

summed over all trajectories and timesteps.

Update the policy, using a policy gradient estimate $\hat{g}$,

which is a sum of terms $\nabla_\theta \log \pi(a_t \mid s_t, \theta)\hat{A}_t$

end for

$R_t \leftarrow Q_\pi$

$b(s_t) \leftarrow V_\pi$

$V_\pi(s) \triangleq E_\pi[R_t \mid S_t = s]$

$Q_\pi(s,a) \triangleq E_\pi[R_t \mid S_t = s, A_t = a]$

$V_\pi(s) = \sum_a \pi(a|s)\, Q_\pi(s,a)$

$\hat{A}_t \dfrac{\nabla_\theta \pi_\theta(A_t|S_t)}{\pi_\theta(A_t|S_t)}$

R_t

$\theta_t \leftarrow \theta_{t-1} + \alpha \nabla_\theta$

~ [Williams, 1992]

REINFORCE

Autoregressive generative model for GPT (Generative pre-trained transformer)

- With a vocabulary Σ, a language model (LM) ρ defines a probability distribution over a sequence of tokens via
- $\rho(x_0 \cdots x_{n-1}) = \prod_{0 \le k < n} \rho(x_k | x_0 \cdots x_{k-1})$ autoregressive
- Apply this model to a task with input space $X = \Sigma^{\le m}$, data distribution D over X and output space $Y = \Sigma^n$. For example, $x \in X$ could be an article of up to 1000 words and $y \in Y$ could be a 100-word summary. ρ defines probabilistic policy for this task via $\rho(y|x) = \frac{\rho(xy)}{\rho(x)}$: fixing the beginning of the sample to x generating subsequent tokens using ρ.
- Trained using cross-entropy loss between the predicted probability distribution and the true probability distribution.

Fine-Tuning Language Models from Human Preferences, 2019

- Explored the use of human preferences between pairs of trajectory segments, showing that they can successfully train complex novel behaviors with about an hour of human time in Deep Reinforcement Learning from Human Preferences, NIPS 2017.

pretrained

- Initialize a policy $\pi_\theta = \rho$ (LM) and then fine-tune π to perform the task well using RL. Defining the task by a reward function $r: X \times Y \to \mathbb{R}$, use RL to directly optimize the expected reward:

$r(x, y_1)$

$$\mathbb{E}_\pi[r] = \mathbb{E}_{x \sim D, y \sim \pi(.|x)}[r(x, y)]$$

- Want to perform tasks defined by human judgments, where we can only learn about the reward by asking humans. To do this, we will first use human labels to train a reward model, and then optimize that reward model.

$e^{r((x,y_1))} > e^{r(x,y_2)}$

Fine-Tuning Language Models from Human Preferences

- Ask human labelers to pick the best response y_i to a given input x. Collect a dataset S of $(x, y_0, y_1, y_2, y_3, b)$ tuples and fit a reward model $r: X \times Y \to \mathbb{R}$ using the loss

$$loss(r) = -\mathbb{E}_{(x,\{y_i\},b)\sim S}\left[log\ \frac{e^{r(x,y_b)}}{\sum_i e^{r(x,y_i)}}\right]$$

decoder

- Since the reward model needs to understand language, initialize it as a random linear function of the final embedding output of the language model policy ρ.

- Now we fine-tune π to optimize the reward model r. To keep π from moving too far from ρ, add a penalty with expectation $\beta\, KL(\pi|\rho)$. Perform RL on the modified reward $R(x,y) = r(x,y) - \beta\ log\ \frac{\pi(y|x)}{\rho(y|x)}$.

$$KL(p|q) = \mathbb{E}_{x\sim p(x)} \log \frac{p(x)}{q(x)} \geq 0$$

Fine-Tuning Language Models from Human Preferences

Overall training process

1. Gather samples (x, y_0, y_1, y_2, y_3) via $x \sim D$, $y_i \sim \rho(\cdot|x)$.
   Ask humans to pick the best y_i from each.

2. Initialize r to pretrained ρ and randomly initialize the final linear layer of r.
   Train r on the human samples using $loss(r)$.

3. At each time step, train π_θ via Proximal Policy Optimization (PPO, 2017)
   with reward the $R(x, y)$ on $x \sim D$ and $y \sim \pi_\theta(\cdot|x)$
   by maximizing $\mathbb{E}_{x\sim D, y\sim\pi_{\theta_{old}}(.|x)} R(x, y)$ with respect to the parameters θ.

Proximal Policy Optimization (PPO, 2017)

- Updates the policy function in a way that increases the probability of taking actions that lead to high rewards. policy optimization

- Uses clipping to ensure that the policy function does not change too much between updates. This helps to stabilize the training process and prevent the policy from becoming too erratic.

- Stability and reliability of trust-region methods but are much simpler to implement.

Proximal Policy Optimization (PPO, 2017)

1. $r_t(\theta) = \dfrac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$

2. $L^{CLIP}(\theta) = \widehat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t]$

3. Each iteration, each of N (parallel) actors collect fixed T timesteps of data. Then, construct the surrogate loss on these NT timesteps of data, and optimize it with minibatch SGD (or usually for better performance, Adam [KB14]), for K epochs.

Proximal Policy Optimization (PPO, 2017)

Algorithm 1 PPO, Actor-Critic Style

---

for iteration$=1, 2, \ldots$ **do**
  for actor$=1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \le NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

Instruct GPT (~ChatGPT)

Training language models to follow instructions with human feedback, NeurIPS 2022

Three steps of Instruct GPT (175B): SFT (Supervised fine-tuning), reward modeling and policy optimization by PPO

Input to the 1^{st} step = policy from a pre-trained LLM

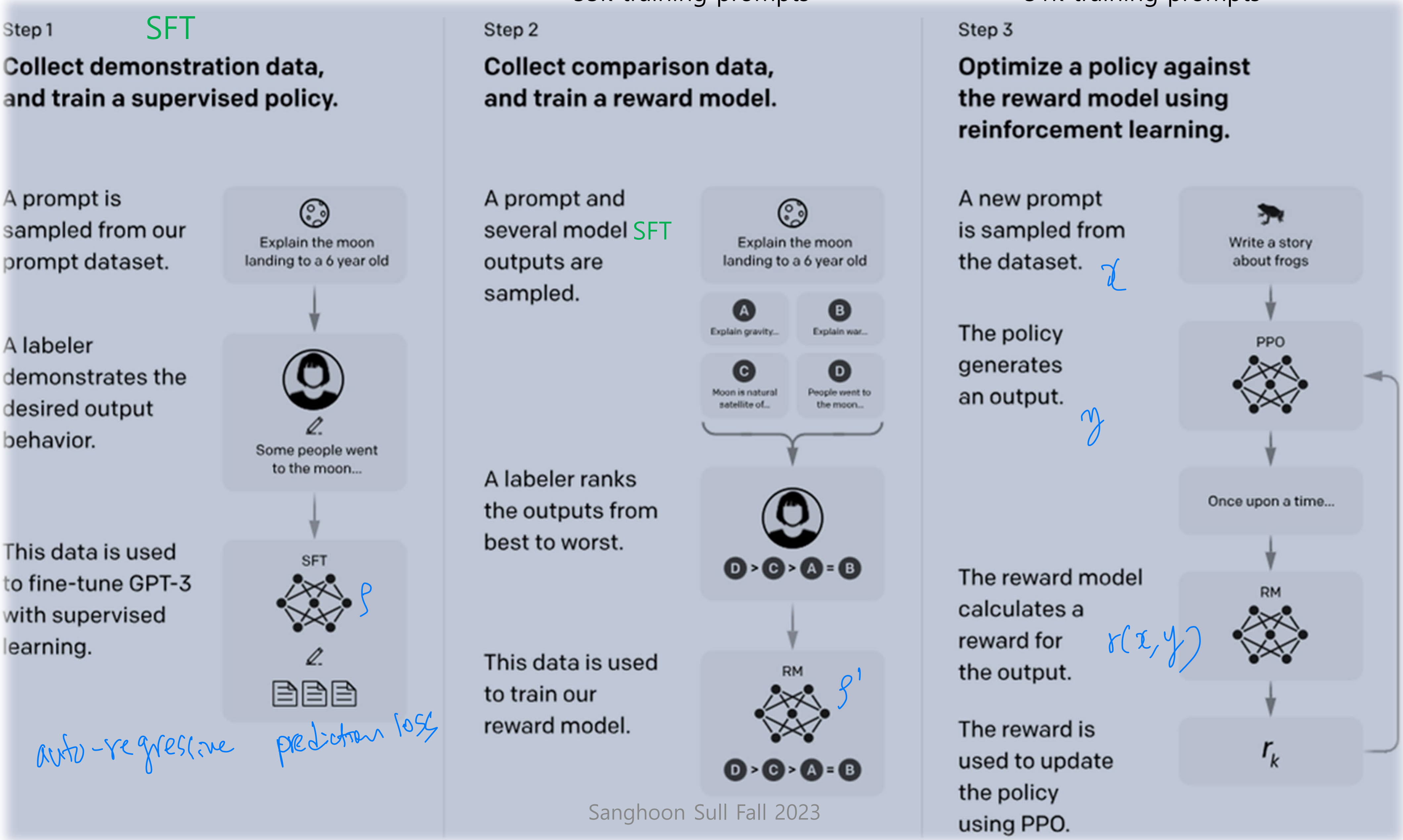
Output from the 3^{rd} step = Improved policy by SFT and PPO.

Fine-tuning updates all the parameters of the pre-trained model.

13k training prompts

33k training prompts

31k training prompts



Step 1: Supervised fine-tuning (SFT) of Instruct GPT

- Fine-tune GPT-3 on our labeler demonstrations using supervised learning.

- Trained for 16 epochs, using a cosine learning rate decay, and residual dropout of 0.2.

- Select final SFT model based on the RM score on the validation set.

Step 2: Reward modeling (RM) of Instruct GPT

- Remove the final unembedding layer (linear head + softmax) for next token prediction from the SFT and add a randomly initialized linear head that outputs a scalar value reward. Only 6B RM used.

- The RM is trained on a dataset of comparisons between two model outputs on the same input. They use a cross-entropy loss, with the comparisons as labels—the difference in rewards represents the log odds that one response will be preferred to the other by a human labeler.

Strep 2: Reward modeling (RM) of Instruct GPT

$1>2>3>4$

$1>2$ $2>4$
$2>3$ $1>4$
$3>4$

Given a prompt x, define the probability of preferring one response/completion y_w over the other y_l as,

$$p(y_w > y_l) = \frac{\exp(r_\theta(y_w))}{\exp(r_\theta(y_w)) + \exp(r_\theta(y_l))} = \sigma(r_\theta(y_w) - r_\theta(y_l)).$$

$\sigma(x) = \frac{1}{1+e^{-x}}$

Use cross-entropy loss function:

$$\mathrm{L} = -1 \times \log p(y_w > y_l) + 0 \times \log p(y_l > y_w).$$

For K responses to rank, the loss function for the reward model is:

$$\mathrm{loss}(\theta) = -\frac{1}{\binom{K}{2}} \mathbb{E}_{(x, y_w, y_l) \sim D}[\log(\sigma(r_\theta(x, y_w) - r_\theta(x, y_l)))]$$

$r_\theta(x, y)$ is the scalar output of the reward model for prompt x and completion y with parameters θ, and D is the dataset of human comparisons.

Note: Since the reward uses the decoder, it is computed autoregressively.

Step 3: RL of Instruct GPT

- Fine-tune the SFT model again on our environment using PPO. The environment is a bandit environment which presents a random customer prompt and expects a response to the prompt. Given the prompt and response, it produces a reward determined by the reward model and ends the episode.

- Add a per-token KL penalty from the SFT model at each token to mitigate overoptimization of the reward model.

- The value function in the advantage estimator of PPO is initialized from the RM.

decoder: π_ϕ^{RL} $r_\theta(x,y)$

π_ϕ^{SFT} value( )

Step 3: RL of Instruct GPT

$$objective(\emptyset) = \mathbb{E}_{(x,y)\sim D_{\pi_\phi^{RL}}}\left[r_\theta(x,y) - \beta \log\left(\frac{\pi_\phi^{RL}(y|x)}{\pi_\phi^{SFT}(y|x)}\right)\right] + $$
$$+ \gamma\, \mathbb{E}_{x\sim D_{pretrain}} \log\left(\pi_\phi^{RL}(x)\right).$$

- Mixed the pretraining gradients into the PPO gradients, in order to fix the performance regressions on public NLP datasets. Then, maximize the above combined objective function in RL training.
- π_ϕ^{RL} computed by summing the log probability rollout is the learned RL policy, π_ϕ^{SFT} is the supervised trained model from Step 1, and $D_{pretrain}$ is the pretraining distribution.

Current Popular LLMs

GPT-4 (OpenAI), BARD (Google AI), LLaMA (Meta AI) and others

Tasks: text generation, translation, question answering, summarization

Used in a wide range of fields, including education, customer service, creative writing, software development

Open source LLMs

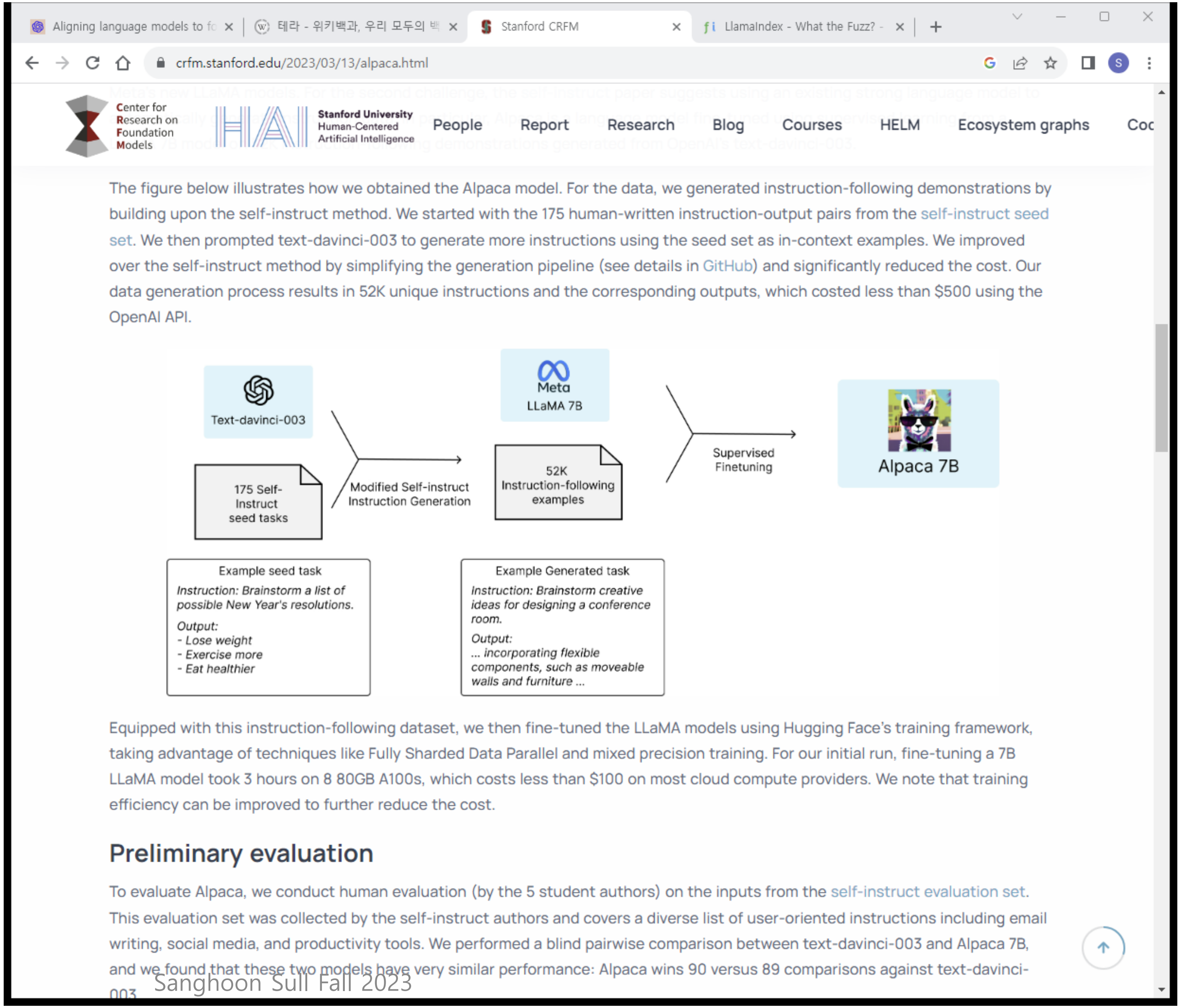
Anthropic

-
-

LLaMA, Alpaca, GPT4All, Vicuna, Flan-T5 and others

Alpaca

https://crfm.stanford.edu/2023/03/13/alpaca.html



Aligning language models to fo | 테라 - 위키백과, 우리 모두의 백 | Stanford CRFM | LlamaIndex - What the Fuzz?

crfm.stanford.edu/2023/03/13/alpaca.html

Center for Research on Foundation Models | HAI Stanford University Human-Centered Artificial Intelligence | People | Report | Research | Blog | Courses | HELM | Ecosystem graphs | Coc

The figure below illustrates how we obtained the Alpaca model. For the data, we generated instruction-following demonstrations by building upon the self-instruct method. We started with the 175 human-written instruction-output pairs from the self-instruct seed set. We then prompted text-davinci-003 to generate more instructions using the seed set as in-context examples. We improved over the self-instruct method by simplifying the generation pipeline (see details in GitHub) and significantly reduced the cost. Our data generation process results in 52K unique instructions and the corresponding outputs, which costed less than $500 using the OpenAI API.

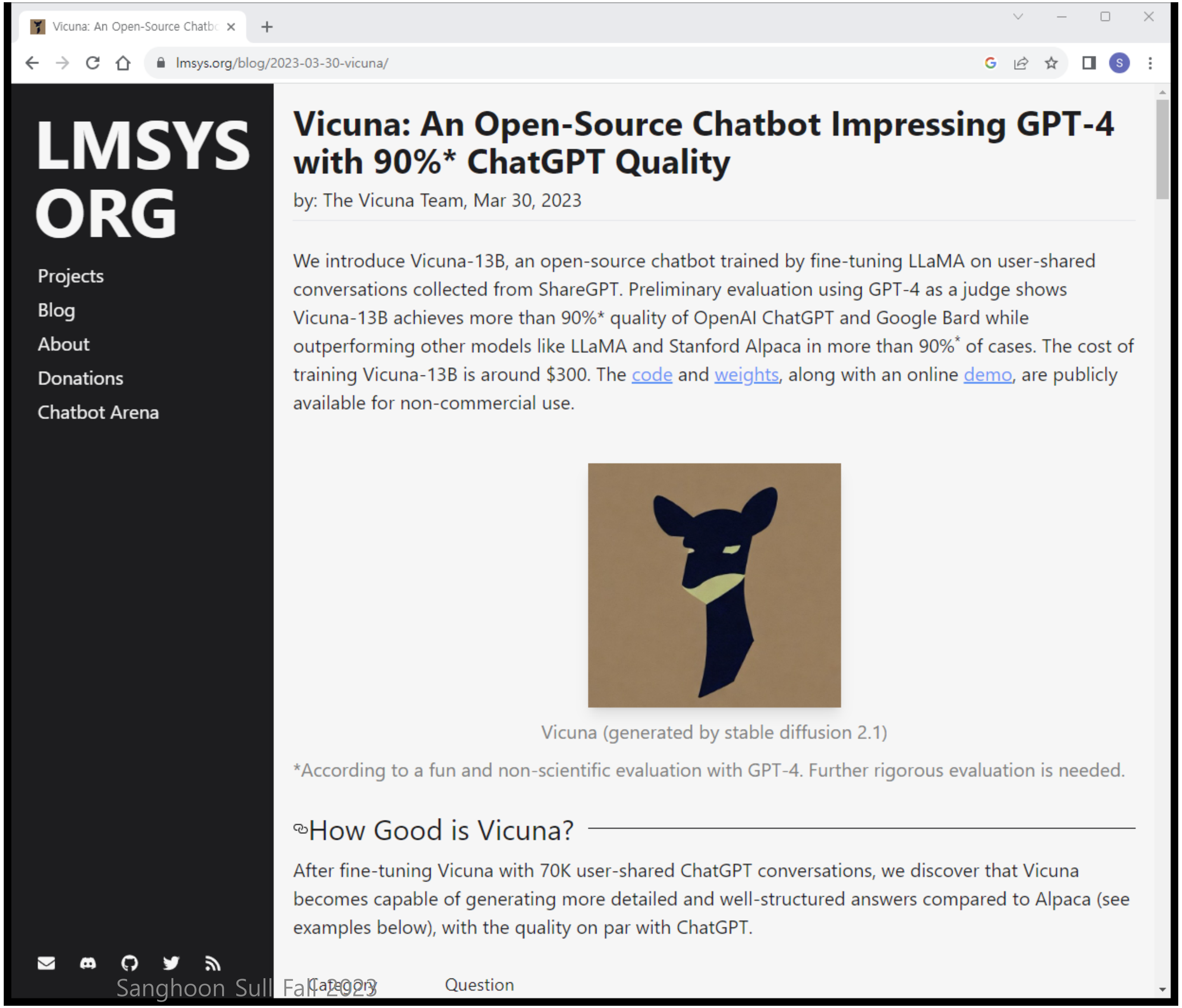
Equipped with this instruction-following dataset, we then fine-tuned the LLaMA models using Hugging Face's training framework, taking advantage of techniques like Fully Sharded Data Parallel and mixed precision training. For our initial run, fine-tuning a 7B LLaMA model took 3 hours on 8 80GB A100s, which costs less than $100 on most cloud compute providers. We note that training efficiency can be improved to further reduce the cost.

Preliminary evaluation

To evaluate Alpaca, we conduct human evaluation (by the 5 student authors) on the inputs from the self-instruct evaluation set. This evaluation set was collected by the self-instruct authors and covers a diverse list of user-oriented instructions including email writing, social media, and productivity tools. We performed a blind pairwise comparison between text-davinci-003 and Alpaca 7B, and we found that these two models have very similar performance: Alpaca wins 90 versus 89 comparisons against text-davinci-003

Vicuna

https://lmsys.org/blog/2023-03-30-vicuna/



Summary

- Transformer: Efficiency at modeling long-range dependencies in text, scalable and relatively easy to implement and train.
- Improving LLM from human feedback: Use RL.
- Locally fine-tuned LLMs
- Generative models for images

References

1. Transformer: Attention Is All You Need, NIPS 2017
2. http://jalammar.github.io/illustrated-transformer/
3. GPT3: Language Models are Few-Shot Learners, NeurIPS 2020
4. https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/slides/cs224n-2021-lecture09-transformers.pdf
5. R. Sutton and A. Barto, Reinforcement Learning: An Introduction, MIT Press, Cambridge, 2018
6. Training language models to follow instructions with human feedback, NeurIPS 2022 (InstructGPT)
7. Deep Reinforcement Learning from Human Preferences, NIPS 2017
8. Fine-Tuning Language Models from Human Preferences, arXiv 2019
9. Proximal Policy Optimization Algorithms, arXiv 2017