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

Encoder
Decoder

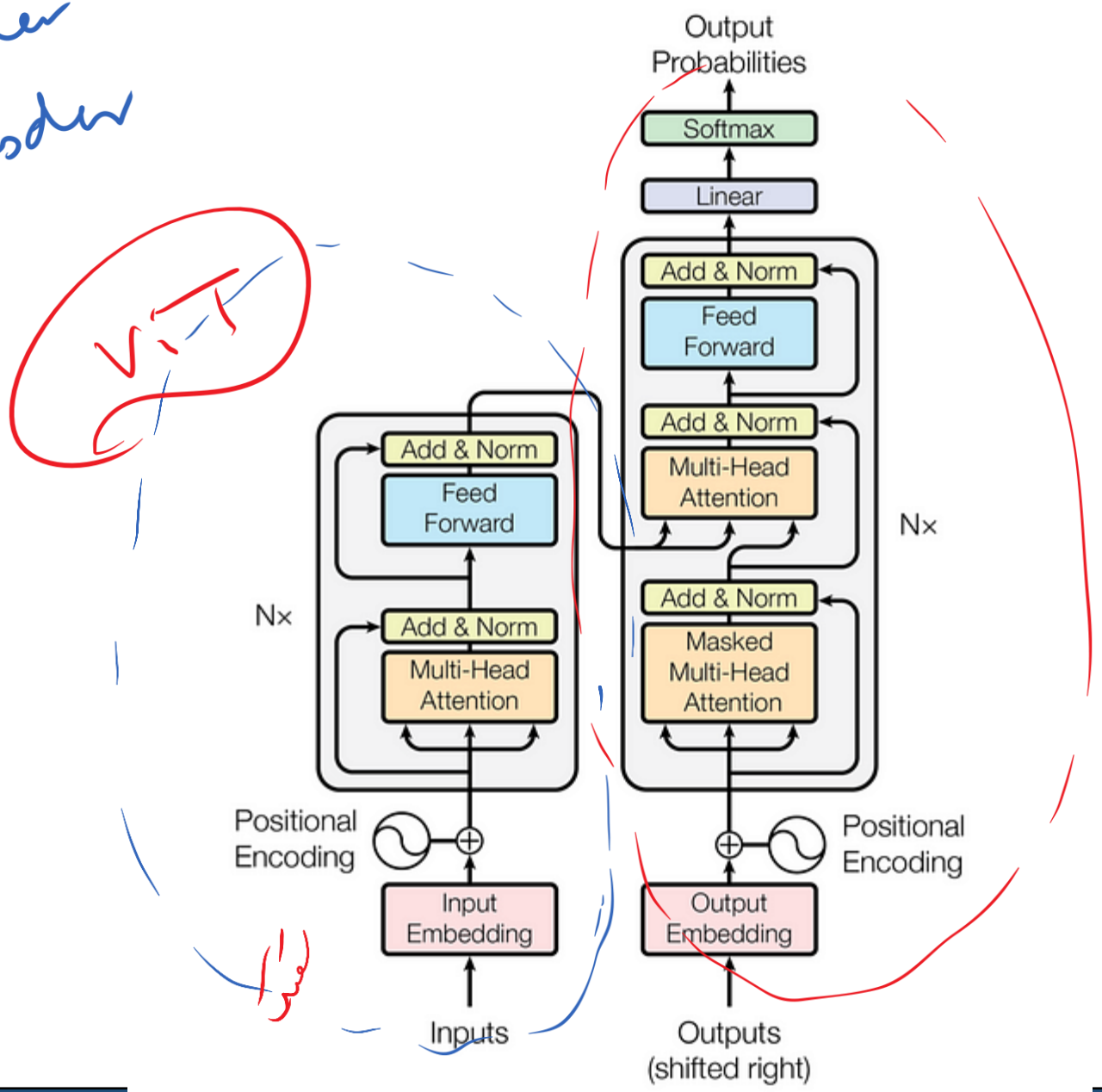
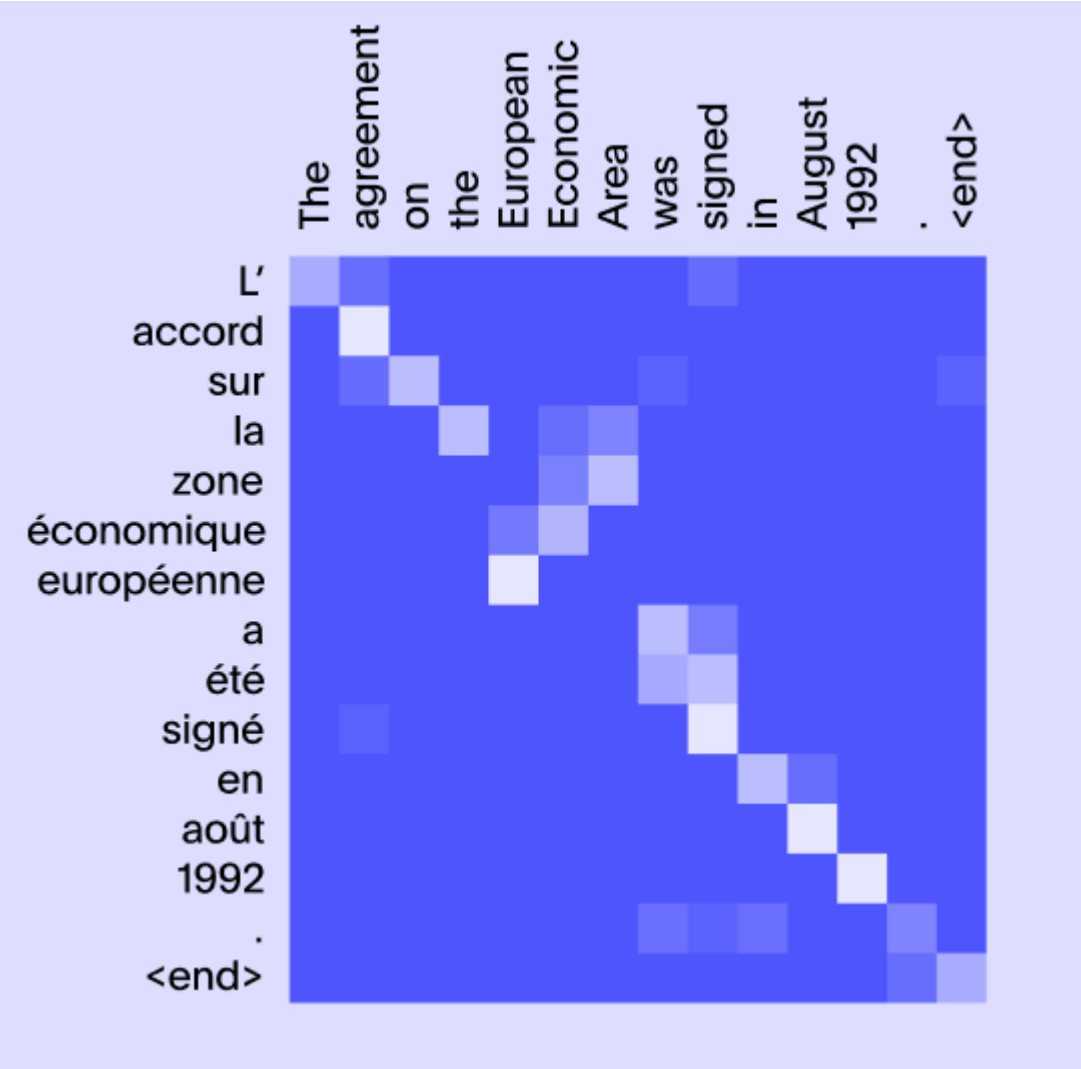


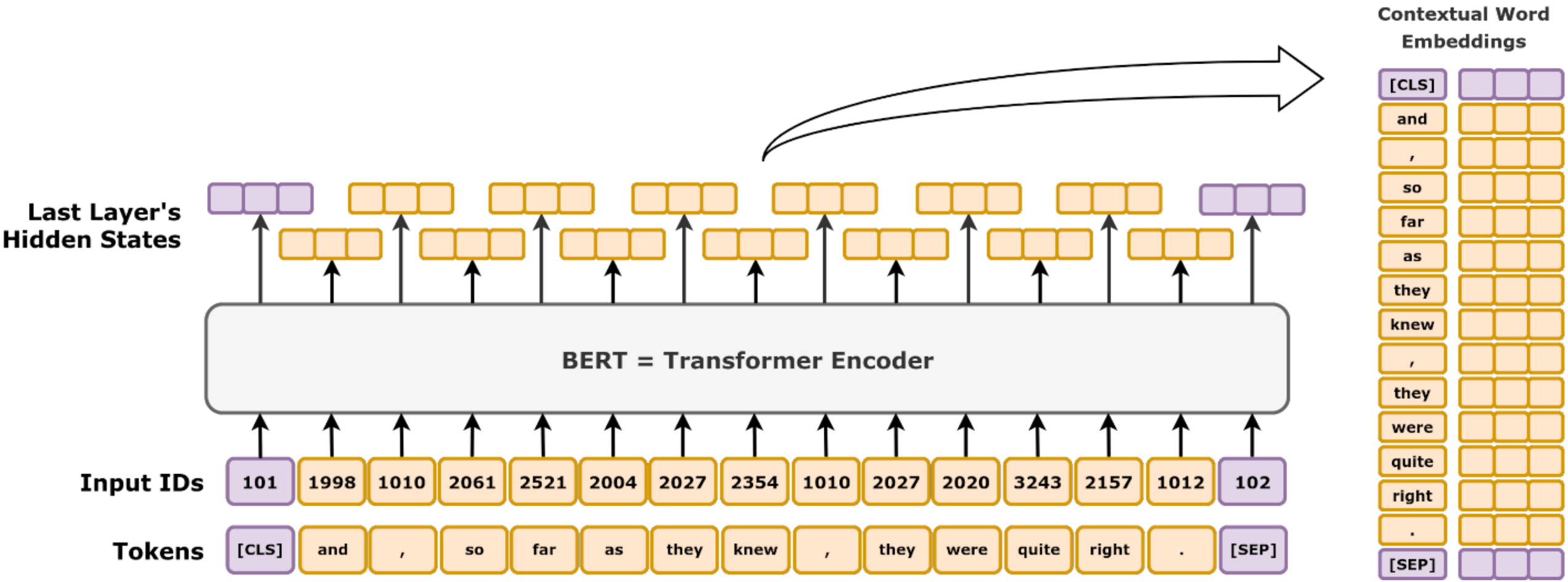
Figure 1: The Transformer - model architecture.

First Task

- 2017
- English-French Translation
- 213 M parameters
- 3.5 days Training by 8 GPUs



BERT (Bidirectional Encoder Representations from Transformers)

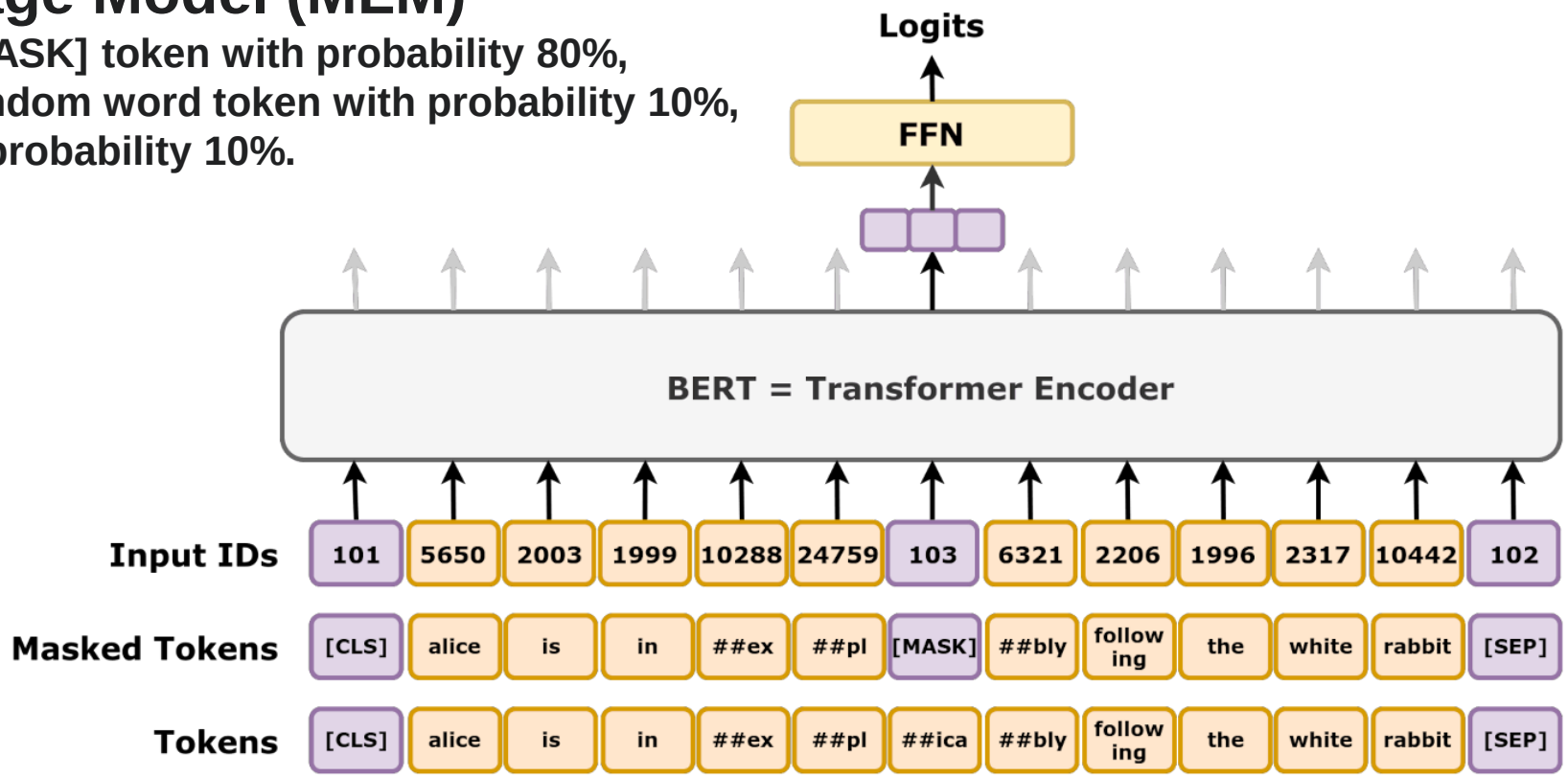


BERT (Bidirectional Encoder Representations from Transformers)

- 2018
  - Classification and Question Answering
- 213 M parameters
- 4 days Training by 64 TPUs

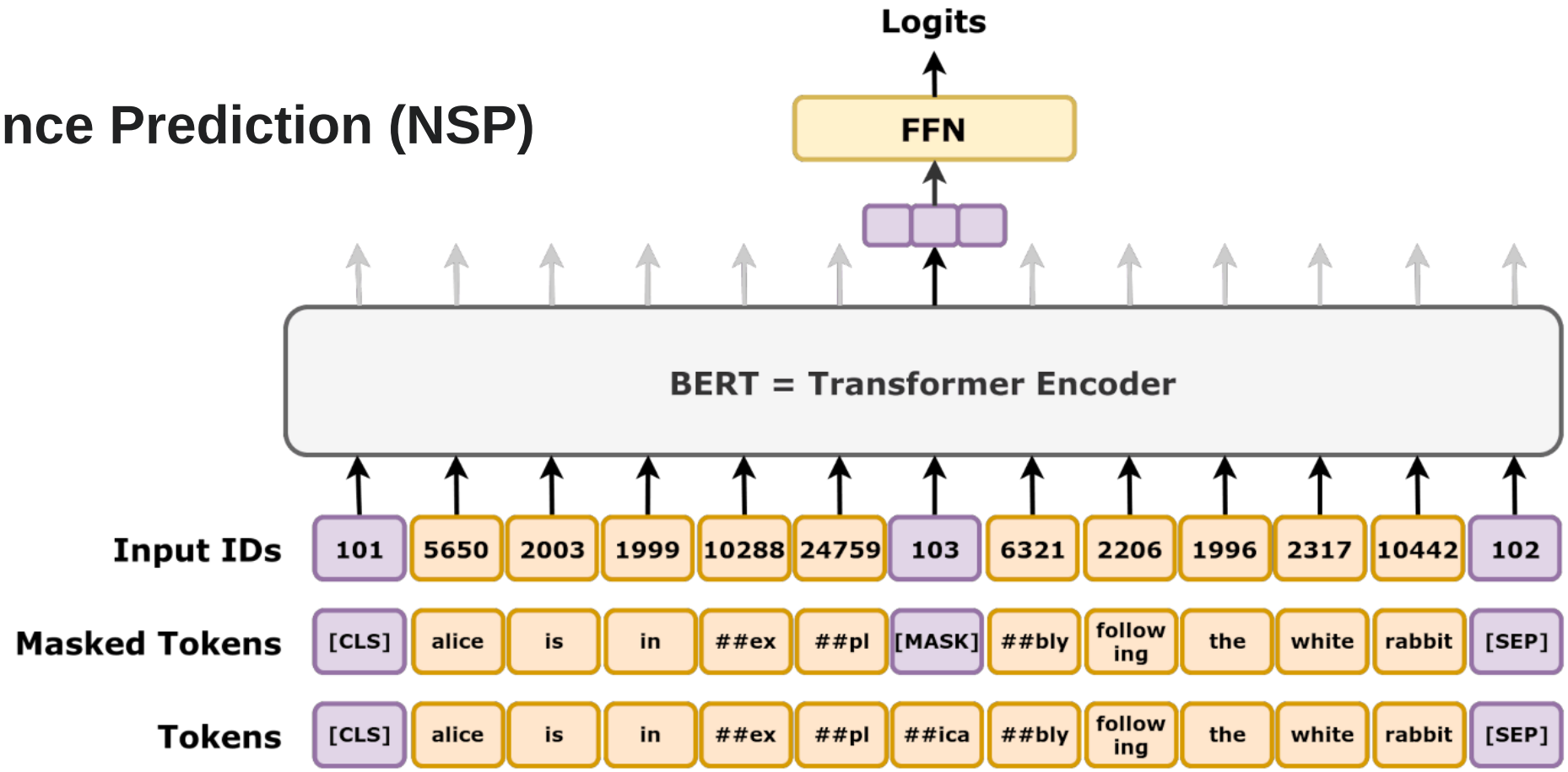
BERT (Bidirectional Encoder Representations from Transformers)

- **Training**:
  - **Masked Language Model (MLM)**
    - **replaced with a [MASK] token with probability 80%,**
    - **replaced with a random word token with probability 10%,**
    - **not replaced with probability 10%.**



BERT (Bidirectional Encoder Representations from Transformers)

- **Training**:
  - **Next Sentence Prediction (NSP)**



BERT (Bidirectional Encoder Representations from Transformers)

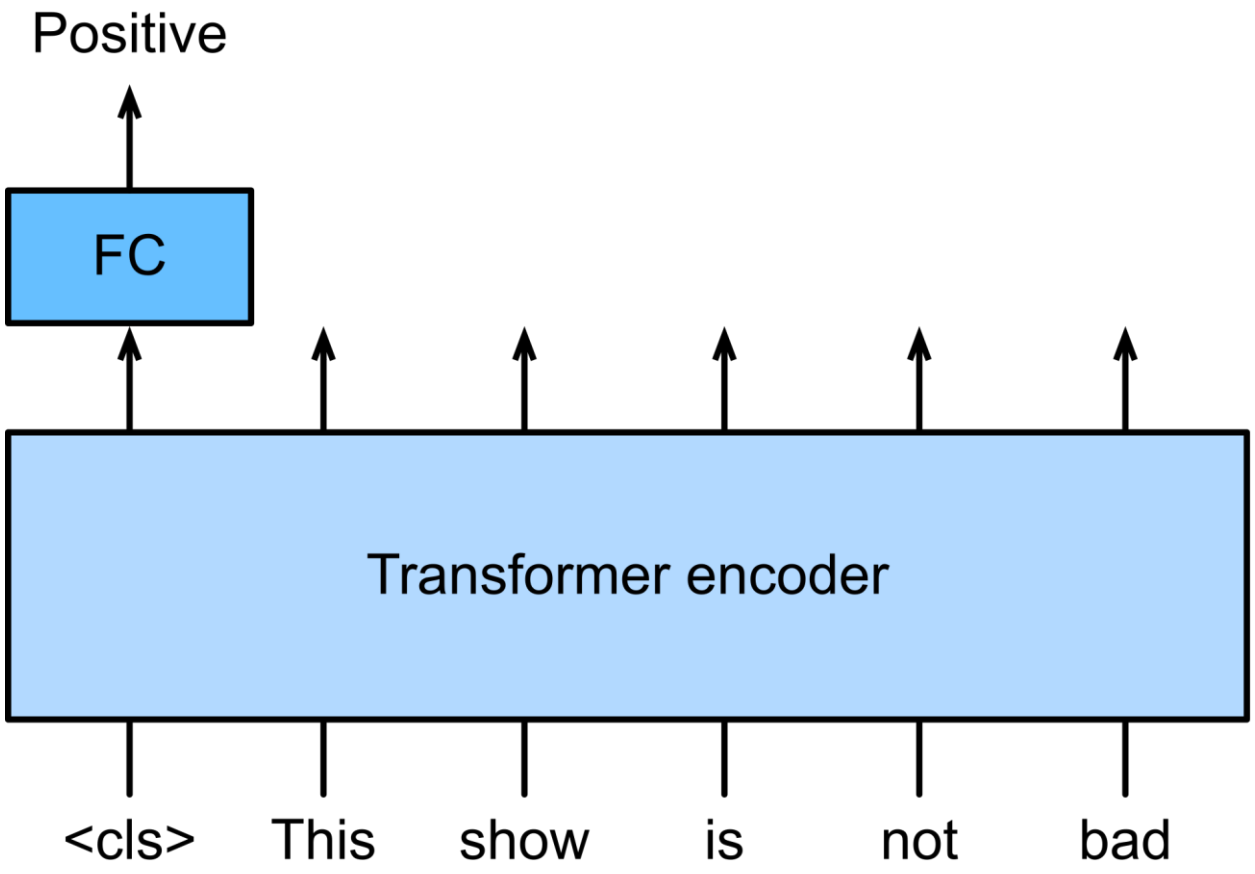
- **Training**:
  - **Next Sentence Prediction (NSP)**

Input = [CLS] the man went to [MASK] store [SEP]
he bought a gallon [MASK] milk [SEP]
Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP]
penguin [MASK] are flight ##less birds [SEP]
Label = NotNext

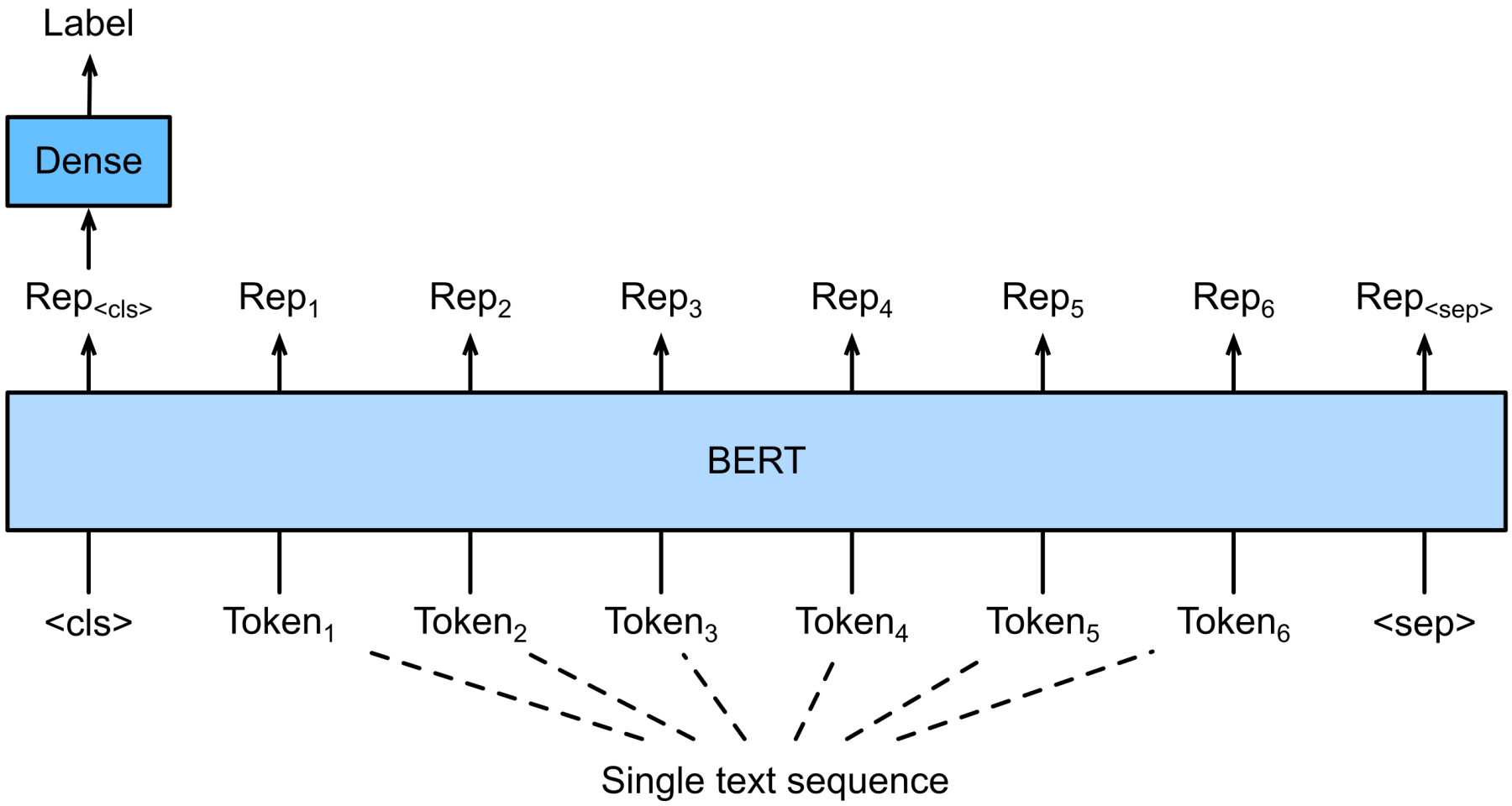
BERT (Bidirectional Encoder Representations from Transformers)

Finetuned tasks for BERT Sentiment Analysis



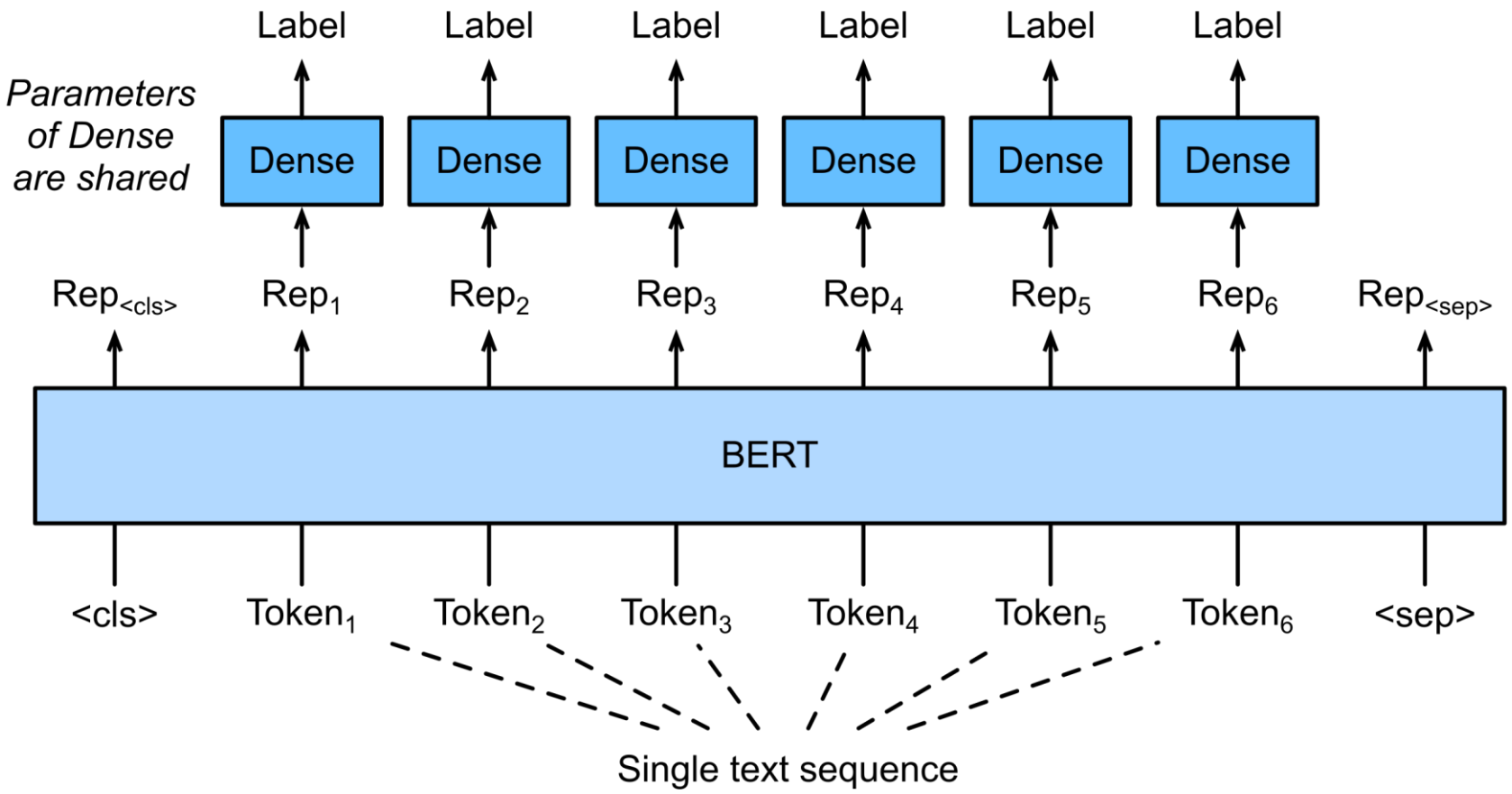
BERT (Bidirectional Encoder Representations from Transformers)

Finetuned tasks for BERT Sentence Classification



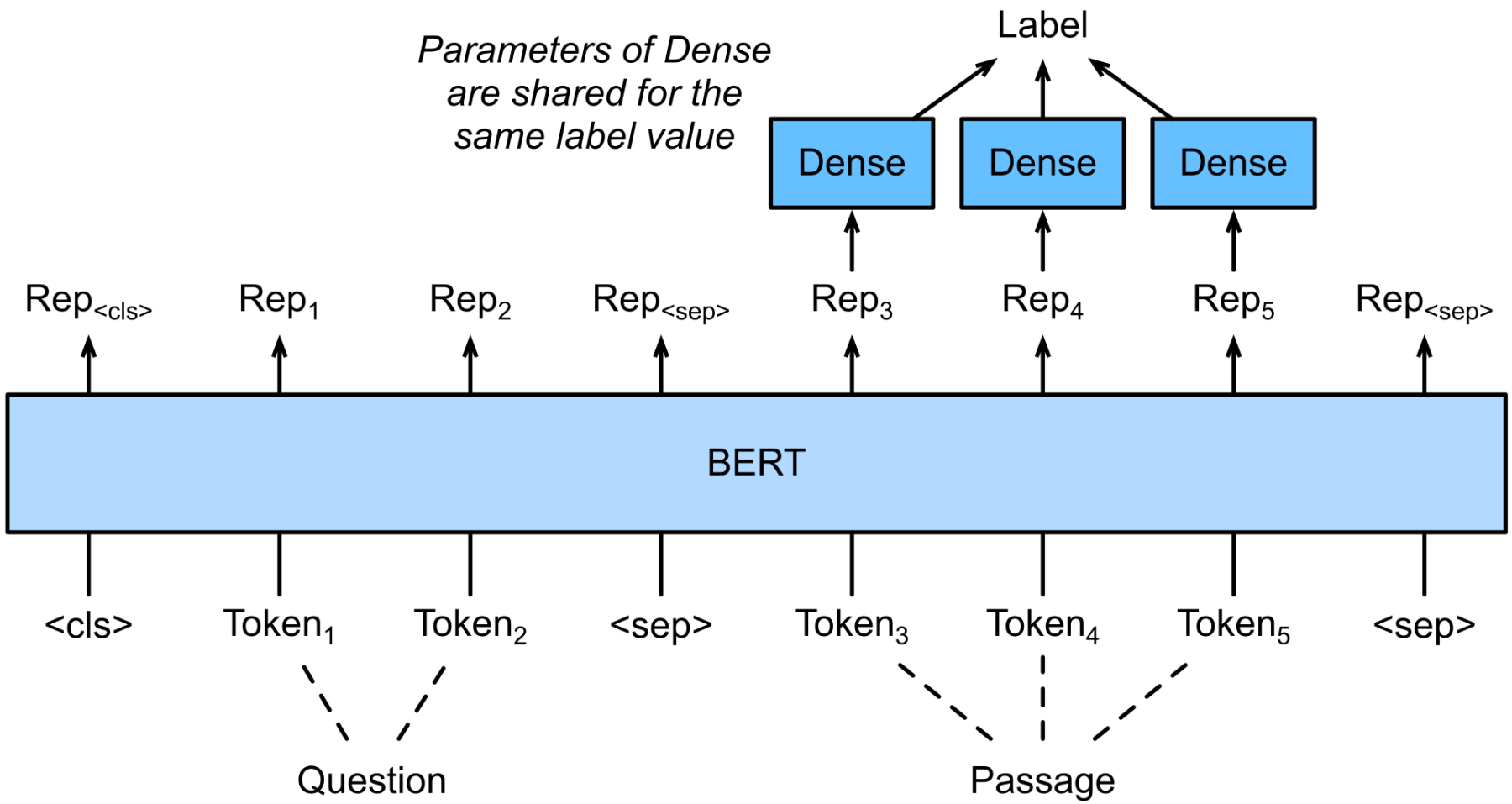
BERT (Bidirectional Encoder Representations from Transformers)

Finetuned tasks for BERT Part of speech tagging



BERT (Bidirectional Encoder Representations from Transformers)

Finetuned tasks for BERT Answering multiple-choice questions

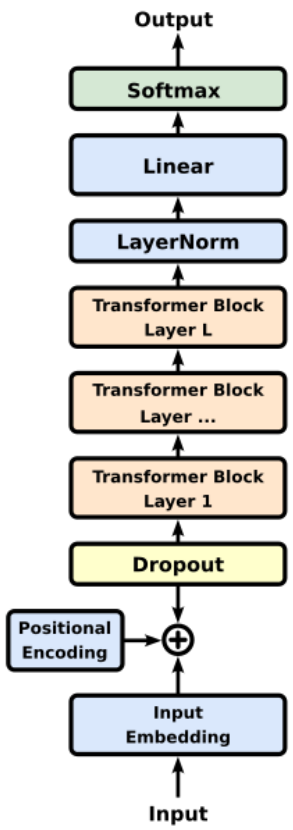


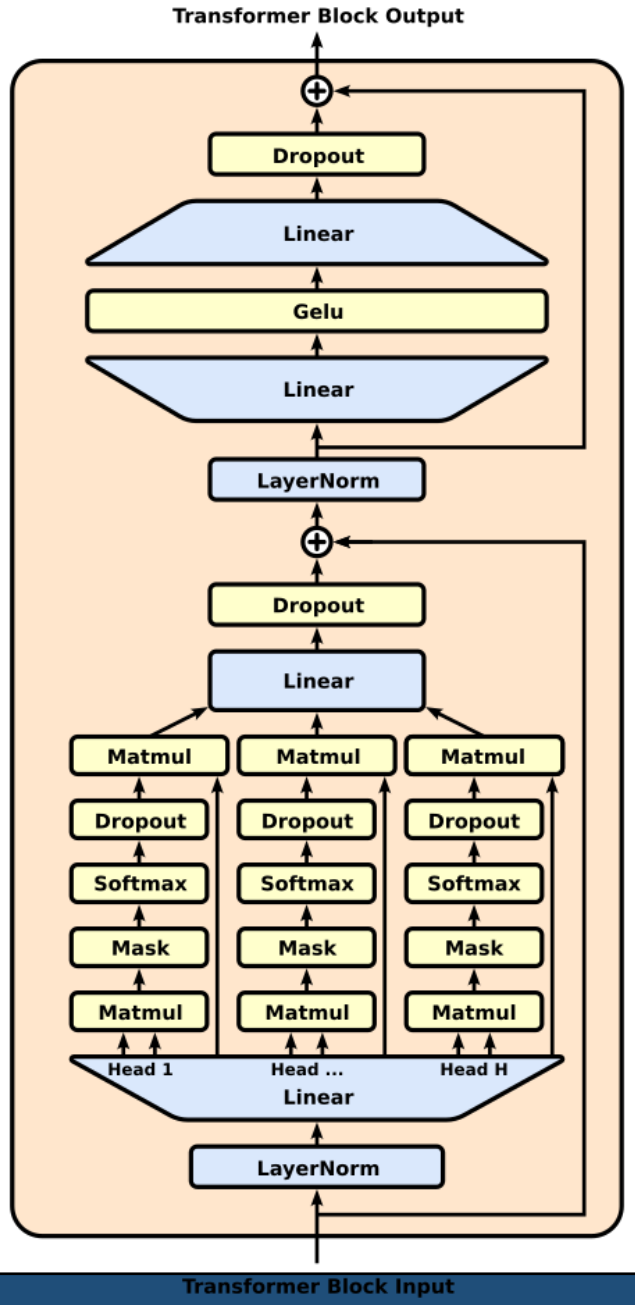
GPTs

OpenAI's *GPT-n* series

Model	Architecture	Parameter count	Training data	Release date	Training cost
GPT-1	12-level, 12-headed Transformer decoder (no encoder), followed by linear-softmax.	117 million	BookCorpus:[39] 4.5 GB of text, from 7,000 unpublished books of various genres.	June 11, 2018[9]	30 days on 8 P600 graphics cards, or 1 petaFLOPS-day.[9]
GPT-2	GPT-1, but with modified normalization	1.5 billion	WebText: 40 GB of text, 8 million documents, from 45 million webpages upvoted on Reddit.	February 14, 2019 (initial/limited version) and November 5, 2019 (full version)[40]	"tens of petaflop/s-day",[41] or 1.5e21 FLOPS.[42]
GPT-3	GPT-2, but with modification to allow larger scaling	175 billion[43]	499 billion tokens consisting of CommonCrawl (570 GB), WebText, English Wikipedia, and two books corpora (Books1 and Books2).	May 28, 2020[41]	3640 petaflop/s-day (Table D.1[41]), or 3.1e23 FLOPS.[42]
GPT-3.5	Undisclosed	175 billion[43]	Undisclosed	March 15, 2022	Undisclosed
GPT-4	Also trained with both text prediction and RLHF; accepts both text and images as input. Further details are not public.[38]	Undisclosed. Estimated 1.7 trillion.[44]	Undisclosed	March 14, 2023	Undisclosed. Estimated 2.1 × 10^{25} FLOPS.[42]

GPT (GPT1)


Output
Softmax
Linear
LayerNorm
Transformer Block Layer L
Transformer Block Layer ...
Transformer Block Layer 1
Dropout
Positional Encoding
Input Embedding
Input


Transformer Block Output
Dropout
Linear
Gelu
Linear
LayerNorm
Dropout
Linear
Matmul
Matmul
Matmul
Dropout
Dropout
Dropout
Softmax
Softmax
Softmax
Mask
Mask
Mask
Matmul
Matmul
Matmul
Head 1
Head ...
Head H
Linear
LayerNorm
Transformer Block Input

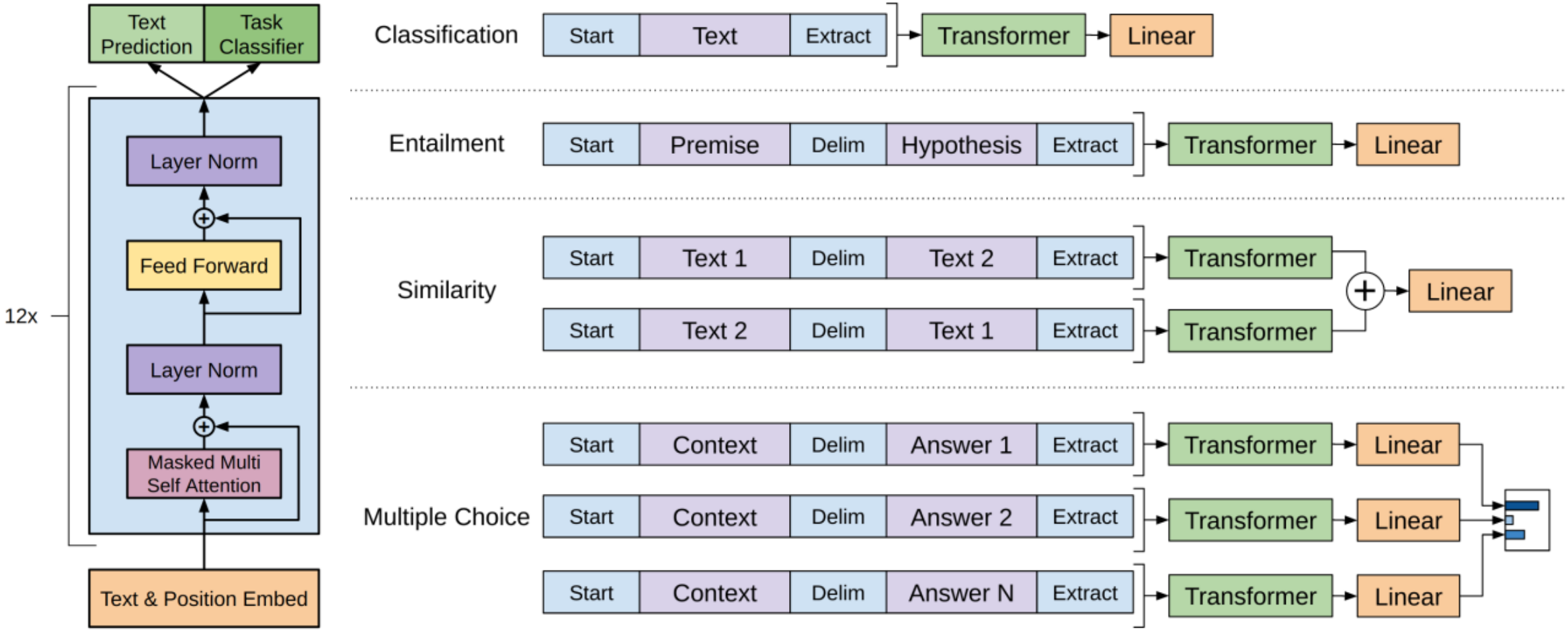
GPT (GPT1)

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

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

where $U = (u_{-k}, \cdots, u_{-1})$ is the context token vector, h_0 is the mapping projection of U, W_c is the token embedding and W_p is the position embedding.

GPT (GPT1) tasks



3.1 Unsupervised pre-training

Given an unsupervised corpus of tokens $\mathcal{U} = \{u_1, \ldots, u_n\}$, we use a standard language modeling objective to maximize the following likelihood:

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

where k is the size of the context window, and the conditional probability P is modeled using a neural network with parameters Θ. These parameters are trained using stochastic gradient descent [51].

In our experiments, we use a multi-layer *Transformer decoder* [34] for the language model, which is a variant of the transformer [62]. This model applies a multi-headed self-attention operation over the input context tokens followed by position-wise feedforward layers to produce an output distribution over target tokens:

$$\begin{aligned} h_0 &= UW_e + W_p \\ h_l &= \texttt{transformer_block}(h_{l-1}) \forall i \in [1, n] \\ P(u) &= \texttt{softmax}(h_n W_e^T) \end{aligned} \tag{2}$$

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

3.2 Supervised fine-tuning

After training the model with the objective in Eq. 1, we adapt the parameters to the supervised target task. We assume a labeled dataset $\mathcal{C}$, where each instance consists of a sequence of input tokens, $x^1, \ldots, x^m$, along with a label y. The inputs are passed through our pre-trained model to obtain the final transformer block's activation h_l^m, which is then fed into an added linear output layer with parameters W_y to predict y:

$$P(y|x^1, \ldots, x^m) = \texttt{softmax}(h_l^m W_y). \tag{3}$$

This gives us the following objective to maximize:

$$L_2(\mathcal{C}) = \sum_{(x,y)} \log P(y|x^1, \ldots, x^m). \tag{4}$$

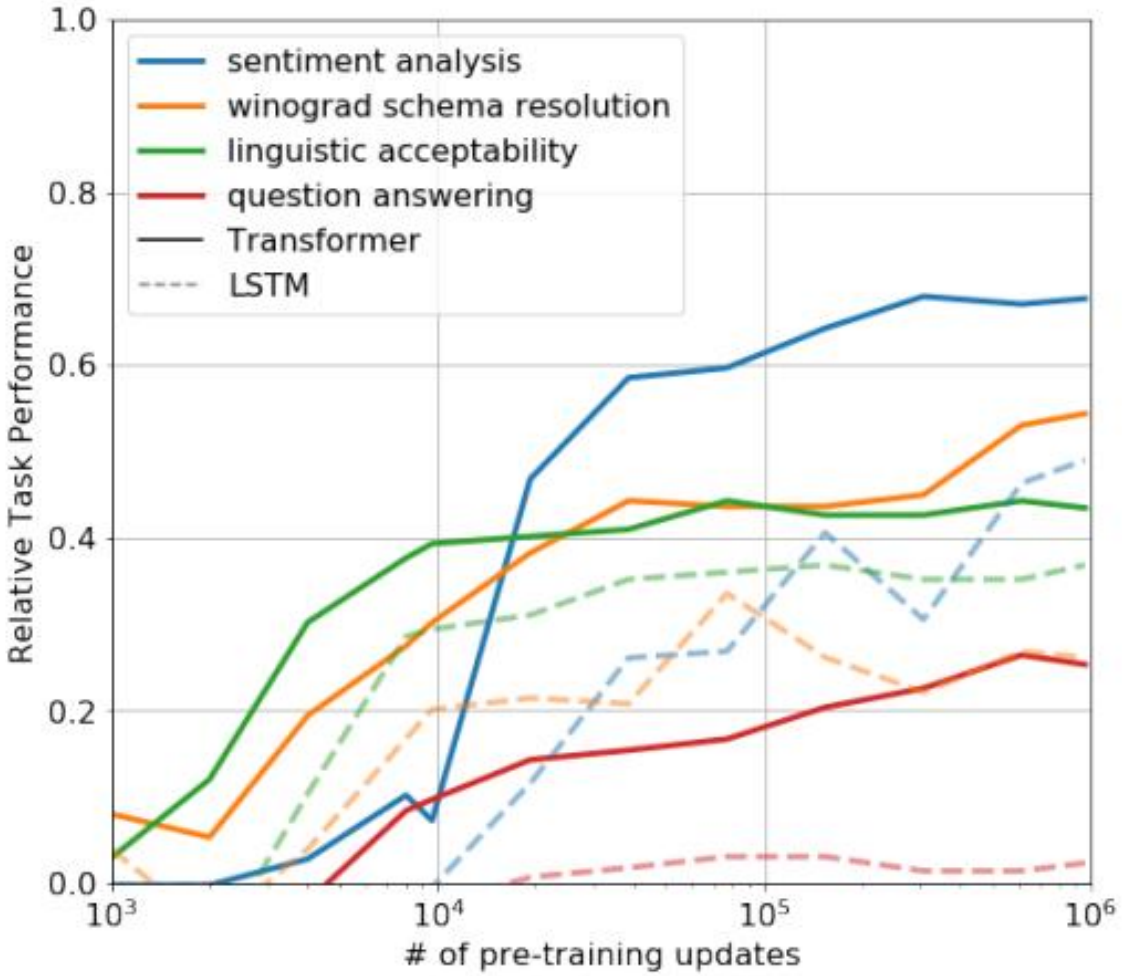
We additionally found that including language modeling as an auxiliary objective to the fine-tuning helped learning by (a) improving generalization of the supervised model, and (b) accelerating convergence. This is in line with prior work [50, 43], who also observed improved performance with such an auxiliary objective. Specifically, we optimize the following objective (with weight λ):

$$L_3(\mathcal{C}) = L_2(\mathcal{C}) + \lambda * L_1(\mathcal{C}) \tag{5}$$

```
from transformers import pipeline
generator = pipeline(task="text-generation", model="openai-community/gpt", torch_dtype=torch.float16, device=0)


result = generator("The future of AI is", max_length=50, do_sample=True)

print(result)
```


sentiment analysis
winograd schema resolution
linguistic acceptability
question answering
Transformer
LSTM
Relative Task Performance
# of pre-training updates
1.0
0.8
0.6
0.4
0.2
0.0
10^3
10^4
10^5
10^6

	architecture	parameter count	training data
GPT-1	12-level, 12-headed Transformer decoder (no encoder), followed by linear-softmax.	0.12 billion	BookCorpus:[38] 4.5 GB of text, from 7000 unpublished books of various genres.
GPT-2	GPT-1, but with modified normalization	1.5 billion	WebText: 40 GB[26] of text, 8 million documents, from 45 million webpages upvoted on Reddit.
GPT-3	GPT-2, but with modification to allow larger scaling.	175 billion	570 GB plaintext, 300 billion tokens of CommonCrawl, WebText, English Wikipedia, and two books corpora (Books1 and Books2).

- **GPT2** :
  - **Normalization after attention and feedforward :**
    - **Stability in training, can increase lr**

  - **Increase sequence length from 512 to 1024**
  - **Better work in few-shot or zeros-shot tasks**

- **GPT3** :
  - **Changes in training**
  - **Increase sequence length from 1024 to 2048**

GPT (GPT3) Training

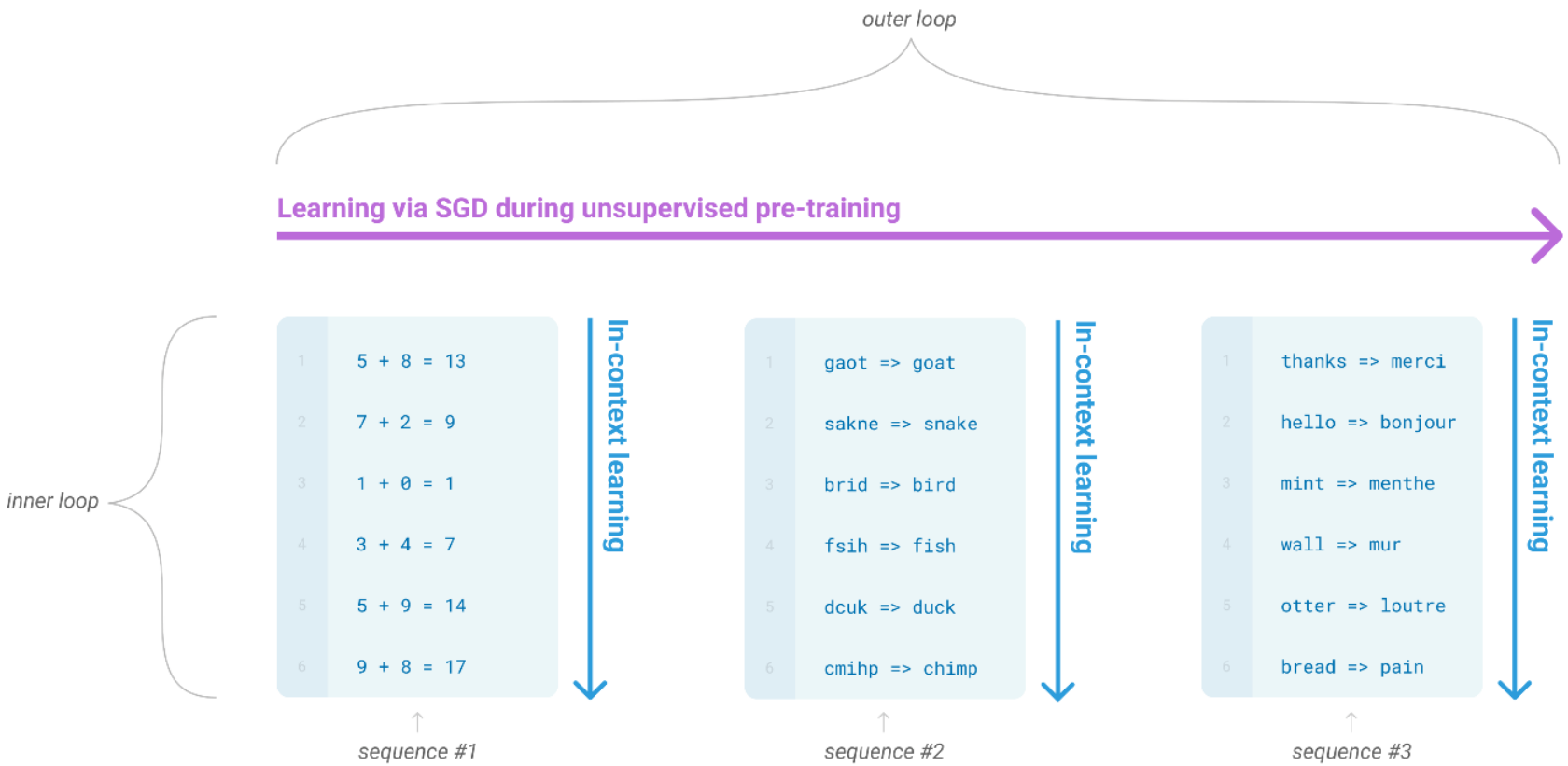


Figure 1.1: Language model meta-learning. During unsupervised pre-training, a language model develops a broad set of skills and pattern recognition abilities. It then uses these abilities at inference time to rapidly adapt to or recognize the desired task. We use the term "in-context learning" to describe the inner loop of this process, which occurs within the forward-pass upon each sequence. The sequences in this diagram are not intended to be representative of the data a model would see during pre-training, but are intended to show that there are sometimes repeated sub-tasks embedded within a single sequence.

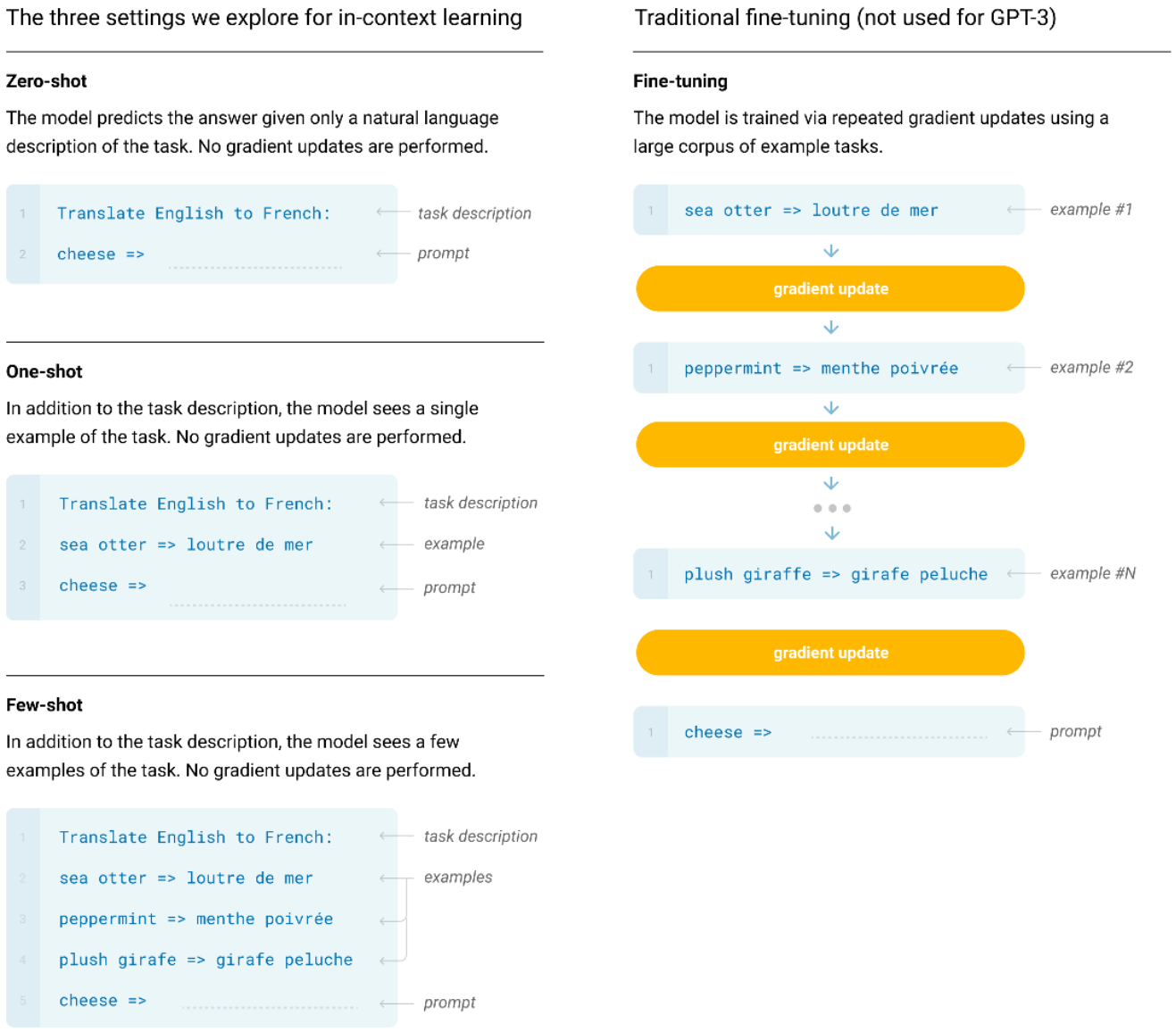


Figure 2.1: Zero-shot, one-shot and few-shot, contrasted with traditional fine-tuning. The panels above show four methods for performing a task with a language model – fine-tuning is the traditional method, whereas zero-, one-, and few-shot, which we study in this work, require the model to perform the task with only forward passes at test time. We typically present the model with a few dozen examples in the few shot setting. Exact phrasings for all task descriptions, examples and prompts can be found in Appendix G.

GPT (GPT3) paper

Language Models are Few-Shot Learners

Tom B. Brown* Benjamin Mann* Nick Ryder* Melanie Subbiah*

Jared Kaplan† Prafulla Dhariwal Arvind Neelakantan Pranav Shyam Girish Sastry

Amanda Askell Sandhini Agarwal Ariel Herbert-Voss Gretchen Krueger Tom Henighan

Rewon Child Aditya Ramesh Daniel M. Ziegler Jeffrey Wu Clemens Winter

Christopher Hesse Mark Chen Eric Sigler Mateusz Litwin Scott Gray

Benjamin Chess Jack Clark Christopher Berner

Sam McCandlish Alec Radford Ilya Sutskever Dario Amodei

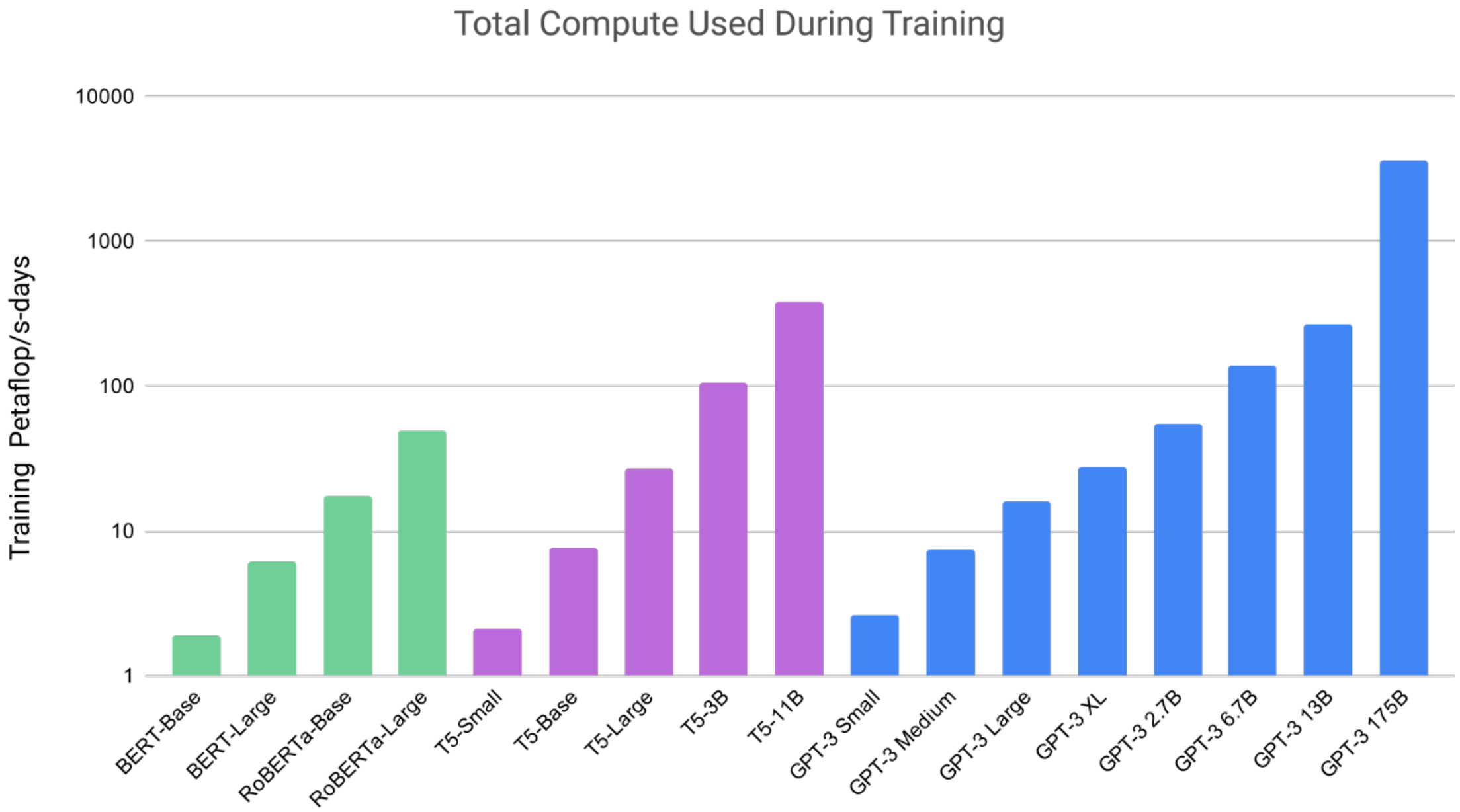
OpenAI

[cs.CL] 22 Jul 2020

GPT (GPT3) Models

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

Table 2.1: Sizes, architectures, and learning hyper-parameters (batch size in tokens and learning rate) of the models which we trained. All models were trained for a total of 300 billion tokens.


Total Compute Used During Training
Training Petaflop/s-days
10000
1000
100
10
1
BERT-Base
BERT-Large
RoBERTa-Base
RoBERTa-Large
T5-Small
T5-Base
T5-Large
T5-3B
T5-11B
GPT-3 Small
GPT-3 Medium
GPT-3 Large
GPT-3 XL
GPT-3 2.7B
GPT-3 6.7B
GPT-3 13B
GPT-3 175B

GPT (GPT3) datasets

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.

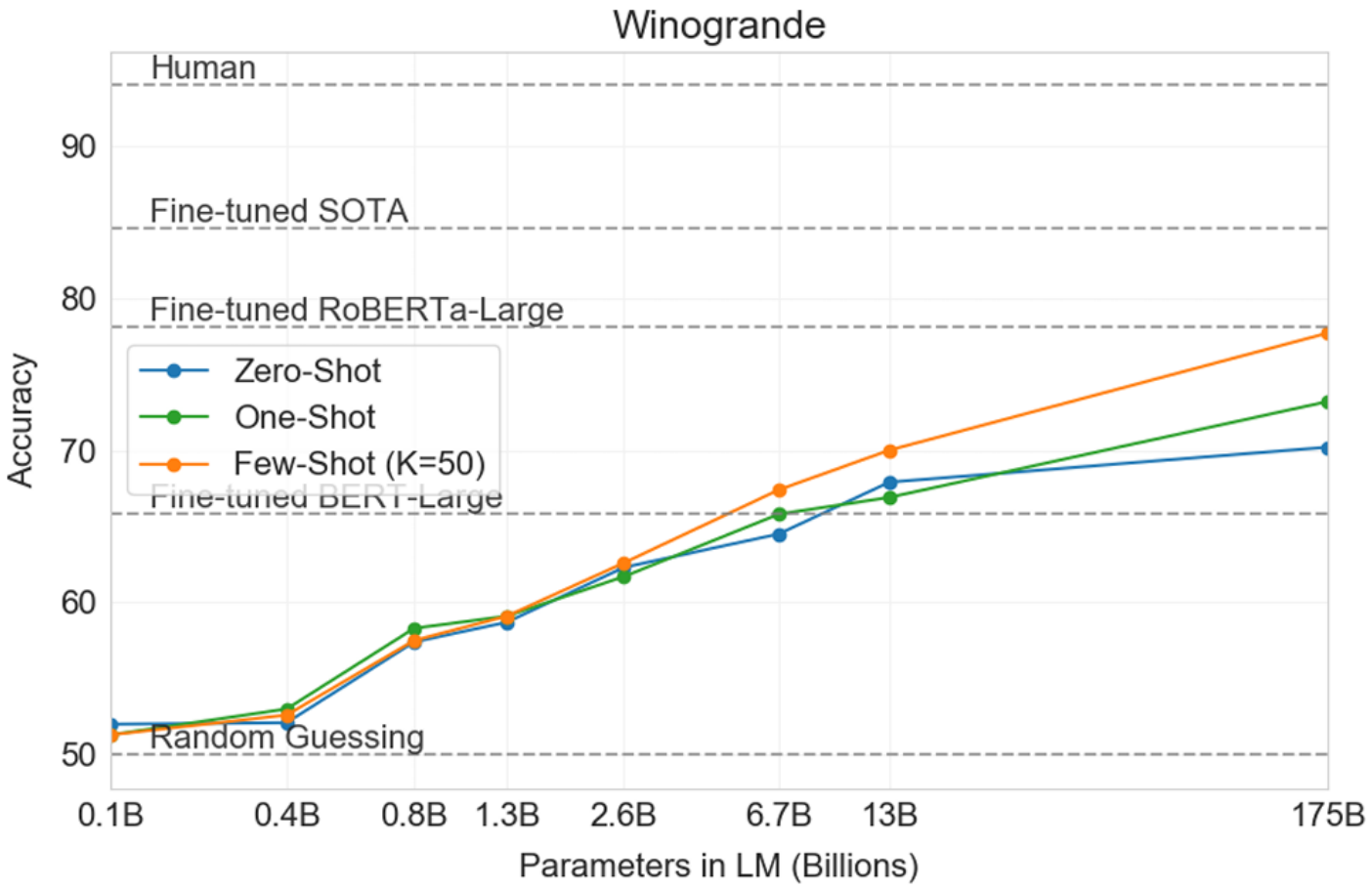
GPT (GPT3) Results

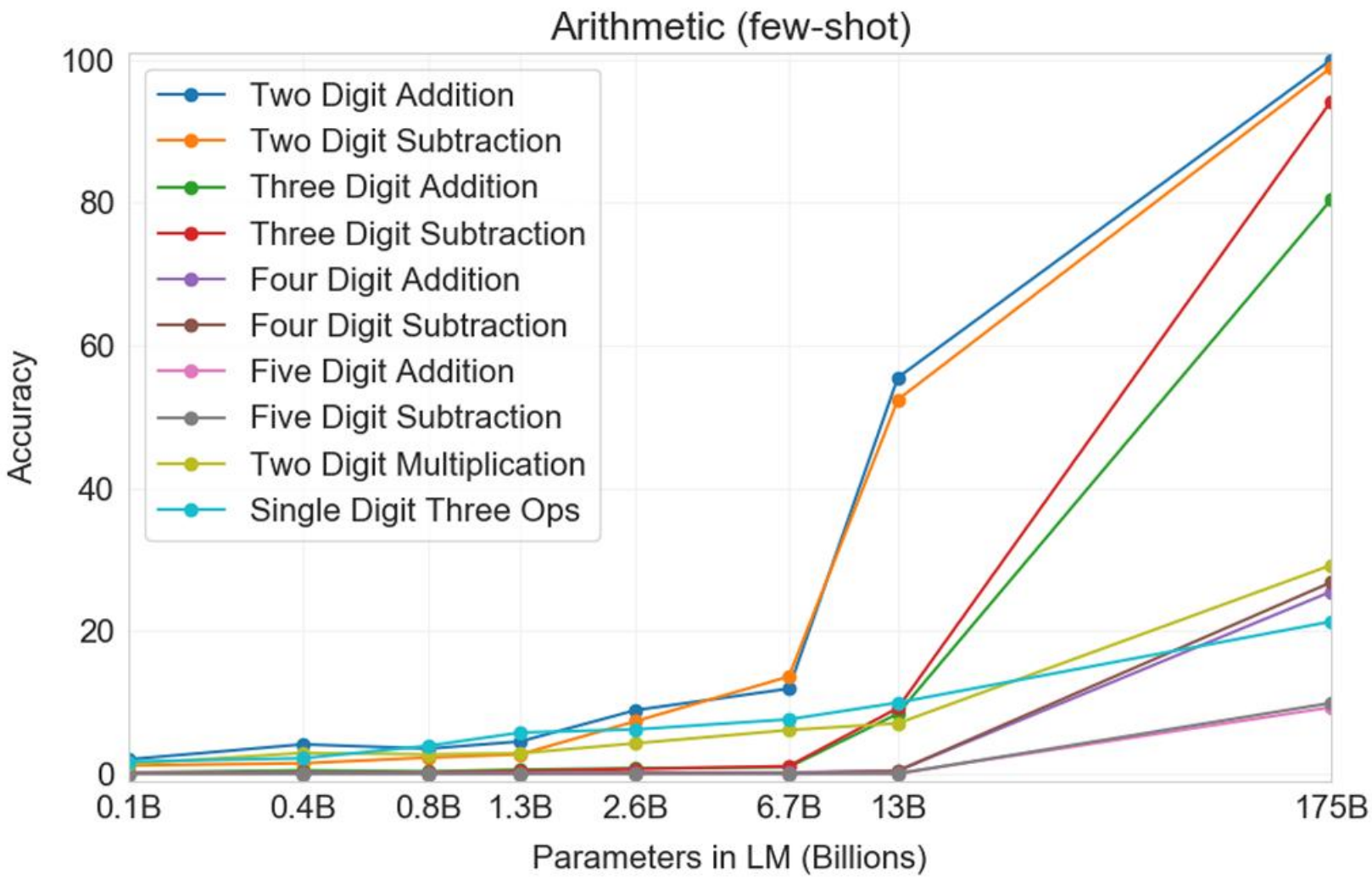
Setting	En→Fr	Fr→En	En→De	De→En	En→Ro	Ro→En
SOTA (Supervised)	**45.6**[a]	35.0 [b]	**41.2**[c]	40.2[d]	**38.5**[e]	**39.9**[e]
XLM [LC19]	33.4	33.3	26.4	34.3	33.3	31.8
MASS [STQ+19]	37.5	34.9	28.3	35.2	35.2	33.1
mBART [LGG+20]	-	-	29.8	34.0	35.0	30.5
GPT-3 Zero-Shot	25.2	21.2	24.6	27.2	14.1	19.9
GPT-3 One-Shot	28.3	33.7	26.2	30.4	20.6	38.6
GPT-3 Few-Shot	32.6	39.2	29.7	40.6	21.0	39.5

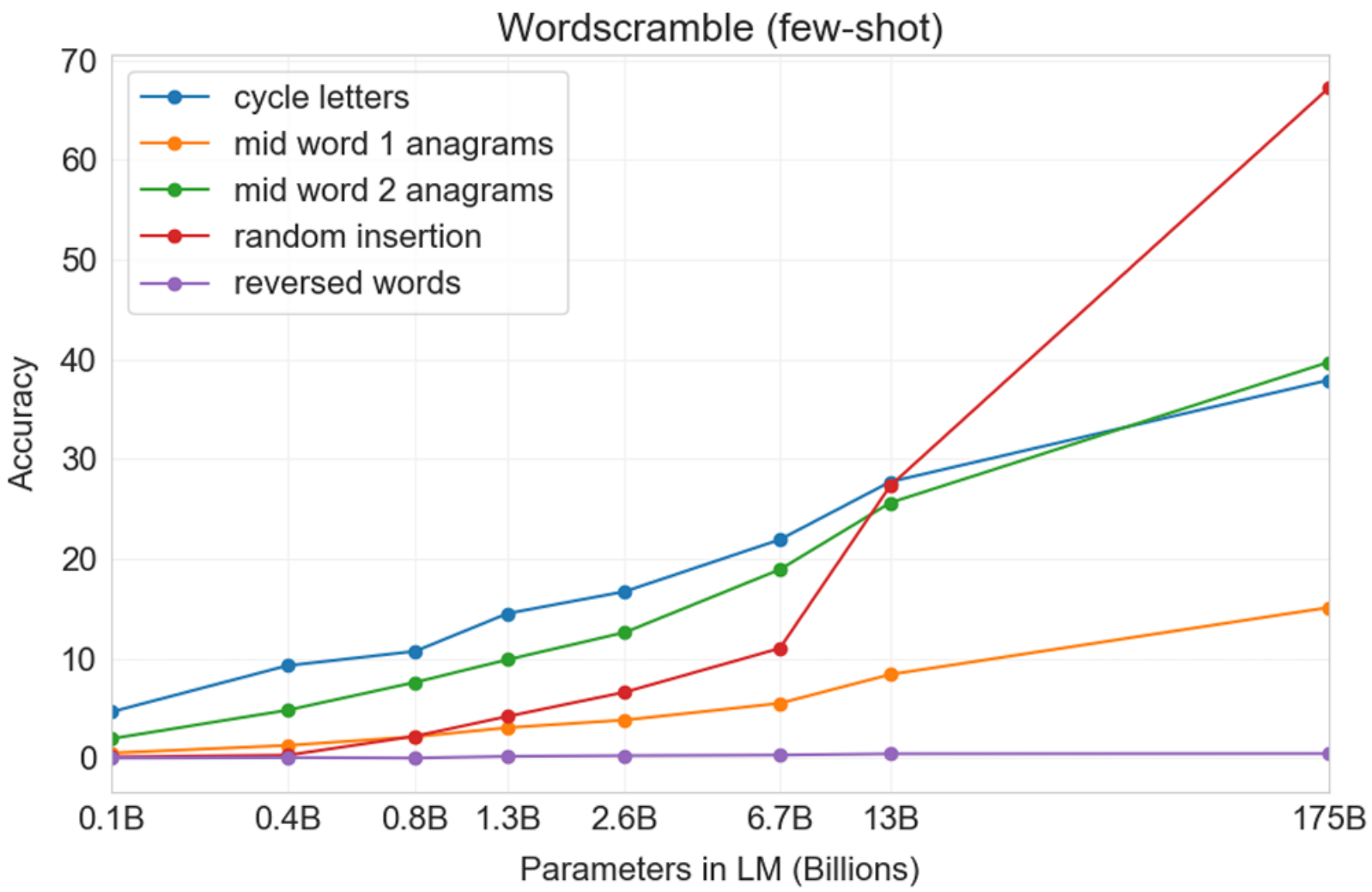
Setting	NaturalQS	WebQS	TriviaQA
RAG (Fine-tuned, Open-Domain) [LPP+20]	**44.5**	**45.5**	**68.0**
T5-11B+SSM (Fine-tuned, Closed-Book) [RRS20]	36.6	44.7	60.5
T5-11B (Fine-tuned, Closed-Book)	34.5	37.4	50.1
GPT-3 Zero-Shot	14.6	14.4	64.3
GPT-3 One-Shot	23.0	25.3	**68.0**
GPT-3 Few-Shot	29.9	41.5	**71.2**

Setting	LAMBADA (acc)	LAMBADA (ppl)	StoryCloze (acc)	HellaSwag (acc)
SOTA	68.0[a]	8.63[b]	**91.8**[c]	**85.6**[d]
GPT-3 Zero-Shot	**76.2**	**3.00**	83.2	78.9
GPT-3 One-Shot	**72.5**	**3.35**	84.7	78.1
GPT-3 Few-Shot	**86.4**	**1.92**	87.7	79.3

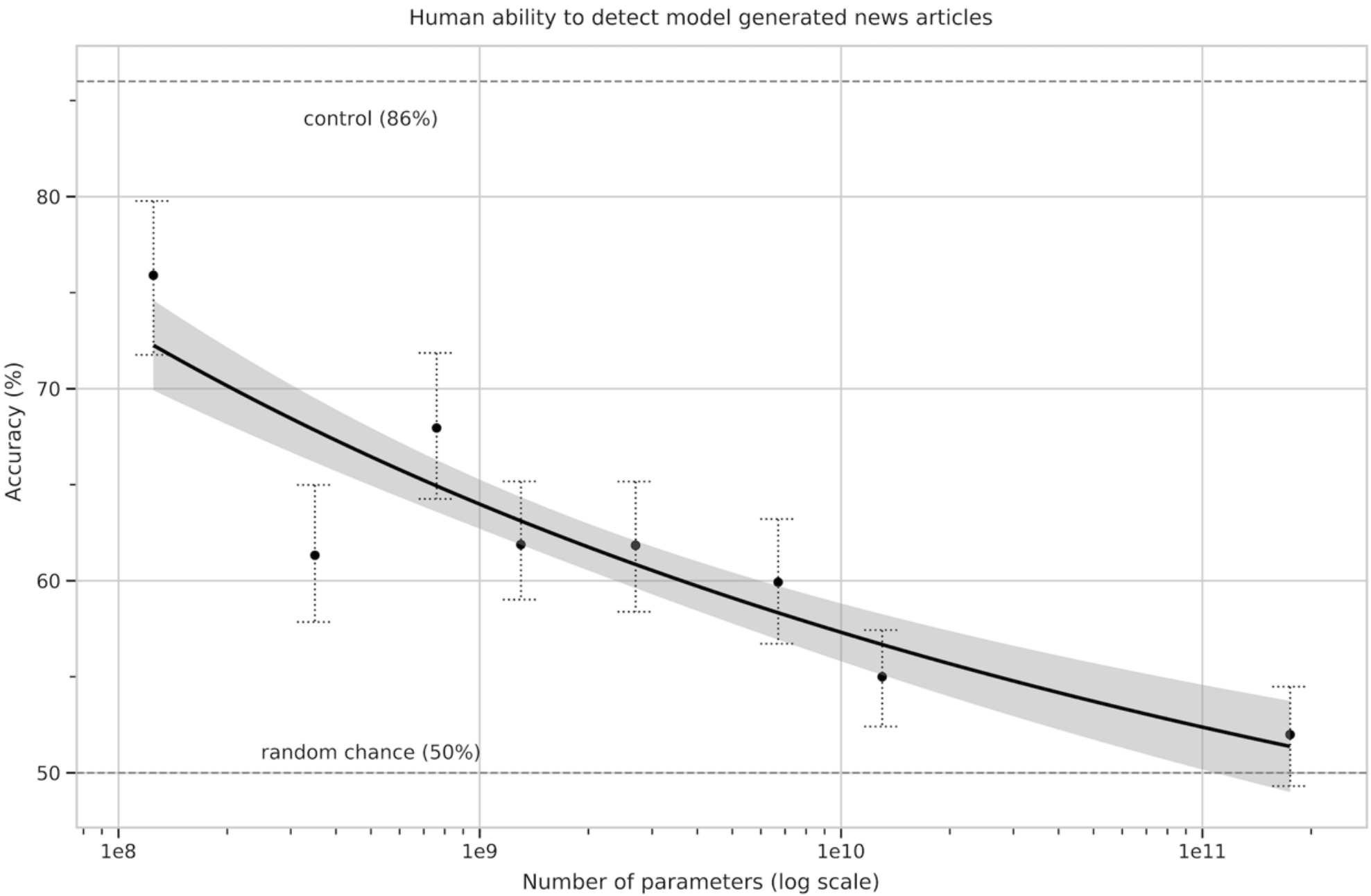
The Winograd Schemas Challenge [LDM12] is a classical task in NLP that involves determining which word a pronoun refers to, when the pronoun is grammatically ambiguous but semantically unambiguous to a human




Arithmetic (few-shot)
Accuracy
100
80
60
40
20
0
Two Digit Addition
Two Digit Subtraction
Three Digit Addition
Three Digit Subtraction
Four Digit Addition
Four Digit Subtraction
Five Digit Addition
Five Digit Subtraction
Two Digit Multiplication
Single Digit Three Ops
0.1B
0.4B
0.8B
1.3B
2.6B
6.7B
13B
175B
Parameters in LM (Billions)


Wordscramble (few-shot)
cycle letters
mid word 1 anagrams
mid word 2 anagrams
random insertion
reversed words
Accuracy
70
60
50
40
30
20
10
0
0.1B
0.4B
0.8B
1.3B
2.6B
6.7B
13B
175B
Parameters in LM (Billions)

- **Cycle letters in word (CL)** – The model is given a word with its letters cycled, then the "=" symbol, and is expected to generate the original word. For example, it might be given "lyinevitab" and should output "inevitably".
- **Anagrams of all but first and last characters (A1)** – The model is given a word where every letter except the first and last have been scrambled randomly, and must output the original word. Example: criroptuon = corruption.
- **Anagrams of all but first and last 2 characters (A2)** – The model is given a word where every letter except the first 2 and last 2 have been scrambled randomly, and must recover the original word. Example: opoepnnt → opponent.
- **Random insertion in word (RI)** – A random punctuation or space character is inserted between each letter of a word, and the model must output the original word. Example: s.u!c/c!e.s s i/o/n = succession.
- **Reversed words (RW)** – The model is given a word spelled backwards, and must output the original word. Example: stcejbo → objects.


Human ability to detect model generated news articles
control (86%)
random chance (50%)
Accuracy (%)
80
70
60
50
1e8
1e9
1e10
1e11
Number of parameters (log scale)

Chat GPT (GPT 3.5 with RLHF)

- **Introduced in 2022.Jan**
- **175B Parameters (like GPT3)**
- **Dataset ?**
- **Architecture ?**
- **+RLHF**

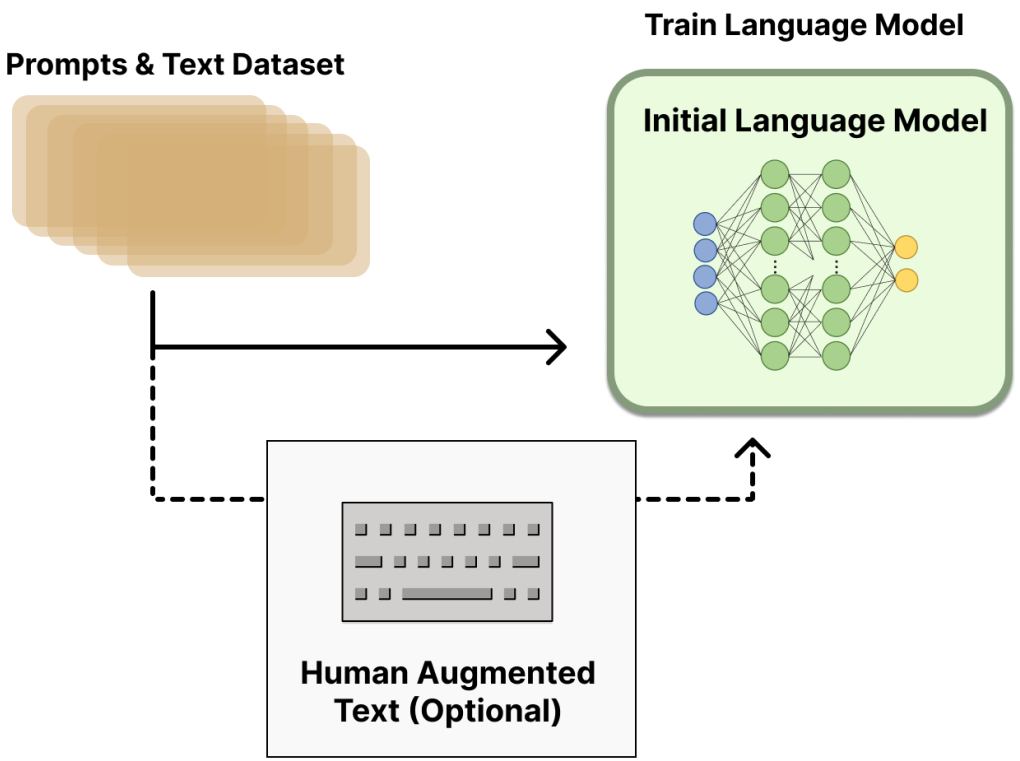
RLHF

- **Steps:**
  1. Pretraining a language model (LM)
  2. gathering data and training a reward model
  3. fine-tuning the LM with reinforcement learning.

1- Pretraining a language model

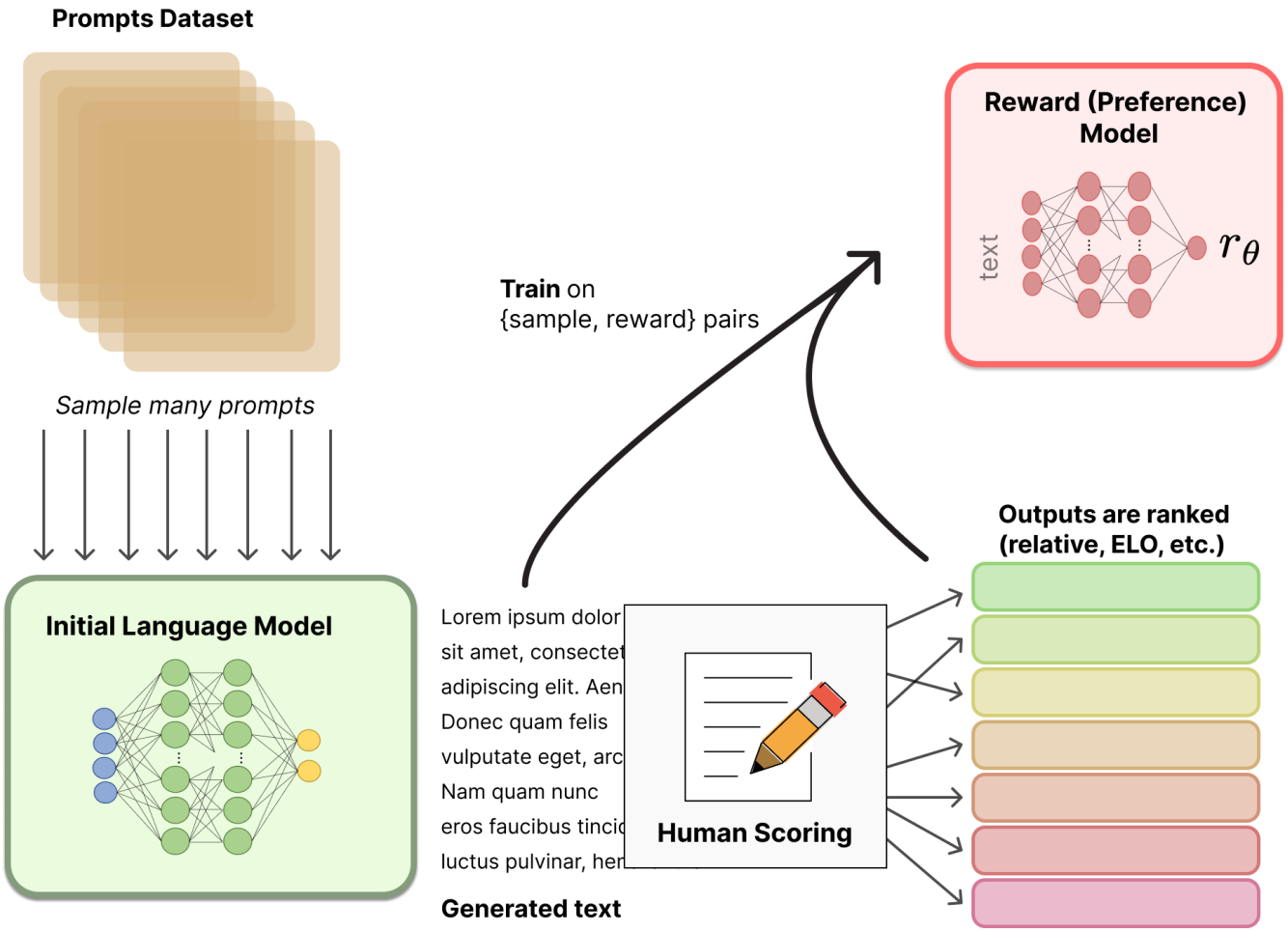
برای مثال، اگر مدل پاسخ‌هایی مانند "من یک ربات هستم" و "من ربات هستم" تولید کند، انسان‌ها ممکن است پاسخ دوم را به دلیل طبیعی‌تر بودن ترجیح دهند.

این داده‌ها برای ایجاد یک مجموعه داده آموزشی استفاده می‌شود که نشان‌دهنده ترجیحات انسانی است.



2- **Reward model training**

یک مدل پاداش با استفاده از داده‌های بازخورد انسانی آموزش داده می‌شود. این مدل یاد می‌گیرد که پاسخ‌های مدل را بر اساس معیارهایی مانند مفید بودن، دقت، و اخلاقی بودن ارزیابی کند (یادگیری با نظارت)

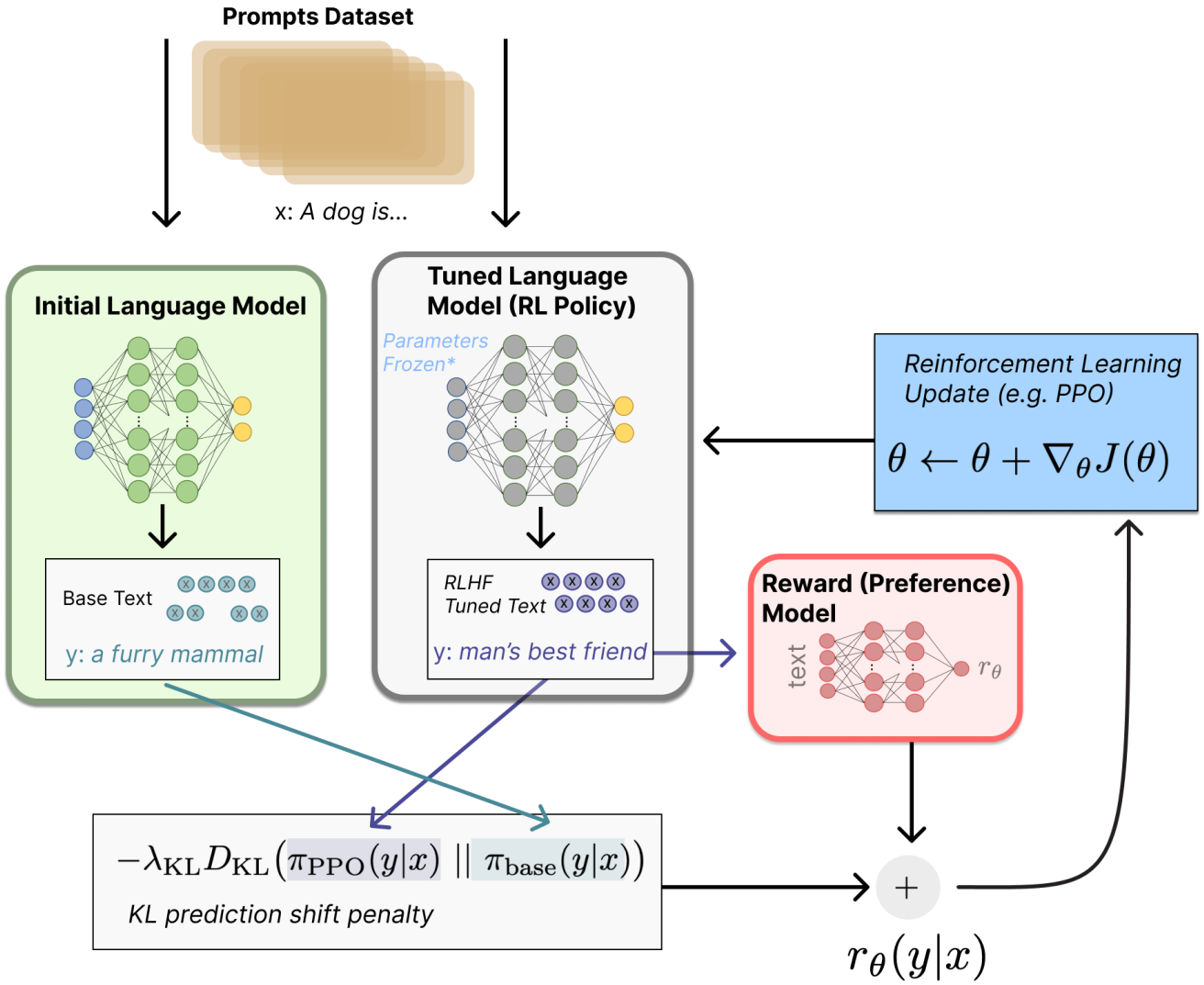


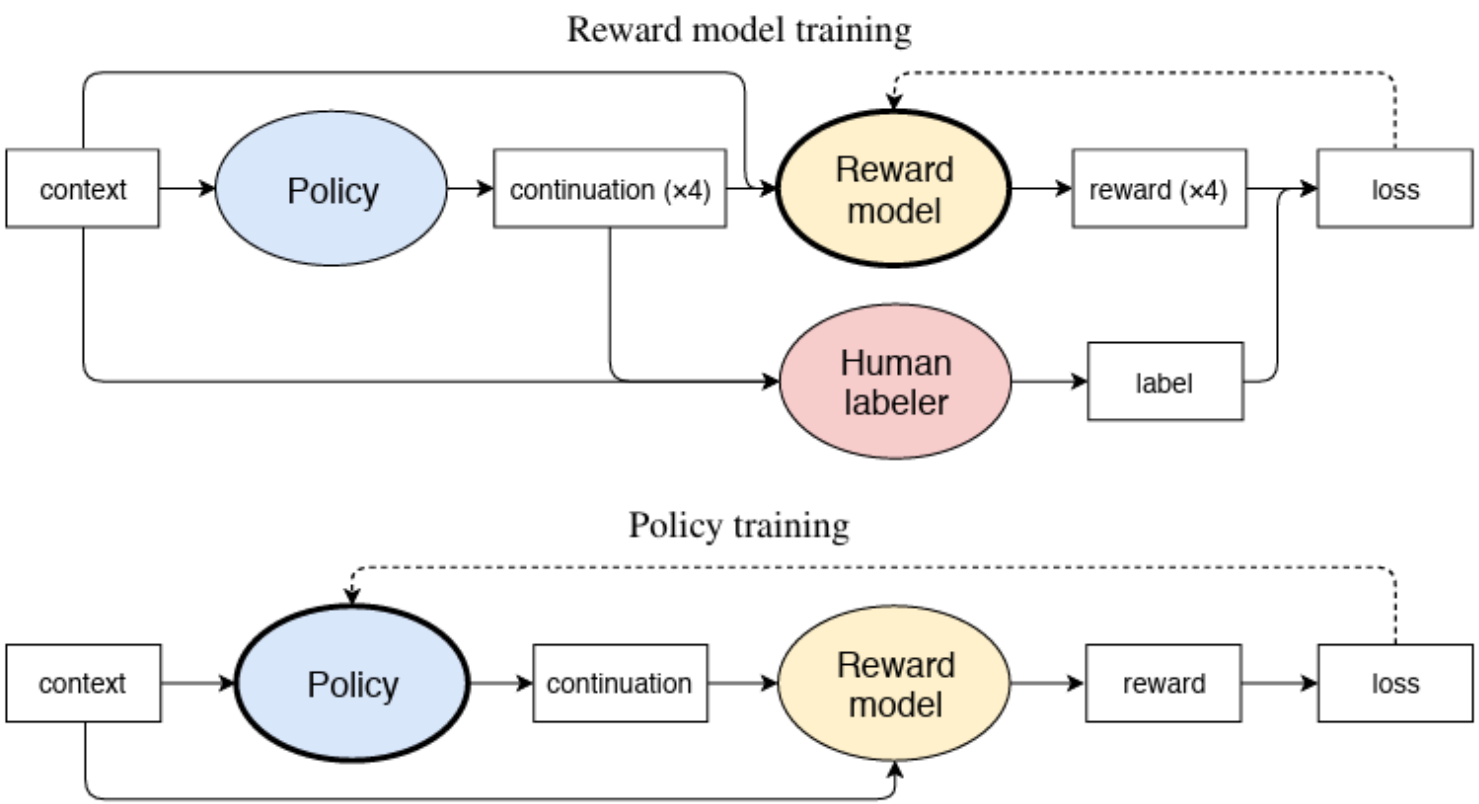
3- Fine-tuning with RL

مدل اصلی ChatGPTبا استفاده از الگوریتم‌های یادگیری تقویتی، مانند Proximal Policy Optimization (PPO)، بهینه می‌شود.

مدل بر اساس پاداش‌هایی که از مدل پاداش دریافت می‌کند، آموزش می‌بیند. هدف این است که سیاست مدل را به گونه‌ای تنظیم کند که پاسخ‌هایی تولید کند که پاداش بیشتری دریافت کنند.

این فرآیند به مدل کمک می‌کند تا پاسخ‌هایی اخلاقی‌تر، ایمن‌تر، و مرتبط‌تر با نیازهای کاربر تولید کند.




Reward model training
context
Policy
continuation (×4)
Reward model
reward (×4)
loss
Human labeler
label
Policy training
context
Policy
continuation
Reward model
reward
loss

$$\rho(x_0 \cdots x_{n-1}) = \prod_{0 \leq k < n} \rho(x_k | x_0 \cdots x_{k-1})$$

We will apply this model to a task with input space $X = \Sigma^{\leq m}$, data distribution $\mathcal{D}$ 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 a probabilistic policy for this task via $\rho(y|x) = \rho(xy)/\rho(x)$: fixing the beginning of the sample to x and generating subsequent tokens using ρ.

We initialize a policy $\pi = \rho$, and then fine-tune π to perform the task well using RL. If the task was defined by a reward function $r : X \times Y \to \mathbb{R}$, then we could use RL to directly optimize the expected reward:

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

Following Christiano et al. (2017), we ask human labelers to pick which of several values of y_i is the best response to a given input x.[1] We ask humans to choose between four options (y_0, y_1, y_2, y_3); considering more options allows a human to amortize the cost of reading and understanding the prompt x. Let $b \in \{0, 1, 2, 3\}$ be the option they select. Having collected a dataset S of $(x, y_0, y_1, y_2, y_3, b)$ tuples, we fit a reward model $r : X \times Y \to \mathbb{R}$ using the loss

$$\text{loss}(r) = \mathbb{E}_{(x,\{y_i\}_i,b)\sim S}\left[\log \frac{e^{r(x,y_b)}}{\sum_i e^{r(x,y_i)}}\right] \tag{1}$$

from ρ rather than π). To keep the scale of the reward model consistent across training, we normalize it so that it has mean 0 and variance 1 for $x \sim \mathcal{D}, y \sim \rho(\cdot|x)$.

Now we fine-tune π to optimize the reward model r. To keep π from moving too far from ρ, we add a penalty with expectation $\beta\,\mathrm{KL}(\pi, \rho)$ (see table 10 for what happens without this). That is, we perform RL on the modified reward

$$R(x, y) = r(x, y) - \beta \log \frac{\pi(y|x)}{\rho(y|x)}. \tag{2}$$

Models trained with different seeds and the same KL penalty β sometimes end up with quite different values of $\mathrm{KL}(\pi, \rho)$, making them hard to compare. To fix this, for some experiments we dynamically vary β to target a particular value of $\mathrm{KL}(\pi, \rho)$ using the log-space proportional controller

$$e_t = \mathrm{clip}\left(\frac{\mathrm{KL}(\pi_t, \rho) - \mathrm{KL}_{\mathrm{target}}}{\mathrm{KL}_{\mathrm{target}}}, -0.2, 0.2\right)$$

$$\beta_{t+1} = \beta_t(1 + K_\beta e_t)$$

We used $K_\beta = 0.1$.

Our overall training process is:

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

2. Initialize r to ρ, using random initialization for the final linear layer of r. Train r on the human samples using loss (1).

3. Train π via Proximal Policy Optimization (PPO, Schulman et al. (2017)) with reward R from (2) on $x \sim \mathcal{D}$.

4. In the online data collection case, continue to collect additional samples, and periodically retrain the reward model r. This is described in section 2.3.

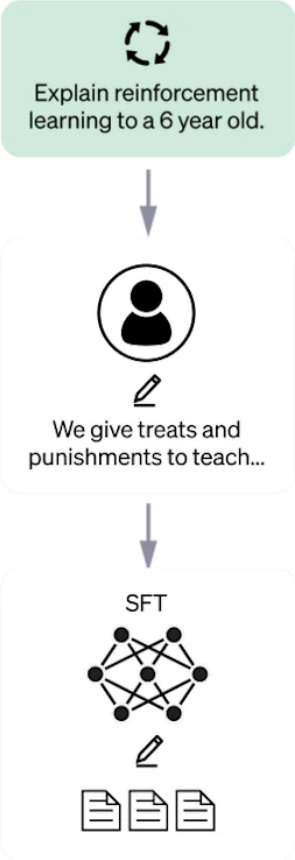
Step 1

Collect demonstration data and train a supervised policy.

A prompt is sample from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3.5 with supervised learning.



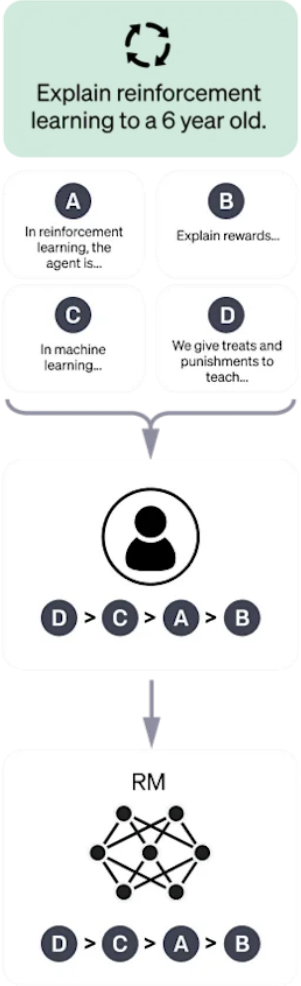
Step 2

Collect comparison data and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.



Step 3

Optimize a policy against the reward model using the PPO reinforcement learning algorithm.

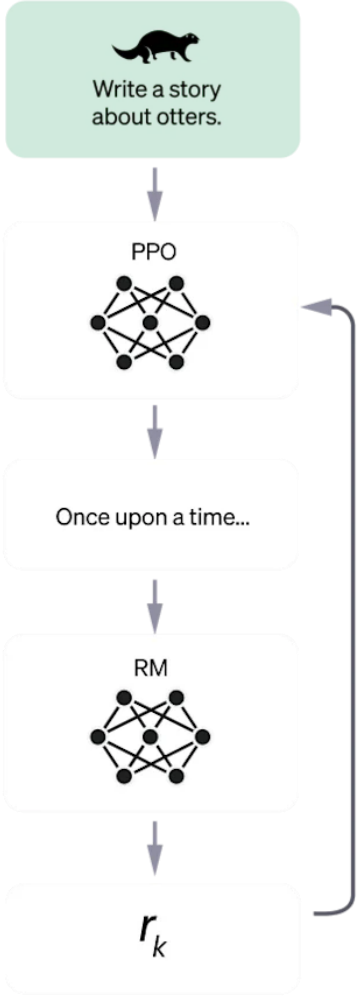
A new prompt is sampled from the dataset.

The PPO model is initialized from the supervised policy.

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.

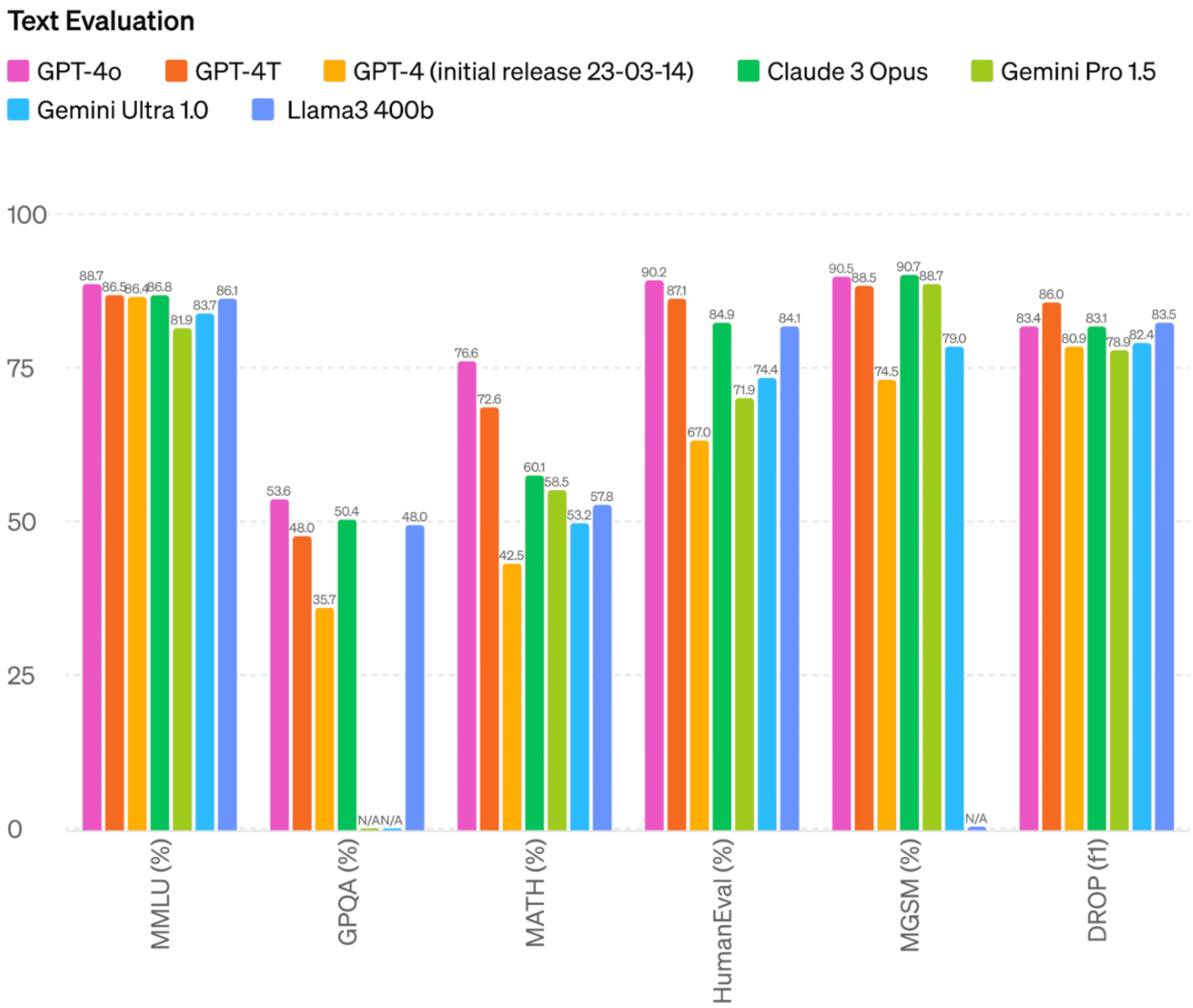


GPT4

- Introduced in 2023.March
- 1.8 Tr Parameters (~10*GPT3)
- Dataset ?
- Architecture ?
- Window(4096-> 8192)

GPT4O

- **Multi-Modal**
- **2024.May**
- **1.8 Tr Parameters (~10*GPT3)**
- **Dataset ?**
- **Architecture ?**
- **Window(8192> 128000)**


Text Evaluation
GPT-4o
GPT-4T
GPT-4 (initial release 23-03-14)
Claude 3 Opus
Gemini Pro 1.5
Gemini Ultra 1.0
Llama3 400b
100
75
50
25
0
88.7
86.5
86.4
86.8
81.9
83.7
86.1
53.6
48.0
35.7
50.4
N/A
N/A
48.0
76.6
72.6
42.5
60.1
58.5
53.2
57.8
90.2
87.1
67.0
84.9
71.9
74.4
84.1
90.5
88.5
74.5
90.7
88.7
79.0
N/A
83.4
86.0
80.9
83.1
78.9
82.4
83.5
MMLU (%)
GPQA (%)
MATH (%)
HumanEval (%)
MGSM (%)
DROP (f1)

You

Which movie is this scene from?

ChatGPT

This scene is from the movie "Home Alone."

You

How many coins do I have?

ChatGPT

You have five coins in the image.

You

How many coins do I have?

ChatGPT

You have five coins in the image. Four of them are silver-colored, and one is gold-colored.

The many things GPT-4o can do include the following:

•**Real-time interactions.** The GPT-4o model can engage in real-time verbal conversations without any real noticeable delays.

•**Knowledge-based Q&A.** As was the case with all prior GPT-4 models, GPT-4o has been trained with a knowledge base and can respond to questions.

•**Text summarization and generation.** As was the case with all prior GPT-4 models, GPT-4o can execute common text LLM tasks, including text summarization and generation.

•**Multimodal reasoning and generation.** GPT-4o integrates text, voice and vision into a single model, allowing it to process and respond to a combination of data types. The model can understand audio, images and text at the same speed. It can also generate responses via audio, images and text.

•**Language and audio processing.** GPT-4o has advanced capabilities in handling more than 50 different languages.

•.

- **Sentiment analysis.** The model understands user sentiment across different modalities of text, audio and video.

- **Voice nuance.** GPT-4o can generate speech with emotional nuances. This makes it effective for applications requiring sensitive and nuanced communication.

- **Audio content analysis.** The model can generate and understand spoken language, which can be applied in voice-activated systems, audio content analysis and interactive storytelling.

- **Real-time translation.** The multimodal capabilities of GPT-4o support real-time translation from one language to another.

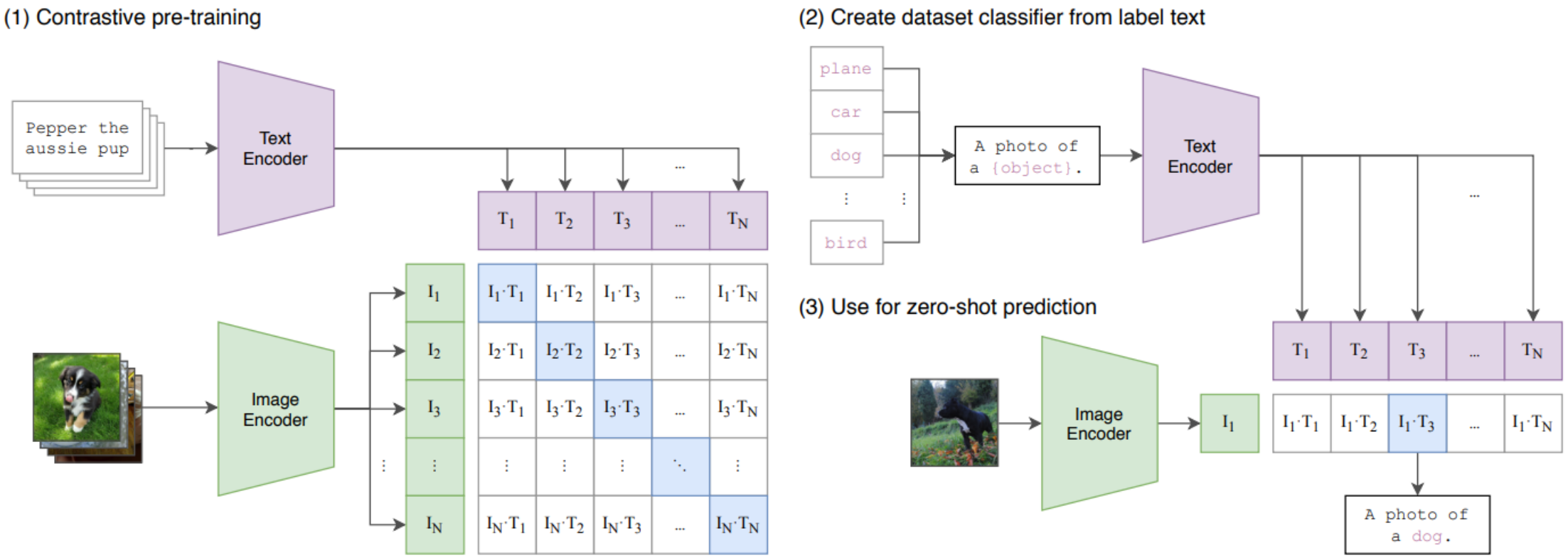
- **Image understanding and vision.** The model can analyze images and videos, allowing users to upload visual content that GPT-4o will understand, explain and provide analysis for.

- **Data analysis.** The vision and reasoning capabilities let users analyze data contained in data charts. GPT-4o can also create data charts based on analysis or a prompt.

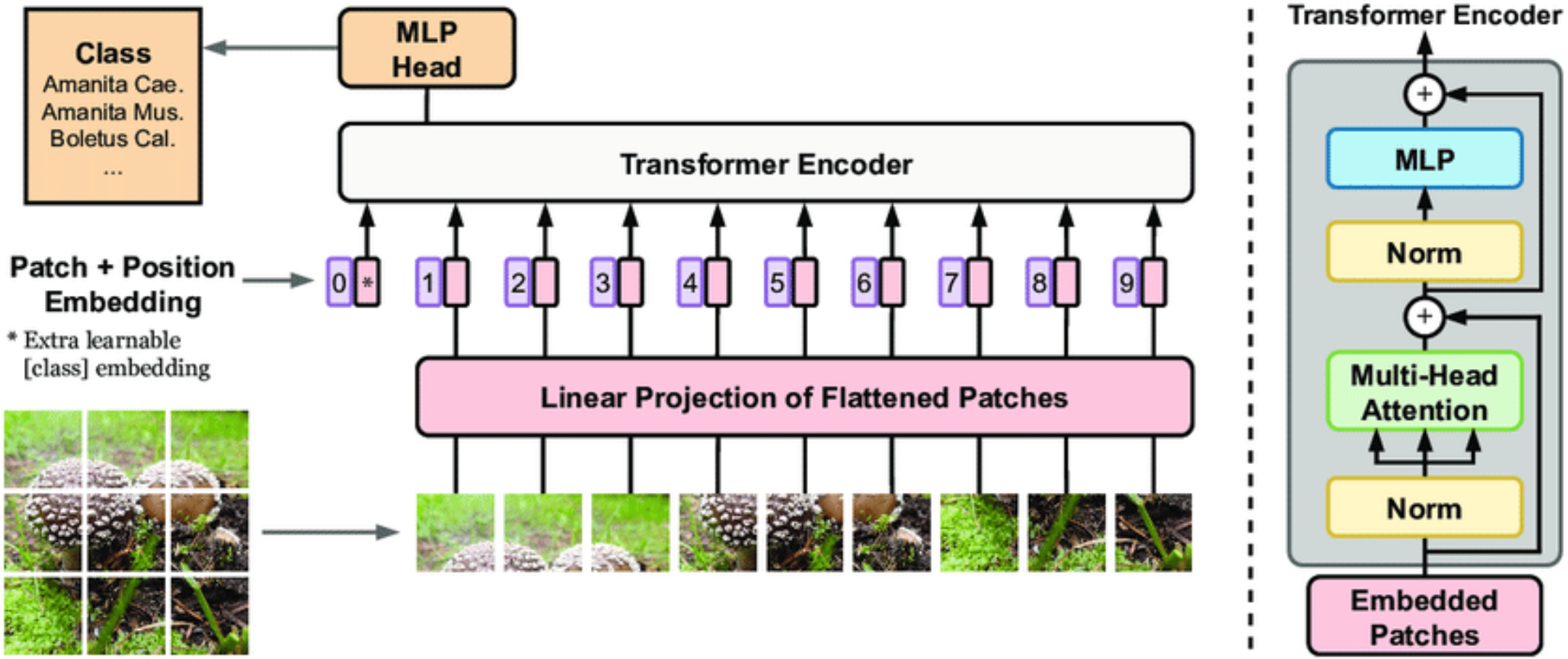
- **Software development.** GPT-4o can generate new code for an application, as well as analyze and debug existing code.

- **File uploads.** Beyond the knowledge cutoff, GPT-4o supports file uploads, letting users analyze specific data for analysis.

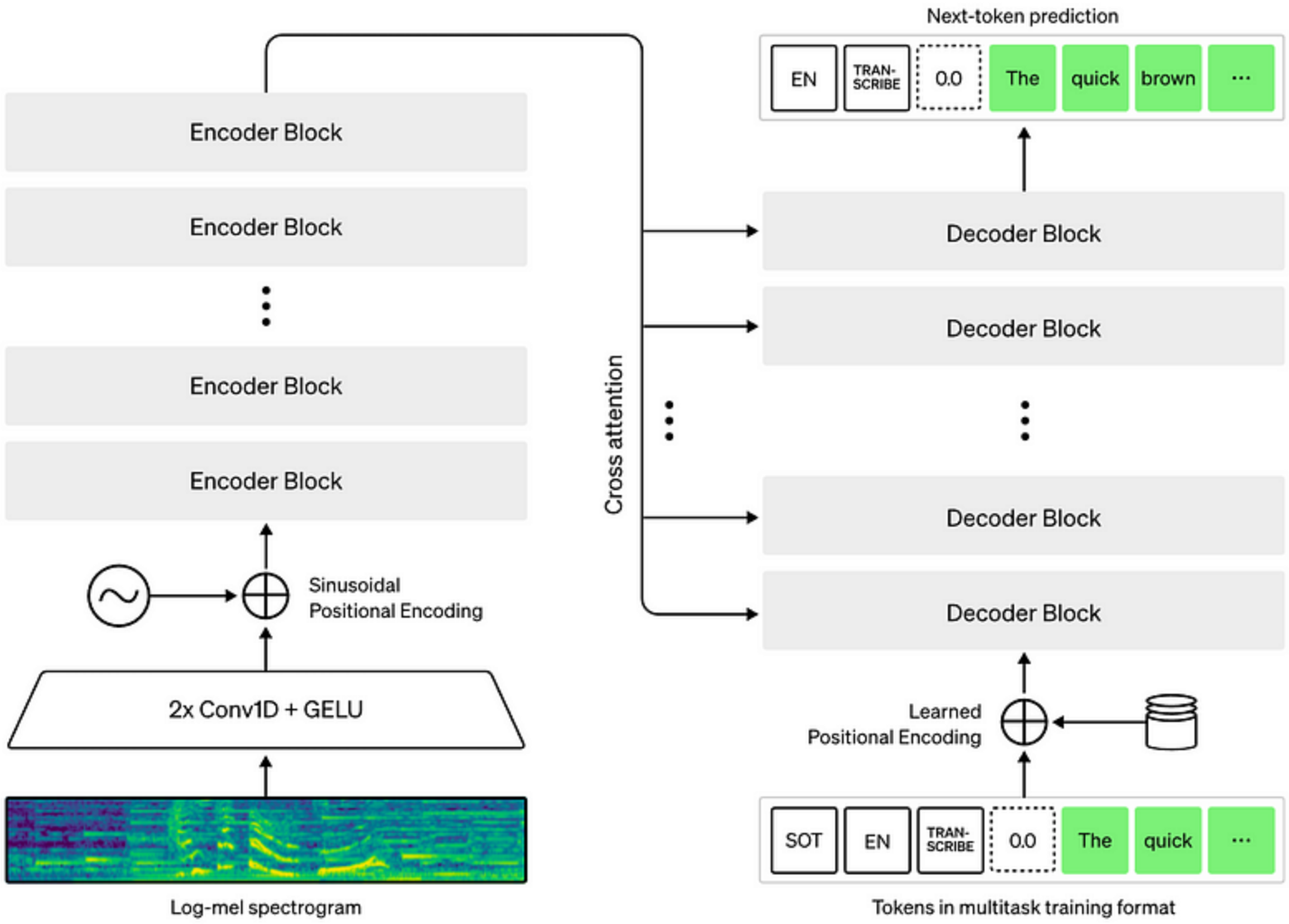
CLIP


(1) Contrastive pre-training
Pepper the aussie pup
Text Encoder
Image Encoder
T_1 T_2 T_3 ... T_N
I_1 I_2 I_3 ⋮ I_N
$I_1 \cdot T_1$ $I_1 \cdot T_2$ $I_1 \cdot T_3$ ... $I_1 \cdot T_N$
$I_2 \cdot T_1$ $I_2 \cdot T_2$ $I_2 \cdot T_3$ ... $I_2 \cdot T_N$
$I_3 \cdot T_1$ $I_3 \cdot T_2$ $I_3 \cdot T_3$ ... $I_3 \cdot T_N$
$I_N \cdot T_1$ $I_N \cdot T_2$ $I_N \cdot T_3$ ... $I_N \cdot T_N$
(2) Create dataset classifier from label text
plane
car
dog
bird
A photo of a {object}.
Text Encoder
(3) Use for zero-shot prediction
Image Encoder
I_1
$I_1 \cdot T_1$ $I_1 \cdot T_2$ $I_1 \cdot T_3$ ... $I_1 \cdot T_N$
A photo of a dog.

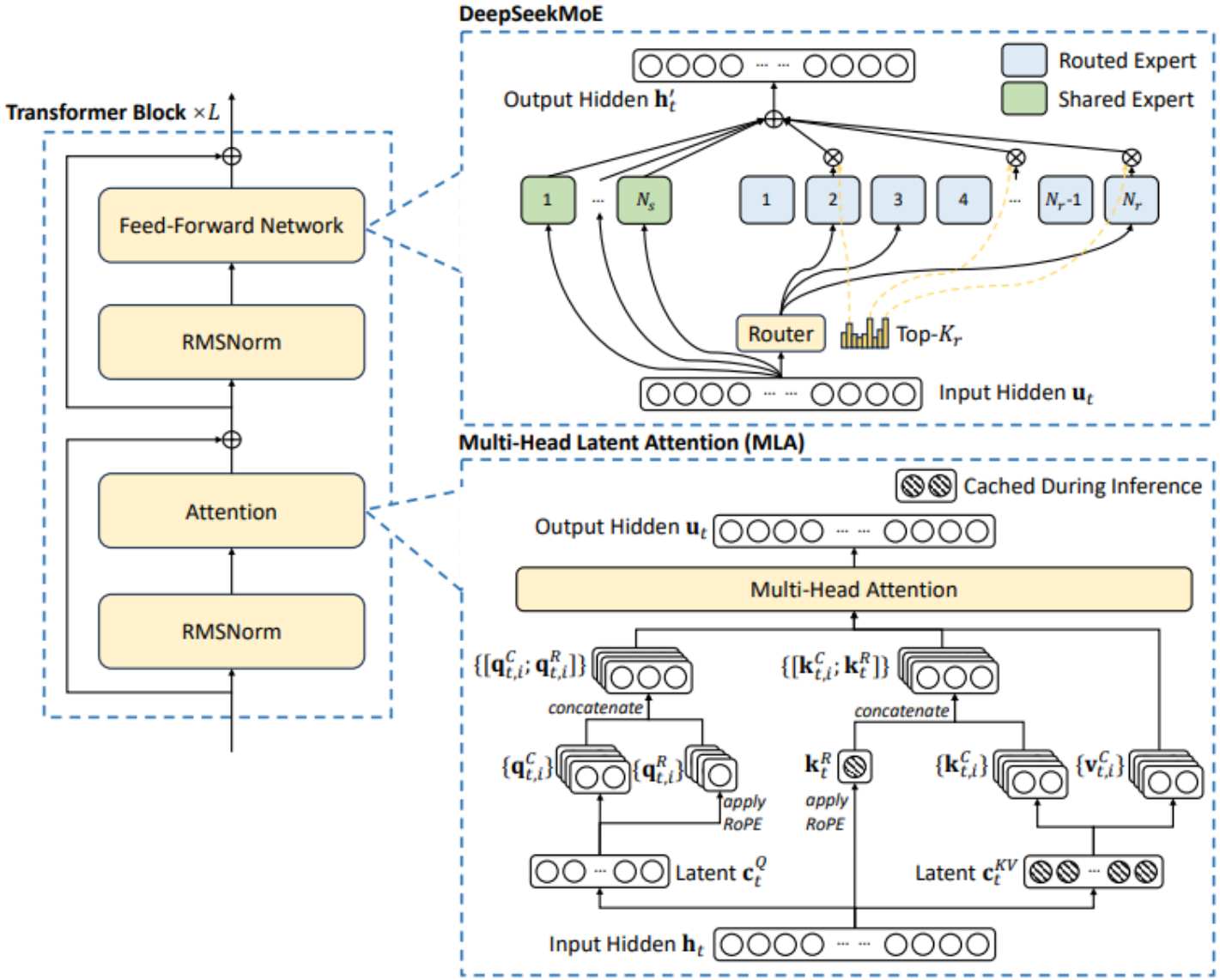
ViT


Class
Amanita Cae.
Amanita Mus.
Boletus Cal.
...
MLP Head
Transformer Encoder
Patch + Position Embedding
* Extra learnable [class] embedding
0 * 1 2 3 4 5 6 7 8 9
Linear Projection of Flattened Patches
Transformer Encoder
MLP
Norm
Multi-Head Attention
Norm
Embedded Patches

Whisper


Next-token prediction
EN
TRAN-SCRIBE
0.0
The
quick
brown
...
Encoder Block
Encoder Block
Encoder Block
Encoder Block
Cross attention
Decoder Block
Decoder Block
Decoder Block
Decoder Block
Sinusoidal
Positional Encoding
2x Conv1D + GELU
Learned
Positional Encoding
SOT
EN
TRAN-SCRIBE
0.0
The
quick
...
Log-mel spectrogram
Tokens in multitask training format

DeepSeek


Transformer Block ×L
Feed-Forward Network
RMSNorm
Attention
RMSNorm
DeepSeekMoE
Routed Expert
Shared Expert
Output Hidden $\mathbf{h}'_t$
1
N_s
1
2
3
4
N_r-1
N_r
Router
Top-K_r
Input Hidden $\mathbf{u}_t$
Multi-Head Latent Attention (MLA)
Cached During Inference
Output Hidden $\mathbf{u}_t$
Multi-Head Attention
$\{[\mathbf{q}^C_{t,i}; \mathbf{q}^R_{t,i}]\}$
$\{[\mathbf{k}^C_{t,i}; \mathbf{k}^R_t]\}$
concatenate
concatenate
$\{\mathbf{q}^C_{t,i}\}$
$\{\mathbf{q}^R_{t,i}\}$
$\mathbf{k}^R_t$
$\{\mathbf{k}^C_{t,i}\}$
$\{\mathbf{v}^C_{t,i}\}$
apply RoPE
apply RoPE
Latent $\mathbf{c}^Q_t$
Latent $\mathbf{c}^{KV}_t$
Input Hidden $\mathbf{h}_t$

DeepSeek

Multi-head Latent Attention (MLA):

MLAبرای کاهش ردپای حافظه KV Cache در استنتاج، به‌ویژه برای زمینه‌های طولانی، استفاده می‌شود. این روش با فشرده‌سازی بردارهای Key/Value با پروجکشن‌های کم‌رتبه، کارایی را بهبود می‌بخشد.

DeepSeekMoE Architecture:

این معماری برای آموزش و استنتاج مقرون‌به‌صرفه طراحی شده و شامل بهینه‌سازی‌هایی مانند segmentation دقیق Expertها و مسیریابی پویا برای تعادل بار است.

برای هر توکن ورودی، ۸ متخصص از ۲۵۶ متخصص انتخاب می‌شوند، به علاوه متخصص اشتراکی، که در مجموع ۹ متخصص فعال می‌شوند.

پیش‌بینی چندتوکنیMulti-Token Prediction – MTP

یک هدف آموزشی است که مدل را وادار می‌کند دو توکن متوالی را پیش‌بینی کند، که درک وابستگی‌های طولانی‌تر را بهبود می‌بخشد و عملکرد را در وظایف متوالی تقویت می‌کند.

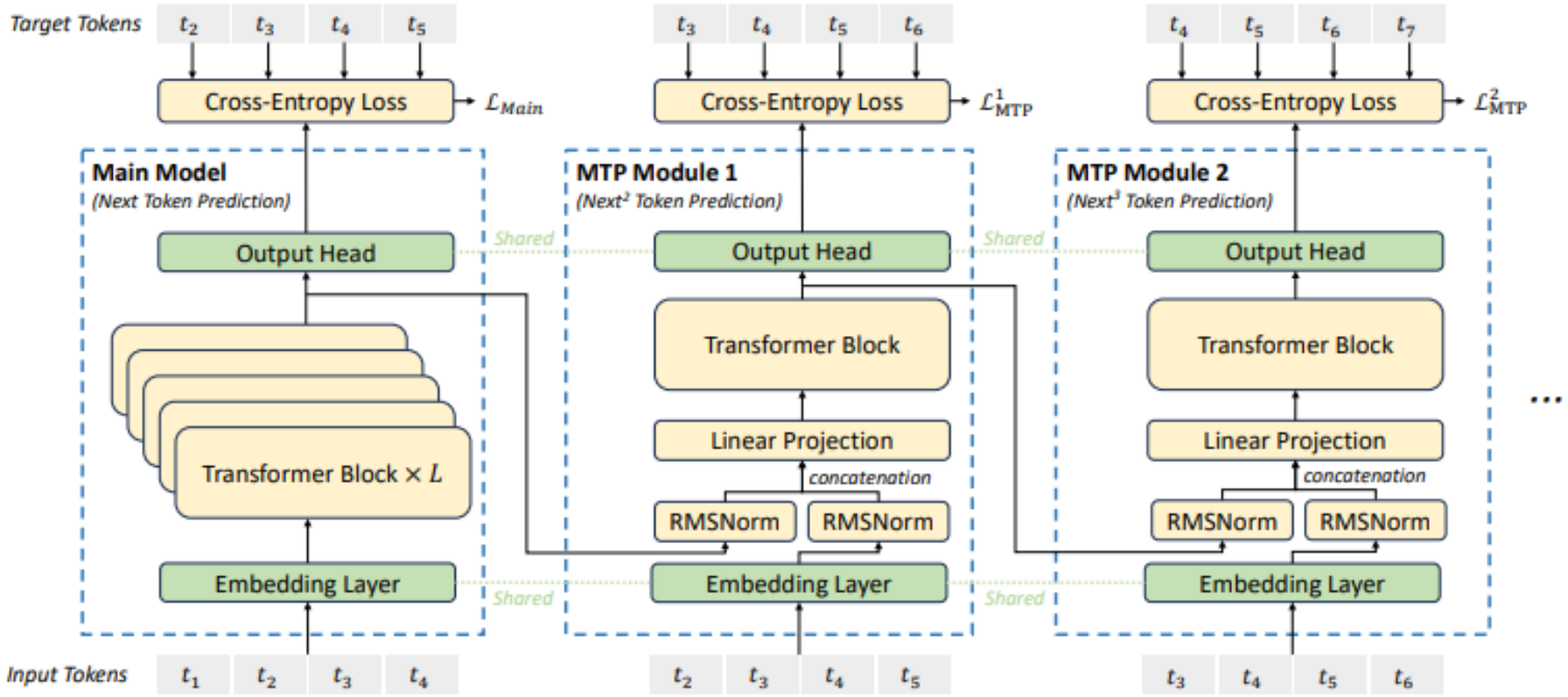


Figure 3 | Illustration of our Multi-Token Prediction (MTP) implementation. We keep the complete causal chain for the prediction of each token at each depth.

آموزش با دقت FP8:

DeepSeek-V3 از آموزش با دقت مخلوط FP8 (E4M3) استفاده می‌کند، که مصرف حافظه و هزینه‌های محاسباتی را کاهش می‌دهد، در حالی که عملکرد را حفظ می‌کند.

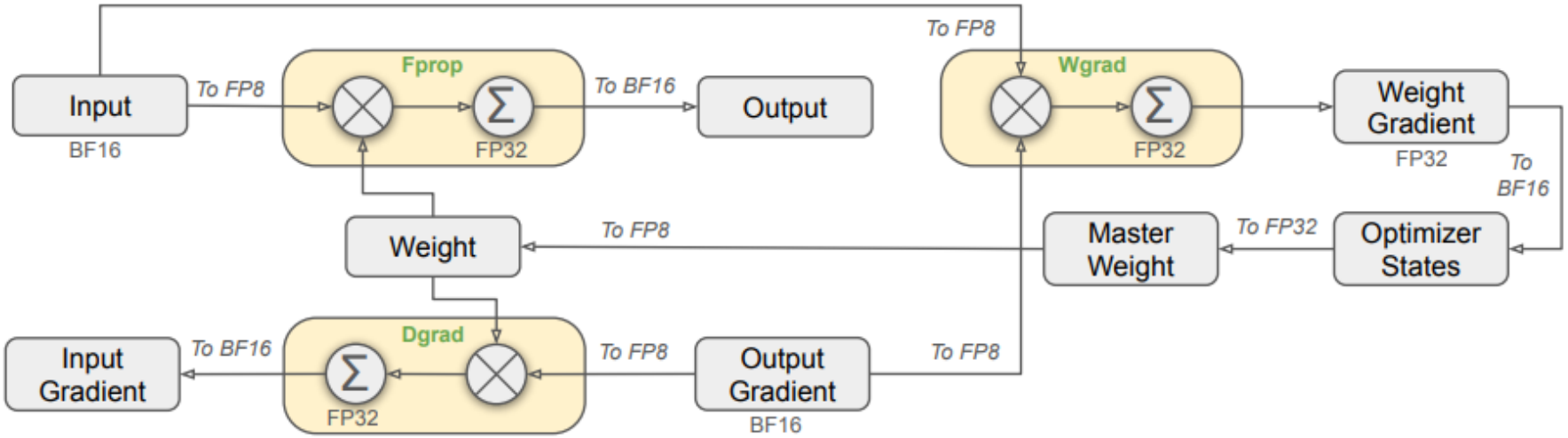


Figure 6 | The overall mixed precision framework with FP8 data format. For clarification, only the `Linear` operator is illustrated.