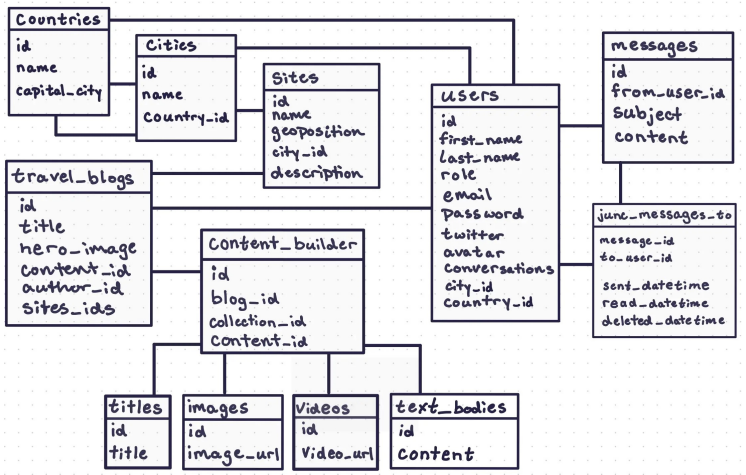
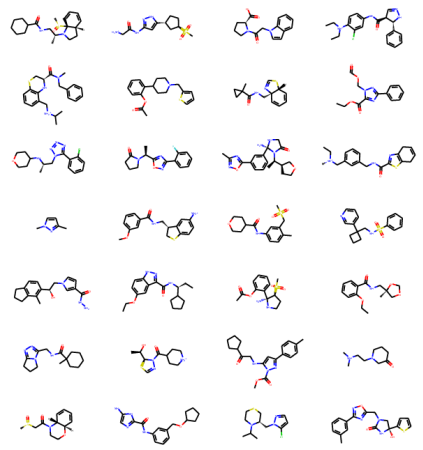
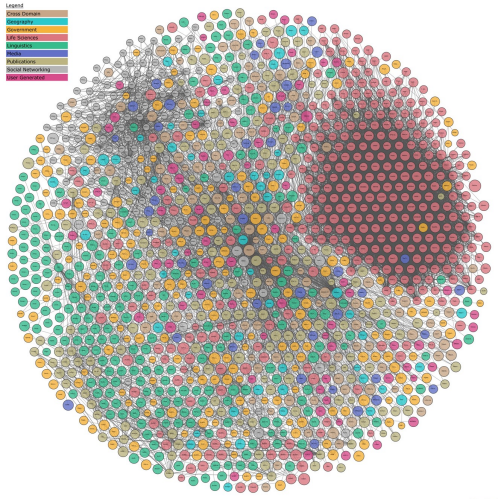
Language Models & Structured Data

Mehwish Alam
Associate Professor
Télécom Paris
Institut Polytechnique de Paris
Winter Semester 2024-2025

Structured Data



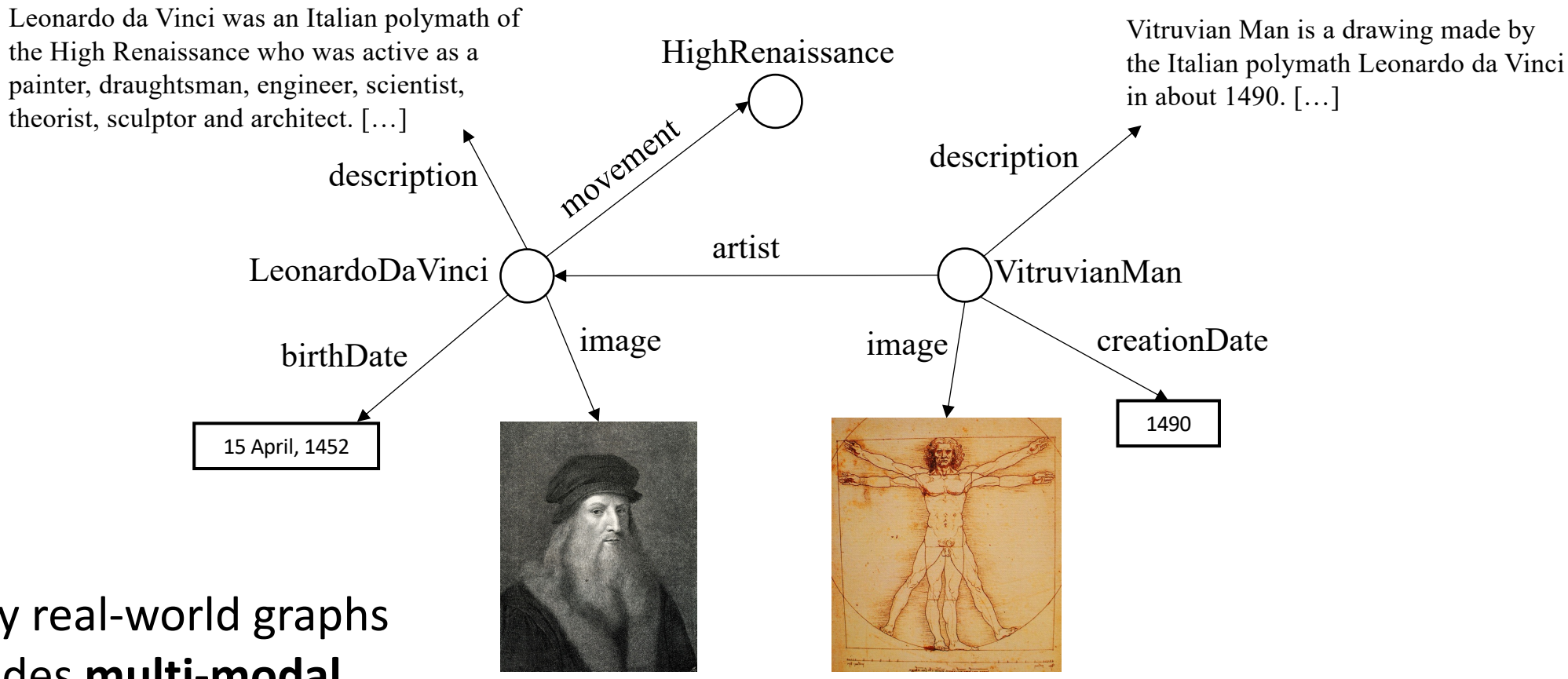




Team	Summer Olympic Games					Winter Olympic Games					Combined total				
Team (IOC code)	No.					No.					No.				
Afghanistan (AFG)	16	0	0	2	2	0	0	0	0	0	16	0	0	2	2
Albania (ALB)	10	0	0	2	2	5	0	0	0	0	15	0	0	2	2
Algeria (ALG)	15	7	4	9	20	3	0	0	0	0	18	7	4	9	20
Argentina (ARG)	26	22	27	31	80	20	0	0	0	0	46	22	27	31	80
Armenia (ARM)	8	2	11	9	22	8	0	0	0	0	16	2	11	9	22
Australasia (ANZ) [ANZ]	2	3	4	5	12	0	0	0	0	0	2	3	4	5	12
Australia (AUS) [AUS][Z]	28	182	192	226	600	20	6	7	6	19	48	188	199	232	619
Austria (AUT)	29	22	35	44	101	24	71	88	91	250	53	93	123	135	351
Azerbaijan (AZE)	8	9	16	31	56	7	0	0	0	0	15	9	16	31	56
Bahamas (BAH)	18	8	2	6	16	0	0	0	0	0	18	8	2	6	16
Bahrain (BRN)	11	4	3	1	8	0	0	0	0	0	11	4	3	1	8
Barbados (BAR) [BAR]	14	0	0	1	1	0	0	0	0	0	14	0	0	1	1
Belarus (BLR)	7	13	30	42	85	8	8	7	5	20	15	21	37	47	105
Belgium (BEL)	28	47	57	63	167	22	2	2	4	8	50	49	59	67	175
Bermuda (BER)	20	1	0	1	2	8	0	0	0	0	28	1	0	1	2
Bohemia (BOH) [BOH][Z]	3	0	1	3	4	0	0	0	0	0	3	0	1	3	4
Botswana (BOT)	12	1	2	1	4	0	0	0	0	0	12	1	2	1	4
Brazil (BRA)	24	40	49	81	170	9	0	0	0	0	33	40	49	81	170
British West	1	0	0	2	2	0	0	0	0	0	1	0	0	2	2

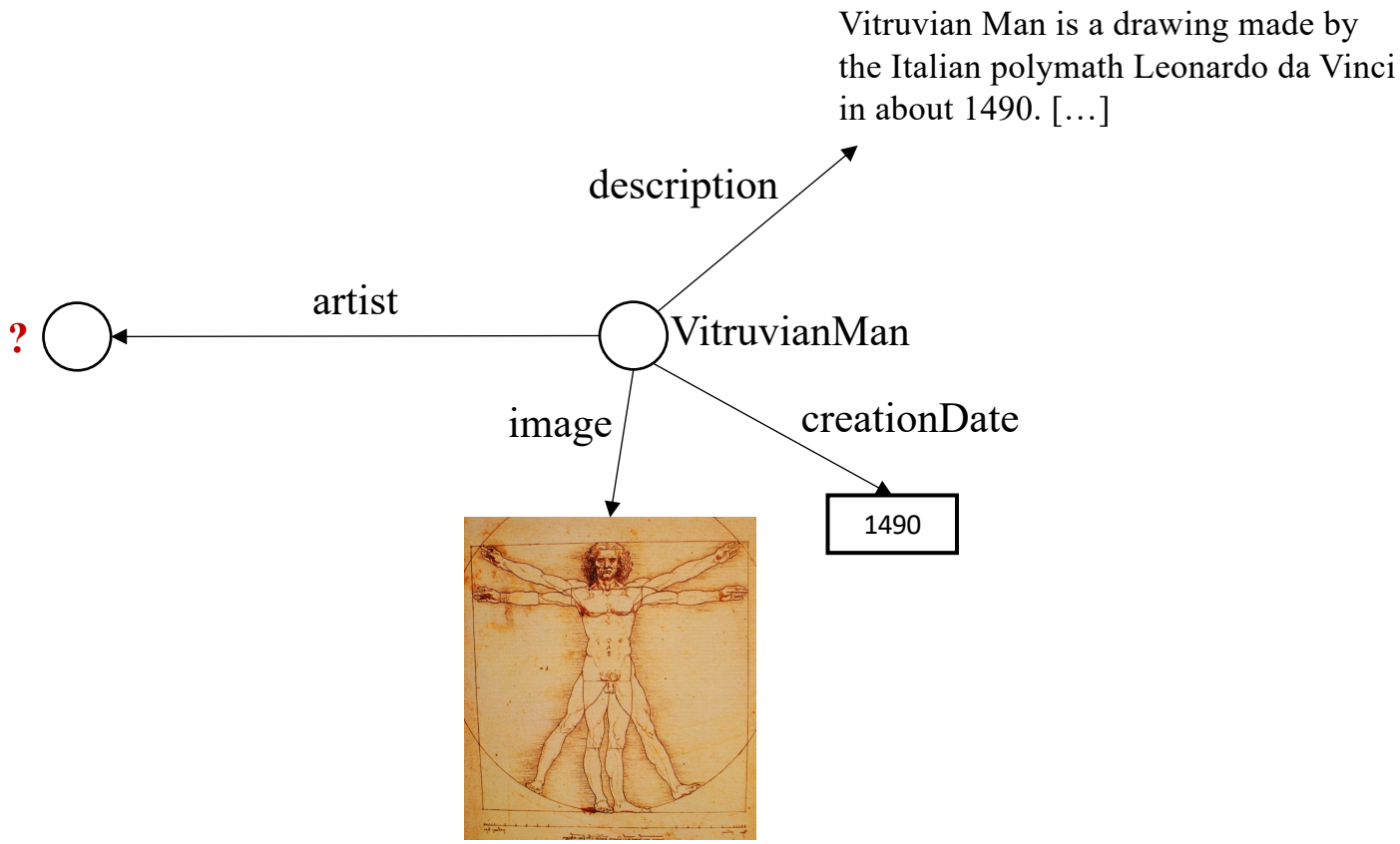
[1] https://docs.directus.io/app/data-model.html
[2] https://lod-cloud.net/
[3] https://en.wikipedia.org/wiki/All-time_Olympic_Games_medal_table

Knowledge Graphs

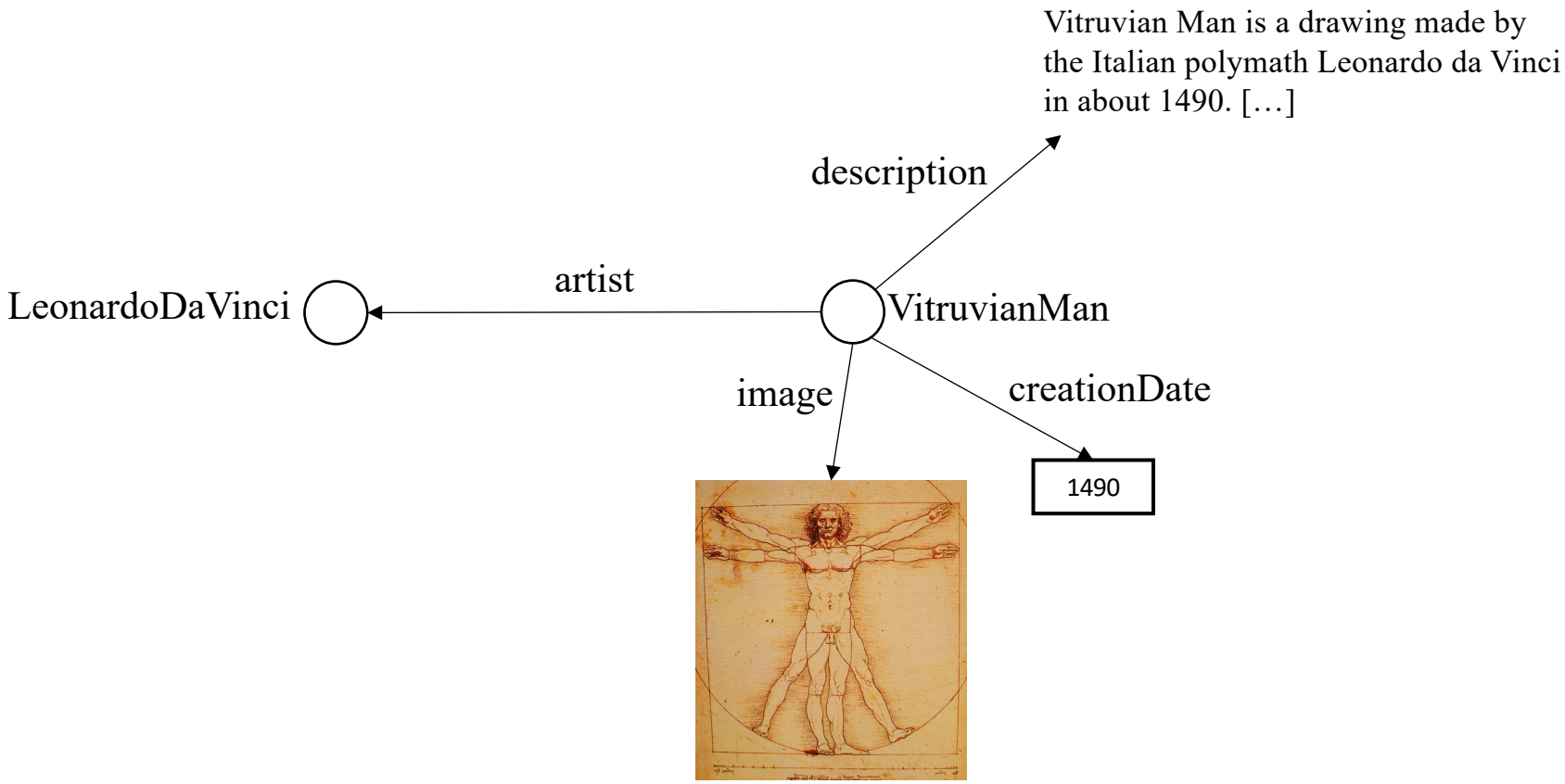


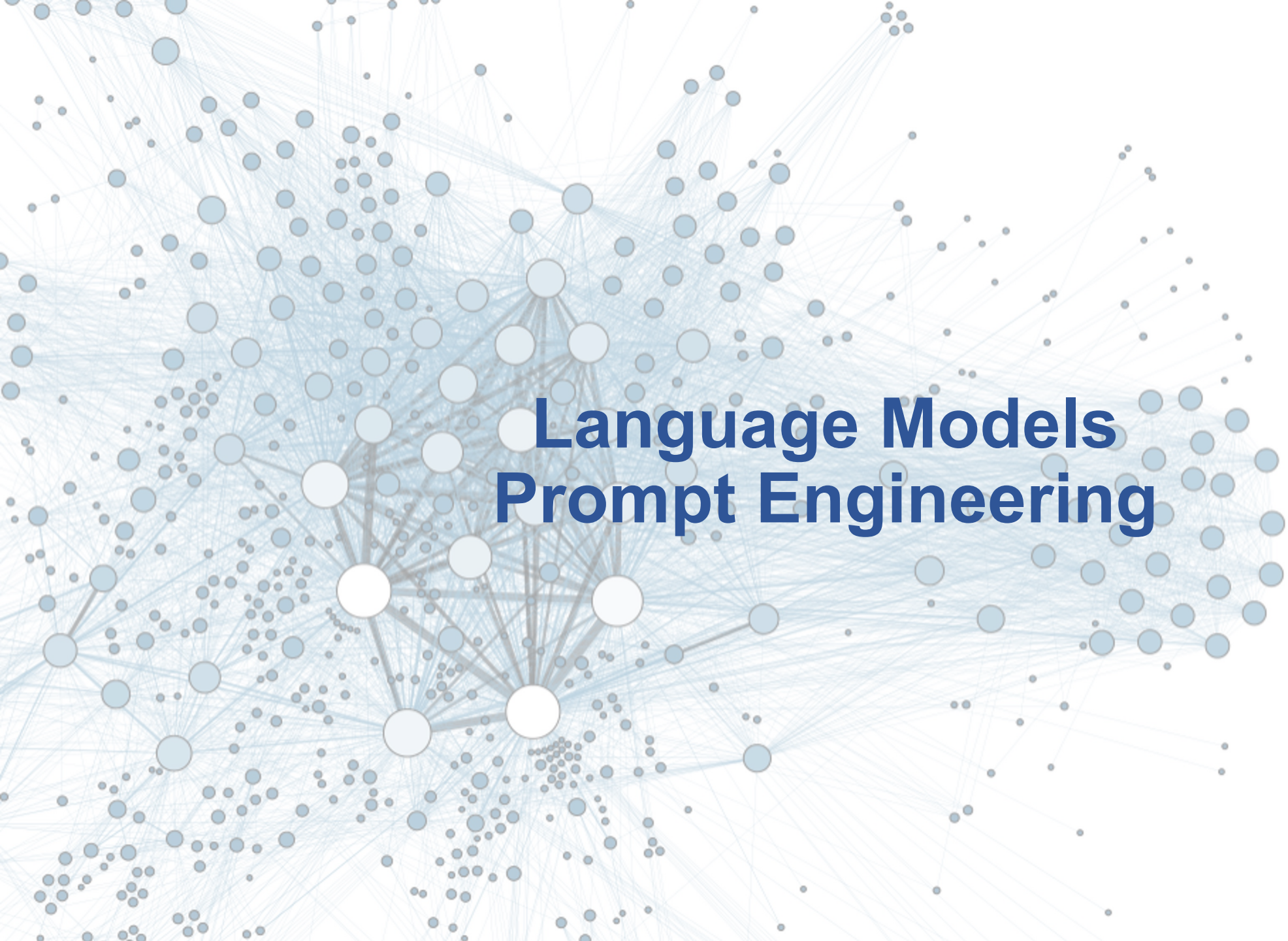
Many real-world graphs includes **multi-modal attributes.**

Multimodal Knowledge Graphs



Multimodal Knowledge Graphs





Language Models
Prompt Engineering

Outline

- Statistical and Neural Language Models
- Pre-trained Language Models
- Large Language Models
- Parameter Efficient Tuning
- Prompt Engineering

Outline

- **Statistical and Neural Language Models**
- **Pre-trained Language Models**
- **Large Language Models**
- **Parameter Efficient Tuning**
- **Prompt Engineering**

Probabilistic Language Models

- Goal: compute the probability of a sentence or sequence of words:

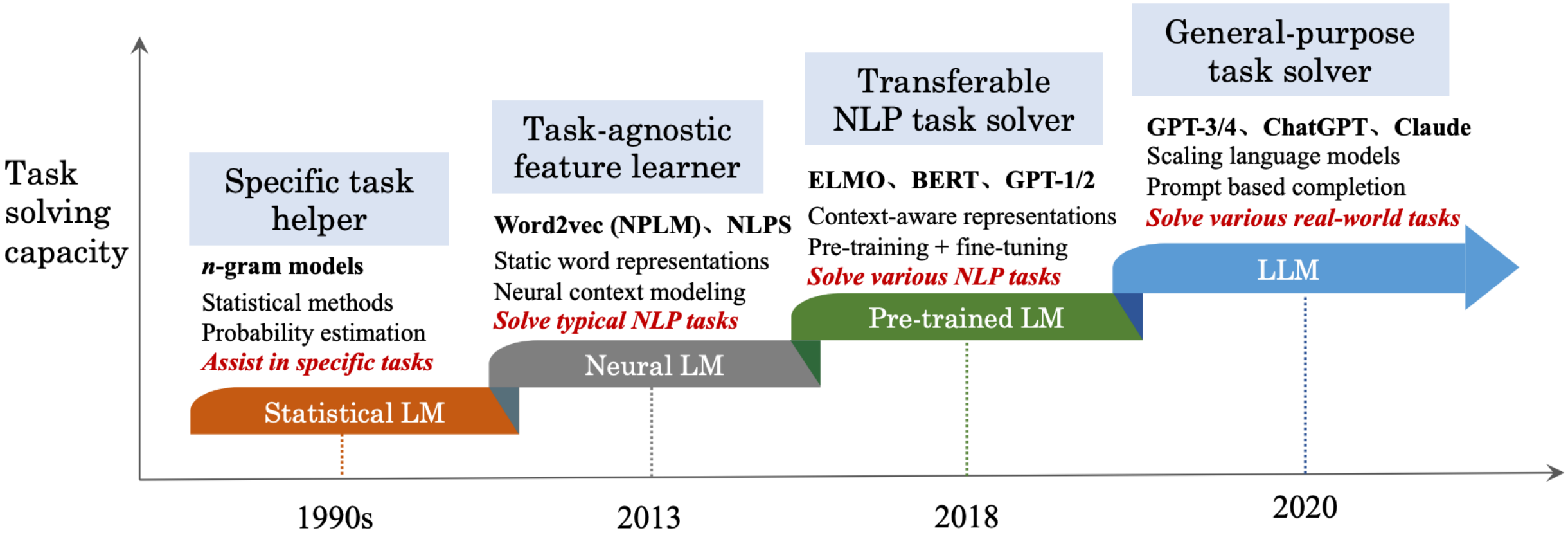
$$P(W) = P(w_1, w_2, w_3, w_4, \dots, w_n)$$

$$P(w_1, w_2, \dots, w_n) = \prod_i P(w_i | w_1 w_2, \dots, w_{i-1})$$

- P("its water is so transparent") = P(its) × P(water|its) × P(is|its water) × P(so|its water is) × P(transparent|its water is so)
- Related task: probability of an upcoming word:

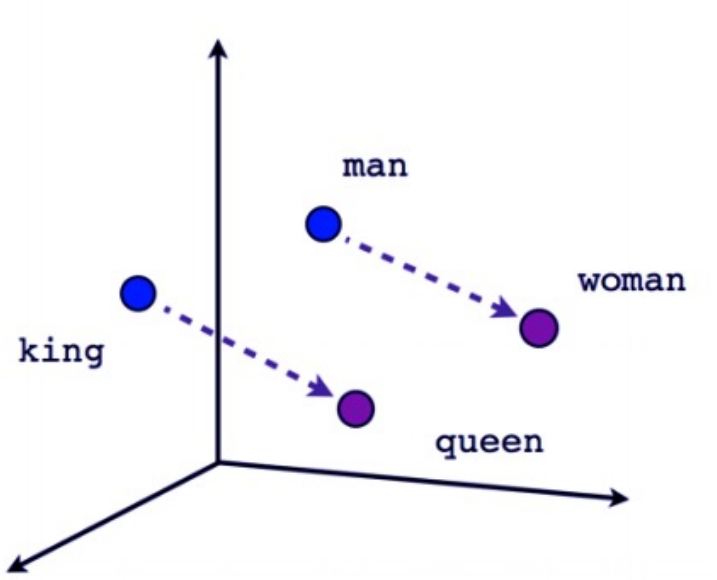
$$P(w_5 | w_1, w_2, w_3, w_4)$$

Evolution of Language Models

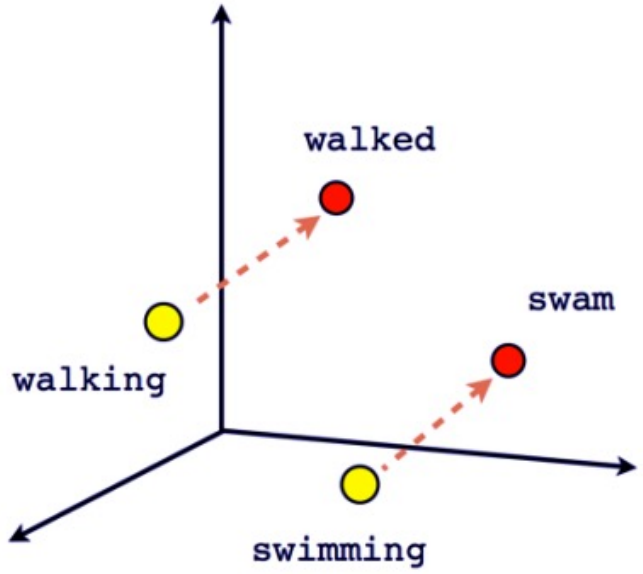


Source: https://arxiv.org/pdf/2303.18223.pdf

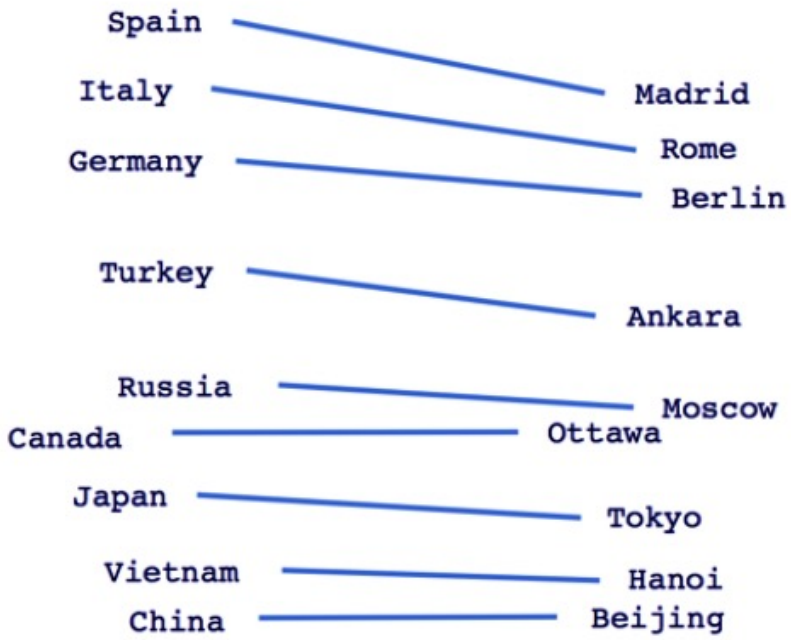
Neural Language Models (Word2Vec)



Male-Female

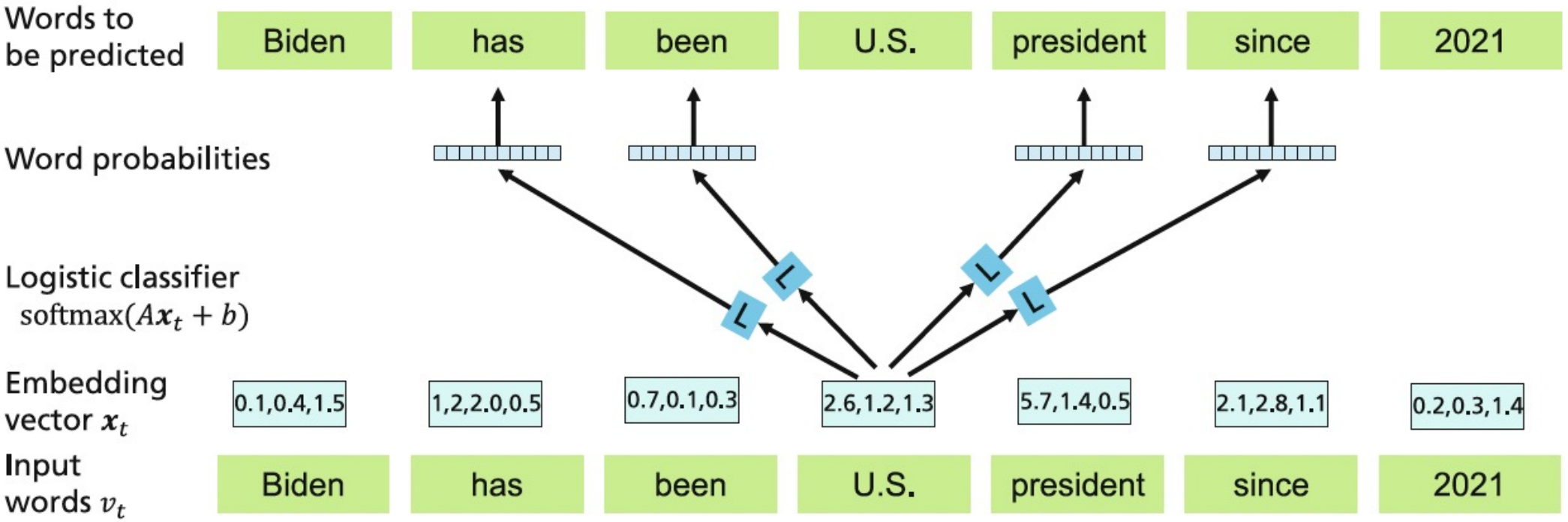


Verb tense



Country-Capital

Neural Language Models (Word2Vec)



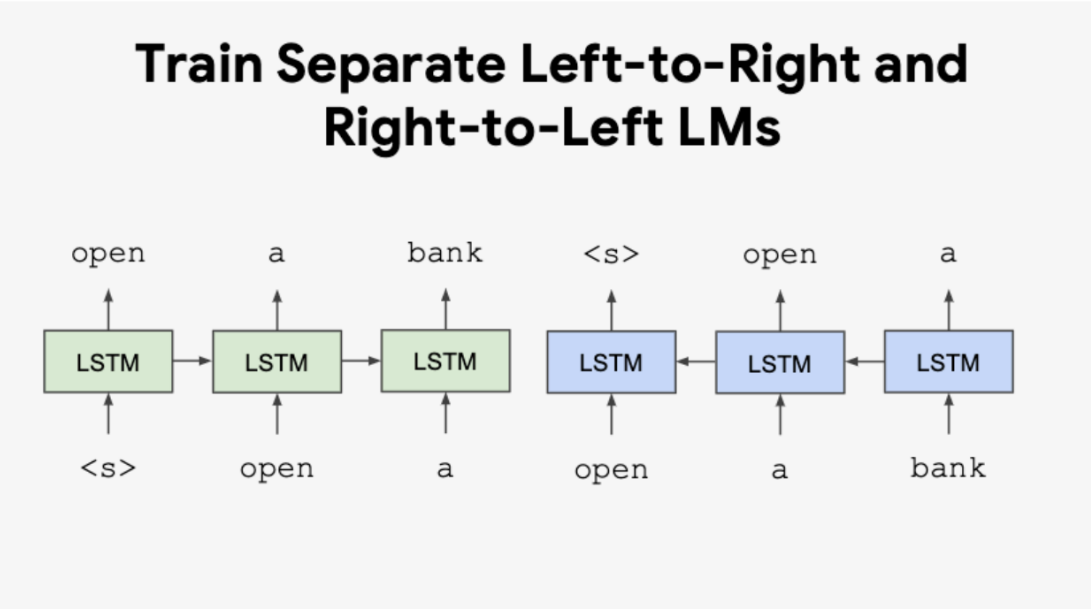
Source: Foundation Models for Natural Language Processing by Gerhard Paaß, Sven Giesselbach.

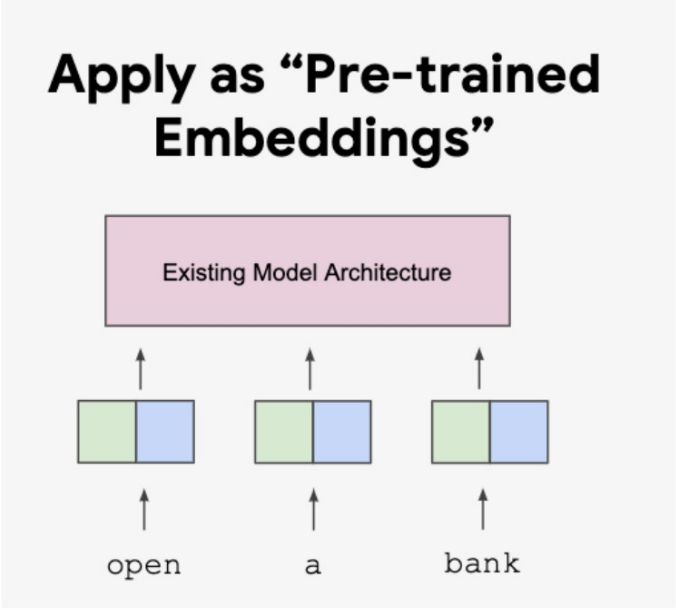
Outline

- Statistical and Neural Language Models
- Pre-trained Language Models
- Large Language Models
- Parameter Efficient Tuning
- Prompt Engineering

Prior Work: ELMo

- ELMo (Peters et al., 2018; NAACL 2018 best paper)
  - Train **two separate unidirectional LMs** (left-to-right and right-to-left) based on LSTMs
  - Feature-based approach: **pre-trained representations** used as input to task-specific models
  - Trained on single sentences from 1B word benchmark (Chelba et al., 2014)

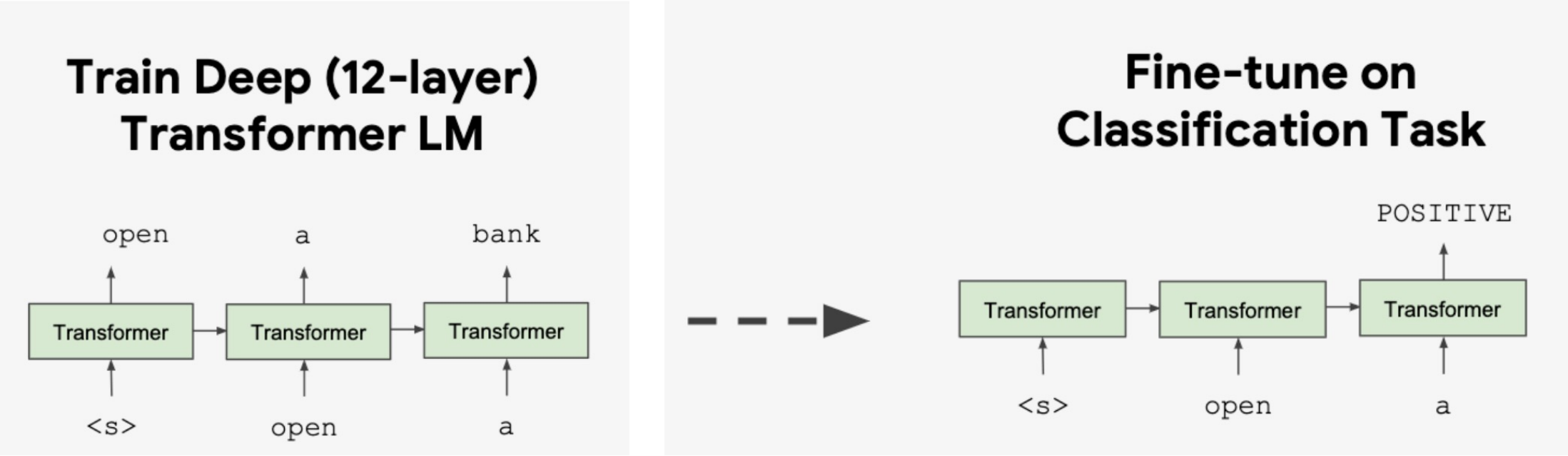




Source: https://www.cs.princeton.edu/courses/archive/fall22/cos597G/

Prior Work: OpenAI GPT

- OpenAI GPT (Radford et al., 2018; released in 2018/6)
  - Train **one unidirectional LM (left-to-right)** based on a deep Transformer decoder
  - Fine-tuning approach: all pre-trained parameters are re-used & updated on downstream tasks
  - Trained on **512-token segments on BooksCorpus** — much longer context!



BERT: key contributions

- It is a **fine-tuning** approach based on **a deep Transformer encode**
- **The key: learn representations based on bidirectional context**

- Pre-training objectives:
  - masked language modeling + next sentence prediction
- SOTA performance on a large set of sentence-level and token-level tasks

Masked Language Modeling (MLM)

- **Q:** Why we can't do language modeling with bidirectional models?

- **Solution:** Mask out k% of the input words, and then predict the masked words

store gallon

The man went to [MASK] to buy a [MASK] of milk.

MLM: masking rate and strategy

- **Q:** What is the value of k?
  - They always use k = 15%
  - Too **little** masking: computationally expensive
  - Too **much** masking: not enough context
- **Q:** How are masked tokens selected?
  - 15% tokens are **uniformly sampled**

Example: He [MASK] from Kuala [MASK] , Malaysia.

MLM: 80-10-10 corruption

For the 15% predicted words,

- 80% of the time, they replace it with [MASK] token

  went to the store ⟶ went to the [MASK]

- 10% of the time, they replace it with a random word in the vocabulary

  went to the store ⟶ went to the running

- 10% of the time, they keep it unchanged

  went to the store ⟶ went to the store

Next Sentence Prediction (NSP)

- **Motivation:** many NLP downstream tasks require understanding the relationship between two sentences (natural language inference, paraphrase detection, QA)
- NSP is designed **to reduce the gap between pre-training and fine-tuning**

[CLS]: a special token always at the beginning

[SEP]: a special token used to separate two segments

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]

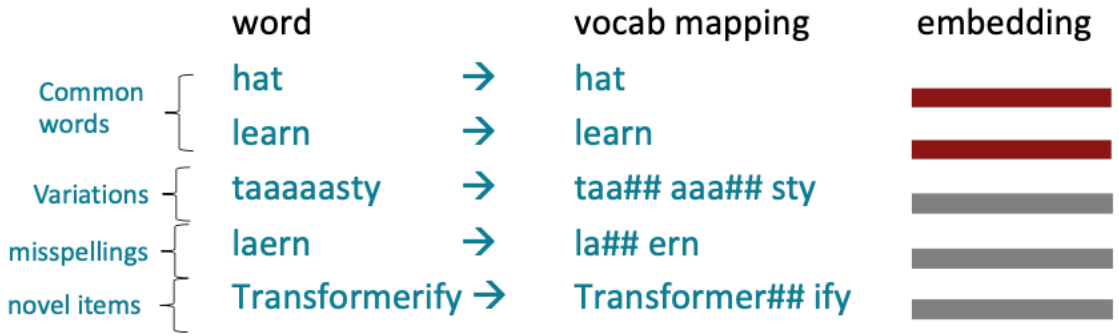
penguin [MASK] are flight ##less birds [SEP]

Label = NotNext

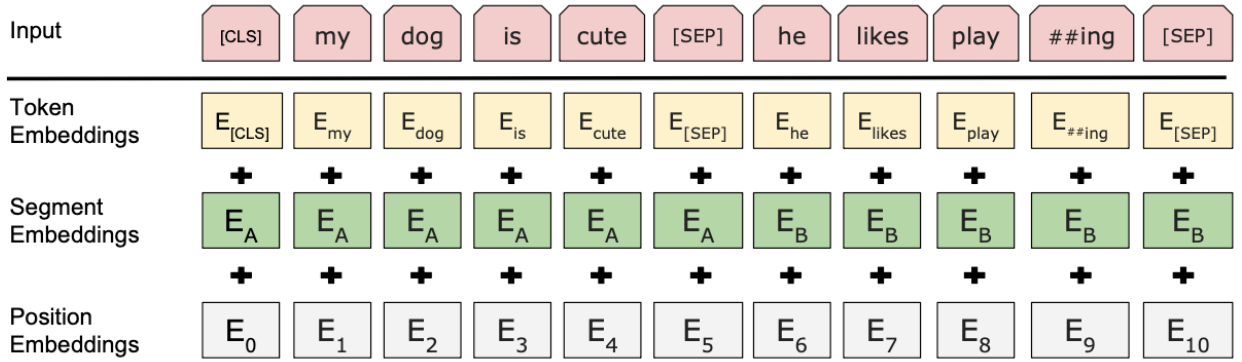
They sample two contiguous segments for 50% of the time and another random segment from the corpus for 50% of the time

BERT pre-training: putting together

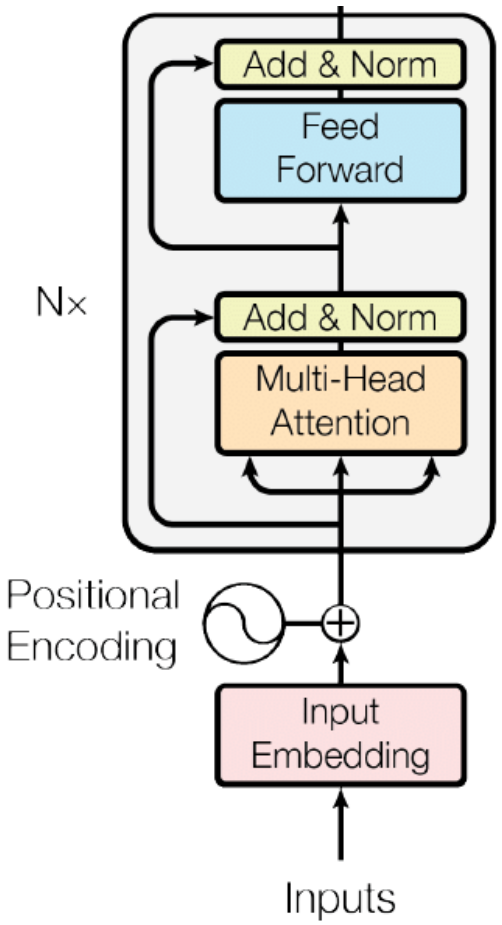
- Vocabulary size: 30,000 workpieces (common sub-word units) (Wu et al., 2016)



- Input Embeddings:



BERT pre-training: putting together

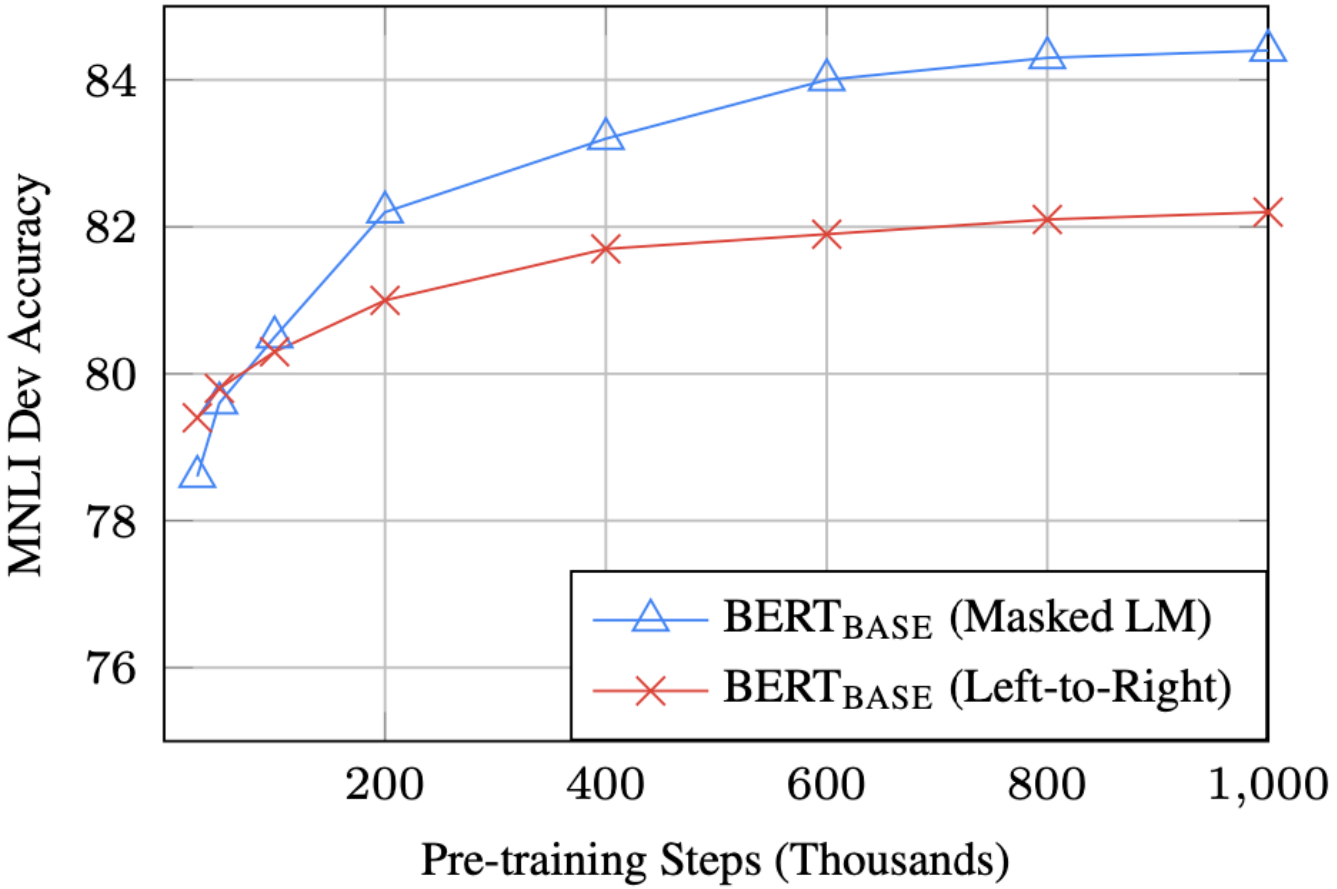


- BERT-base: 12 layers, 768 hidden size, 12 attention heads, 110M parameters
- BERT-large: 24 layers, 1024 hidden size, 16 attention heads, 340M parameters
- Training corpus: Wikipedia (2.5B) + BooksCorpus (0.8B)
- Max sequence size: 512 word pieces
  (roughly 256 and 256 for two non-contiguous sequences)
- Trained for 1M steps, batch size 128k

Same as OpenAI GPT

OpenAI GPT was trained on BooksCorpus only!

Ablation study: training efficiency



MLM takes slightly longer to converge
because it only predicts 15% of tokens

What happened after BERT?

- RoBERTa (Liu et al., 2019)
  - Trained on 10x data & longer, no NSP
  - Much stronger performance than BERT (e.g., 94.6 vs 90.9 on SQuAD)
  - Still one of the most popular models to date
- ALBERT (Lan et al., 2020)
  - Increasing model sizes by sharing model parameters across layers
  - Less storage, much stronger performance but runs slower..
- ELECTRA (Clark et al., 2020)
  - It provides a more efficient training method by predicting 100% of tokens instead of 15% of tokens

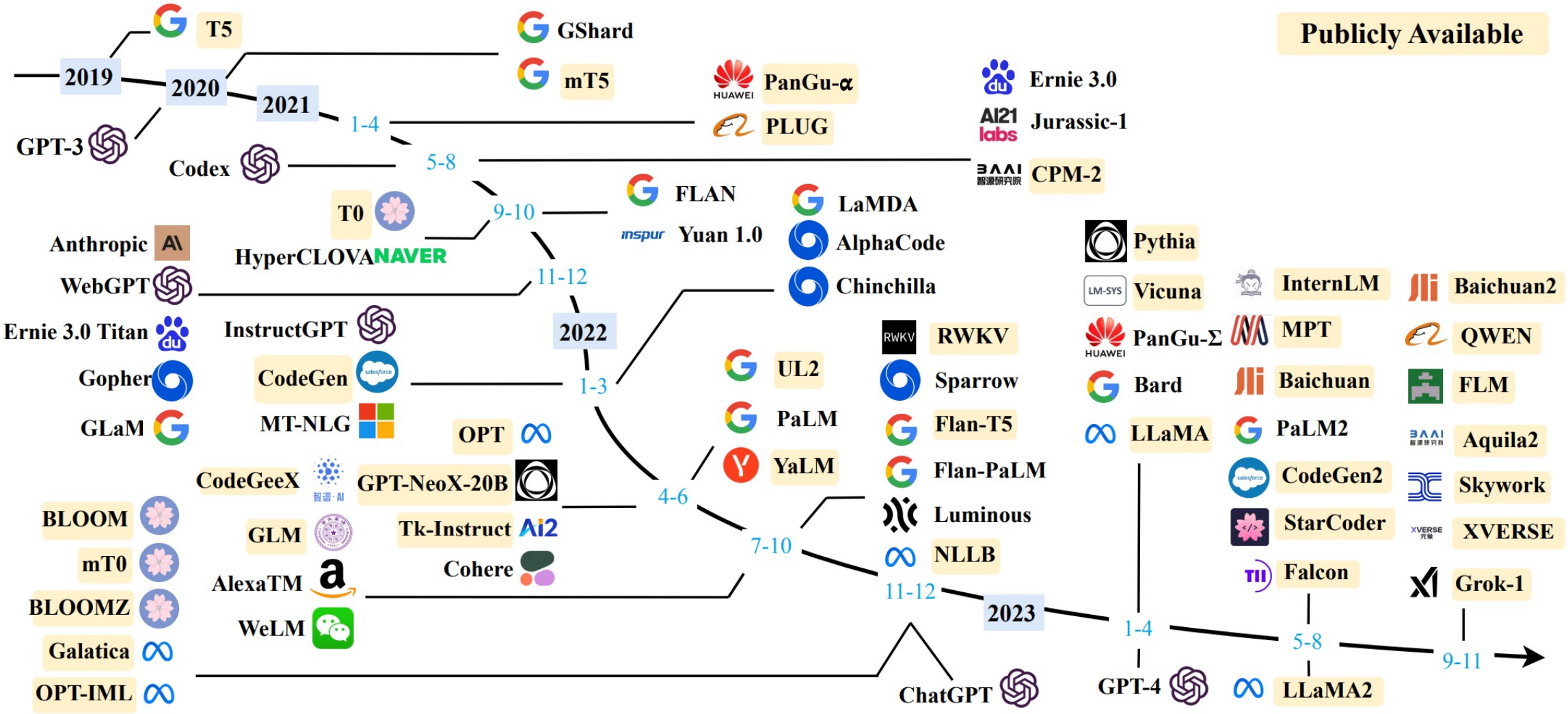
What happened after BERT?

- Models that handle long contexts (512 tokens)
  - Longformer, Big Bird, …
- Multilingual BERT
  - Trained single model on 104 languages from Wikipedia. Shared 110k WordPiece vocabulary
- BERT extended to different domains
  - SciBERT, BioBERT, FinBERT, ClinicalBERT, …
- Making BERT smaller to use
  - DistillBERT, TinyBERT, …

Outline

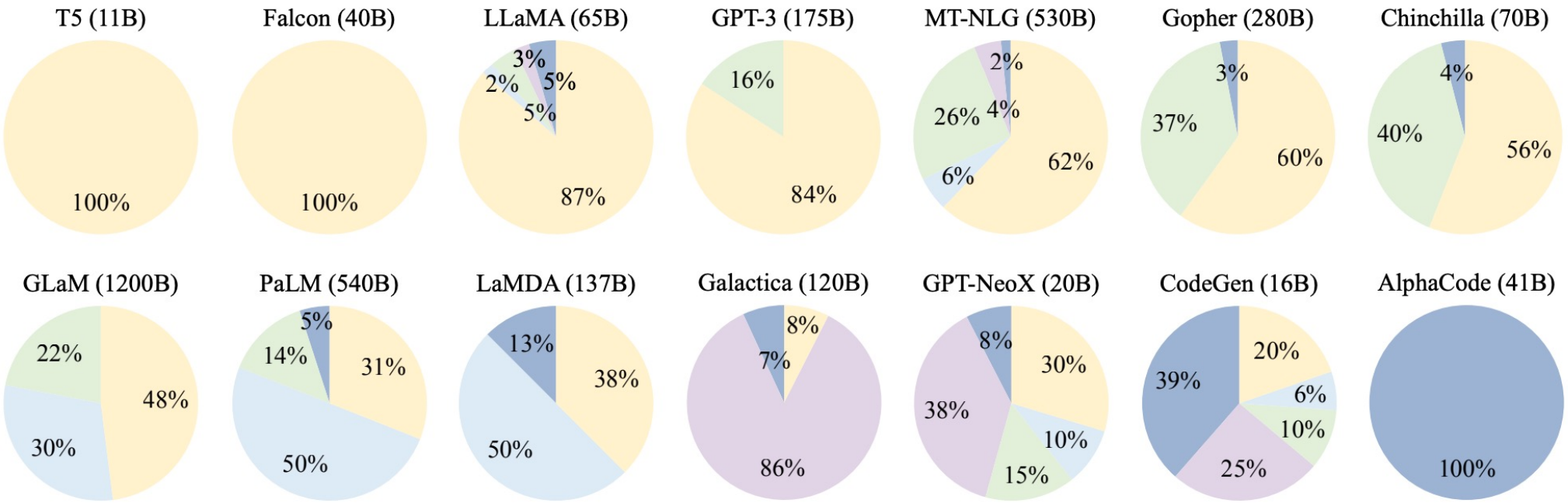
- Statistical and Neural Language Models
- Pre-trained Language Models
- Large Language Models
- Parameter Efficient Tuning
- Prompt Engineering

Large Language Models



Source: https://arxiv.org/pdf/2303.18223.pdf

Distribution of Training Data



Webpages	C4 (800G, 2019), OpenWebText (38G, 2023), Wikipedia (21G, 2023)
Conversation Data	the Pile - StackExchange (41G, 2020)
Books & News	BookCorpus (5G, 2015), Gutenberg (-, 2021), CC-Stories-R (31G, 2019), CC-NEWES (78G, 2019), REALNEWs (120G, 2019)
Scientific Data	the Pile - ArXiv (72G, 2020), the Pile - PubMed Abstracts (25G, 2020)
Code	BigQuery (-, 2023), the Pile - GitHub (61G, 2020)

Source: https://arxiv.org/pdf/2303.18223.pdf

Evaluating LLMs

- **Perplexity:** How well LLM predicts unseen words
- For an unseen test set the perplexity is given as:

$$Perplexity_\theta(w_{1:n}) = P_\theta(w_{1:n})^{-1/n}$$

model *n tokens*

- Perplexity range is [1,∞]
- The higher the probability of the word sequence, the lower the perplexity
- Thus the lower the perplexity of a model on the data, the better the model

Scaling Laws for LLMs

- Loss L as a function as # parameters N, dataset size D, compute budget C

$$L(N) = \left(\frac{N_C}{N}\right)^{\alpha_N}$$

$$L(D) = \left(\frac{D_C}{D}\right)^{\alpha_D}$$

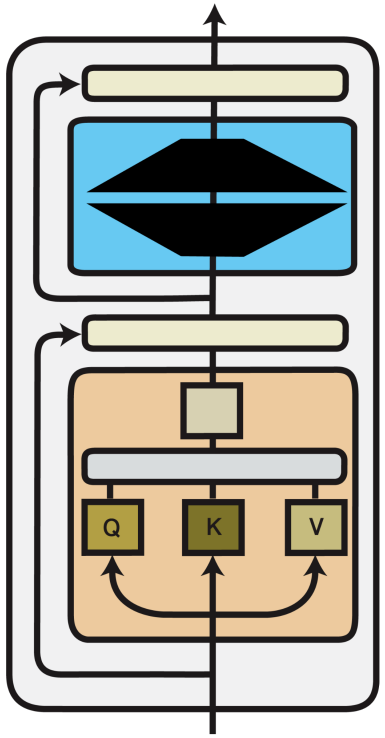
$$L(C) = \left(\frac{C_C}{C}\right)^{\alpha_C}$$

Source: https://arxiv.org/pdf/2001.08361

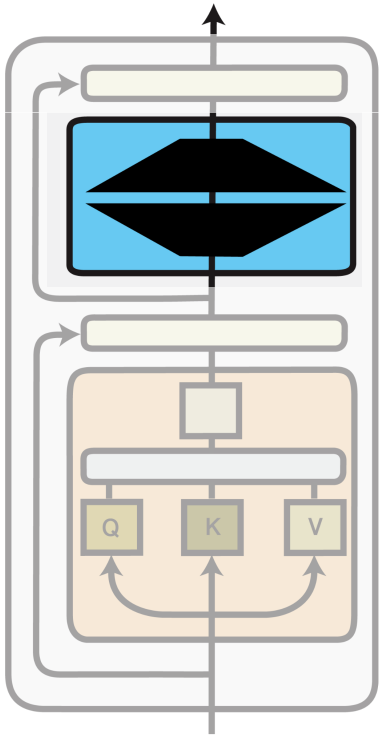
Outline

- Statistical and Neural Language Models
- Pre-trained Language Models
- Large Language Models
- Parameter Efficient Tuning
- Prompt Engineering

Fine-Tuning to Parameter Efficient Fine-Tuning



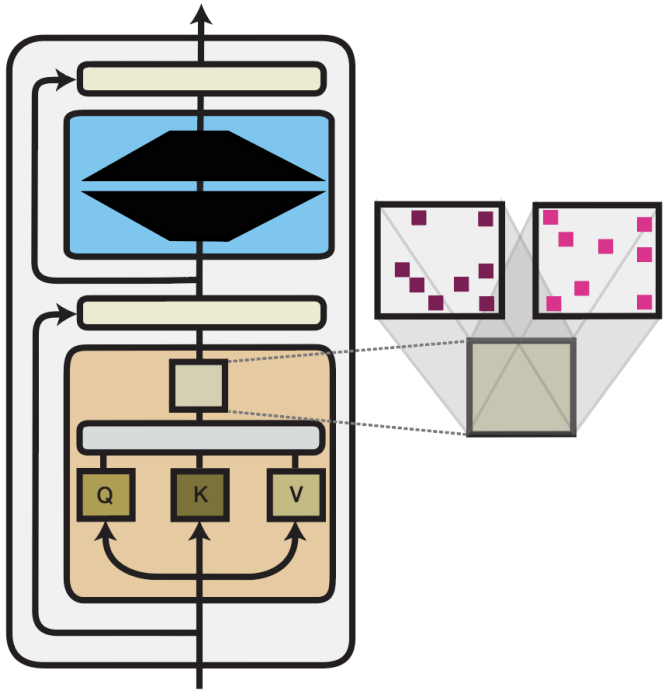
Full Fine-tuning Update
all model parameters



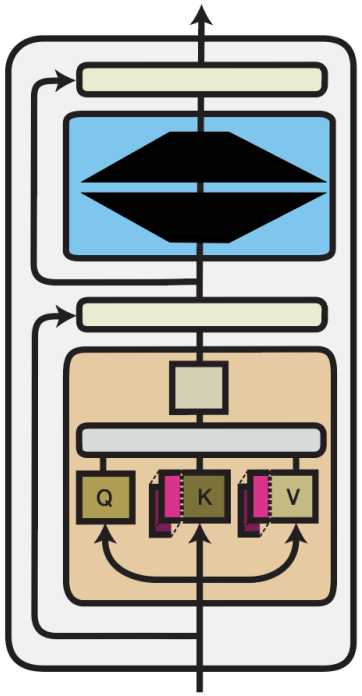
Parameter-efficient Fine-tuning
Update a small subset of model
parameters

Source: https://www.modulardeeplearning.com/

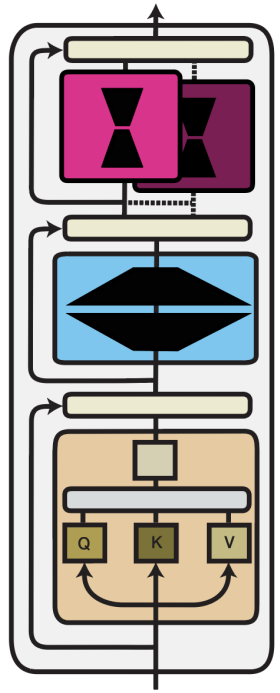
Three Computation Functions



Parameter Composition

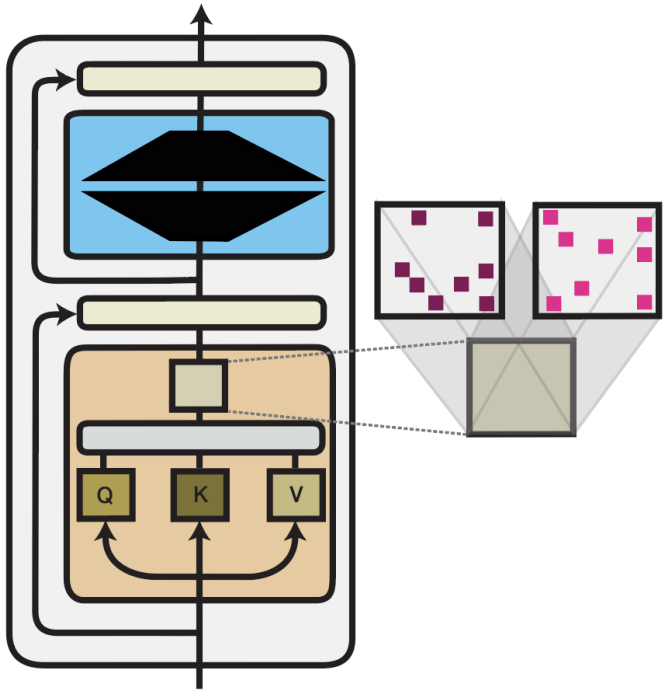


Input Composition



Function Composition

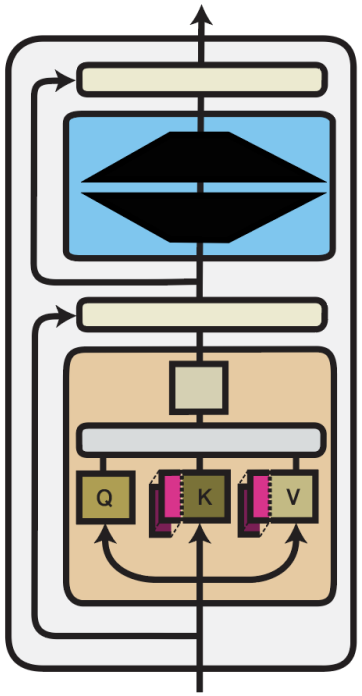
Three Computation Functions



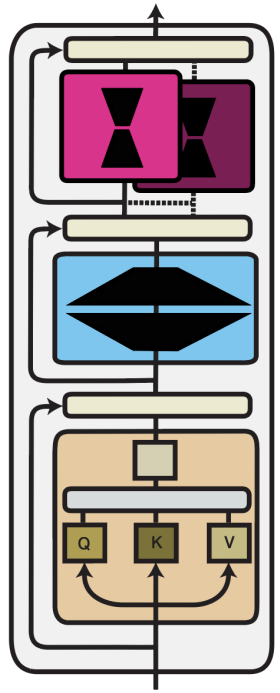
Parameter Composition

Sparse Subnetworks

Low Rank Composition



Input Composition



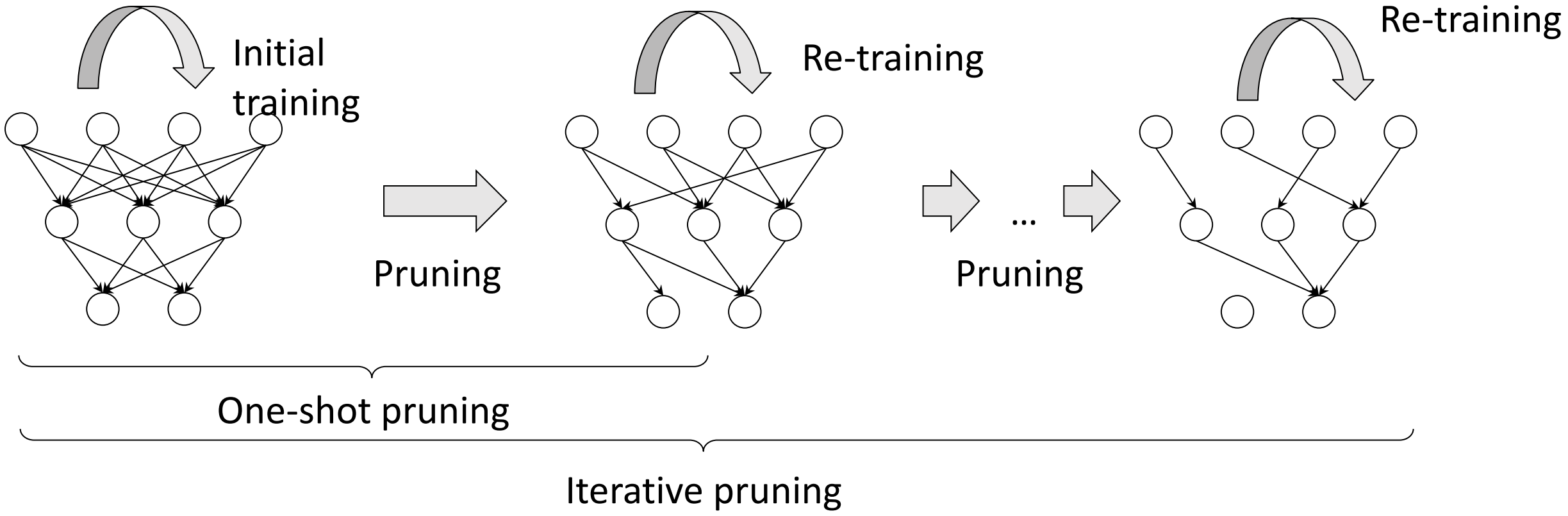
Function Composition

Sparse Subnetworks

- A common inductive bias on the module parameters is sparsity
- Most common sparsity method: pruning
- Pruning:
  - Applying a binary mask $b \in \{0,1\}^{|\theta|}$ that selectively keeps or removes each connection in a model and produces a subnetwork.
  - Most common pruning criterion: weight magnitude

Source: https://arxiv.org/abs/1607.04381

Pruning



- During pruning, a fraction of the lowest-magnitude weights are removed
- The non-pruned weights are re-trained

Pruning and Binary Mask

- Pruning as adding a task-specific vector $\emptyset$ to the parameters of an existing model: $f'_\theta = f_{\theta+\emptyset}$ where $\emptyset_i = 0$ if $b_i = 0$.

- If the final model should be sparse, we can multiply the existing weights with the binary mask to set the pruned weights to 0: $f'_\theta = f_{\theta \circ \boldsymbol{b}+\emptyset}$

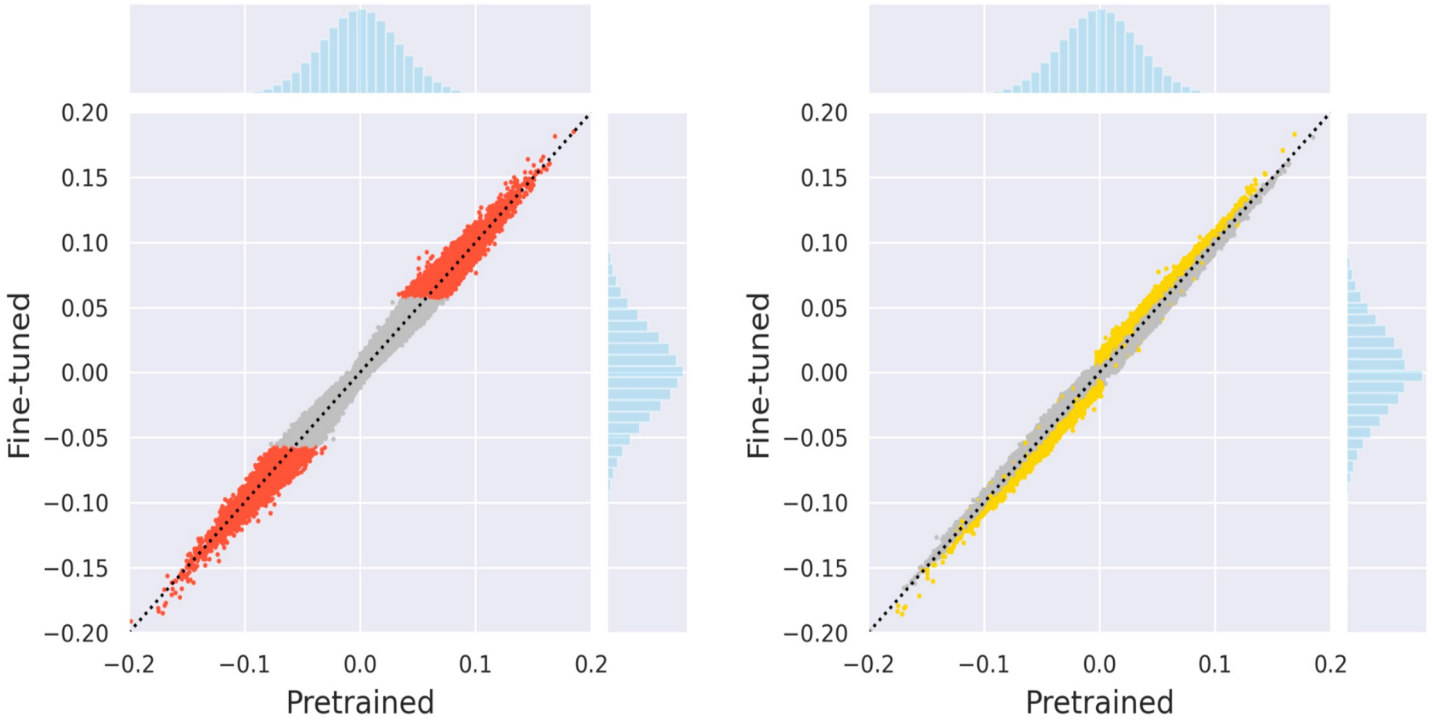
Element-wise product
(Hadamard product)

- The main benefit is that these weight values were moving to 0 anyway

Pruning Pre-trained Models

- Pruning does not consider how weights change during fine-tuning
- **Magnitude pruning**: keep weights farthest from 0
- **Movement pruning** [Sanh et al., 2020]: keep weights that *move the most away* from 0

Fine-tuned weights stay close to their their pre-trained values.
Magnitude pruning (left) selects weights that are far from 0.



Movement pruning (right) selects weights that move away from 0.

LoRA: Low-Rank Adaptation

- For each downstream task, learn a different set of parameters $\Delta_{\emptyset}$
- $|\Delta_{\emptyset}| = |\emptyset_o|$
- GPT-3 has a $|\emptyset_o|$ of 175 billion
- Expensive and challenging for storing and deploying many independent instances
- Key idea: encode the task-specific parameter increment $\Delta_{\emptyset} = \Delta_{\emptyset}(\Theta)$ by a smaller sized set of parameters Θ, $|\Theta| \ll |\emptyset_o|$
- The task of finding $\Delta_{\emptyset}$ becomes optimizing over Θ

$$\max_{\Theta} \sum_{(x,y)} \sum_{t=1}^{|y|} \log(P_{\emptyset_o + \Delta_{\emptyset}(\Theta)}(y_t | x, y_{<t}))$$

Source: https://arxiv.org/pdf/2106.09685

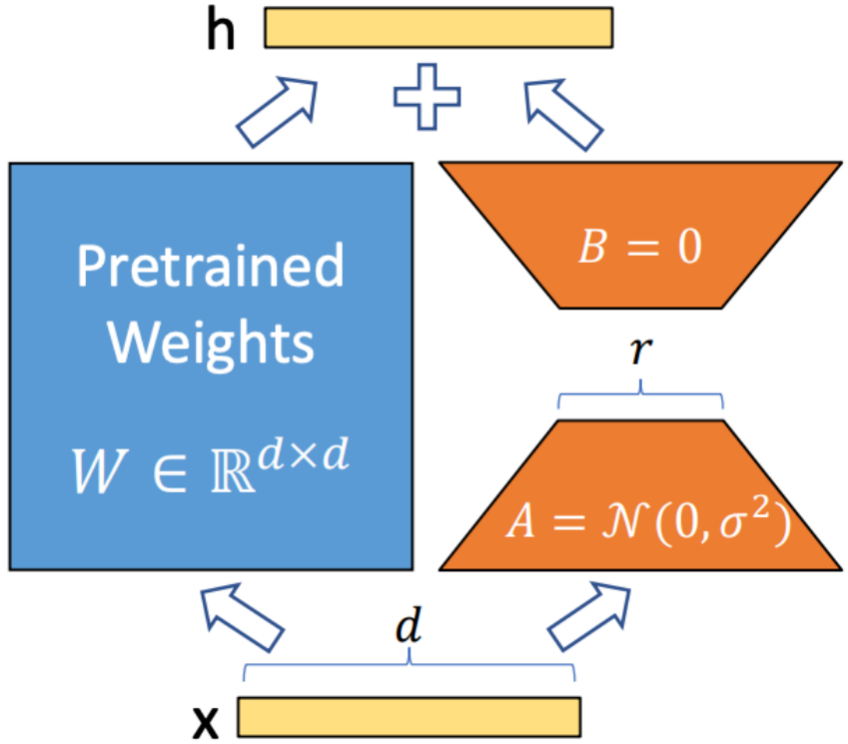
Low-rank-parameterized update matrices

- Updates to the weights have a low "intrinsic rank" during adaptation (Aghajanyan et al. 2020)
- $W_0 \in \mathbb{R}^{d\times k}$: a pretrained weight matrix
- Constrain its update with a low-rank decomposition:

$$W_0 + \Delta W = W_0 + BA$$

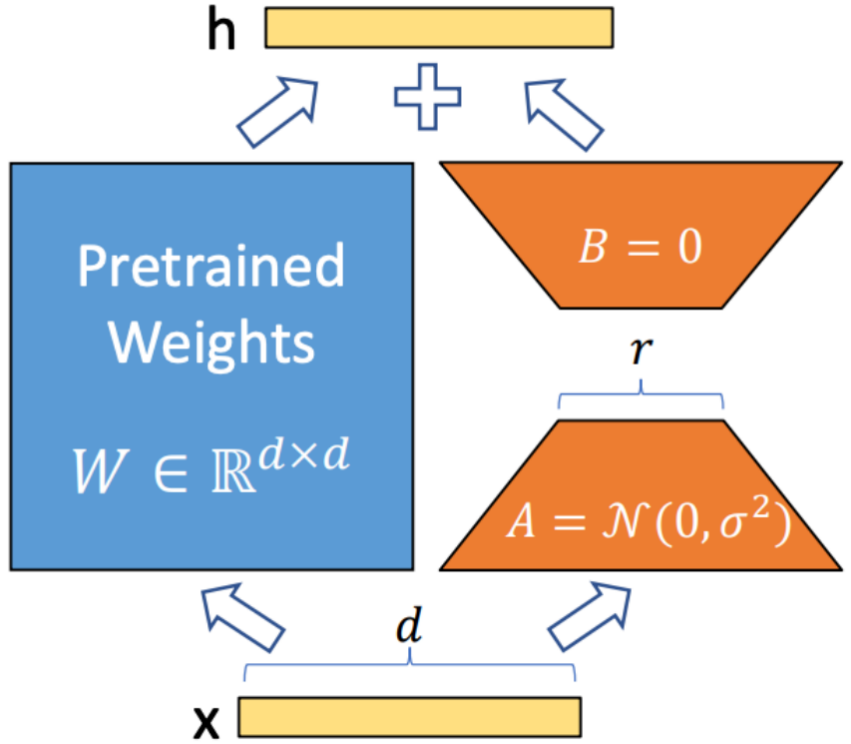
where $B \in \mathbb{R}^{d\times r}, A \in \mathbb{R}^{r\times k}, r \ll \min(d,k)$

- Only A and B contain trainable parameters



Low-rank-parameterized update matrices

- As one increase the number of trainable parameters, training LoRA converges to training the original model
- No additional inference latency: when switching to a different task, recover W_0 by subtracting BA and adding a different B' A'
- Often LoRA is applied to the weight matrices in the self-attention module

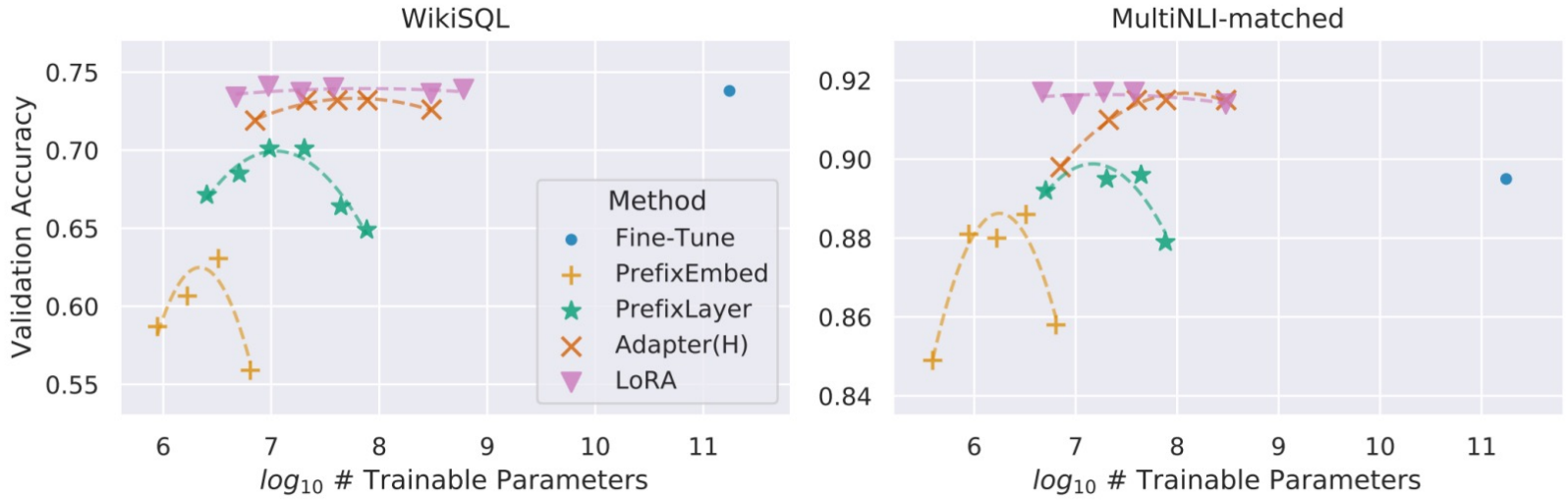


Applying LoRA to Transformers

Model & Method	# Trainable Parameters	E2E NLG Challenge				
		BLEU	NIST	MET	ROUGE-L	CIDEr
GPT-2 M (FT)*	354.92M	68.2	8.62	46.2	71.0	2.47
GPT-2 M (Adapter^{L})*	0.37M	66.3	8.41	45.0	69.8	2.40
GPT-2 M (Adapter^{L})*	11.09M	68.9	8.71	46.1	71.3	2.47
GPT-2 M (Adapter^{H})	11.09M	$67.3_{\pm.6}$	$8.50_{\pm.07}$	$46.0_{\pm.2}$	$70.7_{\pm.2}$	$2.44_{\pm.01}$
GPT-2 M (FT^{Top2})*	25.19M	68.1	8.59	46.0	70.8	2.41
GPT-2 M (PreLayer)*	0.35M	69.7	8.81	46.1	71.4	2.49
GPT-2 M (LoRA)	0.35M	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.85}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{71.8}_{\pm.1}$	$\mathbf{2.53}_{\pm.02}$
GPT-2 L (FT)*	774.03M	68.5	8.78	46.0	69.9	2.45
GPT-2 L (Adapter^{L})	0.88M	$69.1_{\pm.1}$	$8.68_{\pm.03}$	$46.3_{\pm.0}$	$71.4_{\pm.2}$	$\mathbf{2.49}_{\pm.0}$
GPT-2 L (Adapter^{L})	23.00M	$68.9_{\pm.3}$	$8.70_{\pm.04}$	$46.1_{\pm.1}$	$71.3_{\pm.2}$	$2.45_{\pm.02}$
GPT-2 L (PreLayer)*	0.77M	70.3	8.85	46.2	71.7	2.47
GPT-2 L (LoRA)	0.77M	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.89}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{72.0}_{\pm.2}$	$2.47_{\pm.02}$

Applying LoRA to GPT-3

Model&Method	# Trainable Parameters	WikiSQL Acc. (%)	MNLI-m Acc. (%)	SAMSum R1/R2/RL
GPT-3 (FT)	175,255.8M	**73.8**	89.5	52.0/28.0/44.5
GPT-3 (BitFit)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 (PreEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 (PreLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 ($\text{Adapter}^{\text{H}}$)	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 ($\text{Adapter}^{\text{H}}$)	40.1M	73.2	**91.5**	53.2/29.0/45.1
GPT-3 (LoRA)	4.7M	73.4	**91.7**	**53.8/29.8/45.9**
GPT-3 (LoRA)	37.7M	**74.0**	**91.6**	53.4/29.2/45.1



Applying LoRA to GPT-3

Which weight matrices in Transformers should we apply LoRA to?

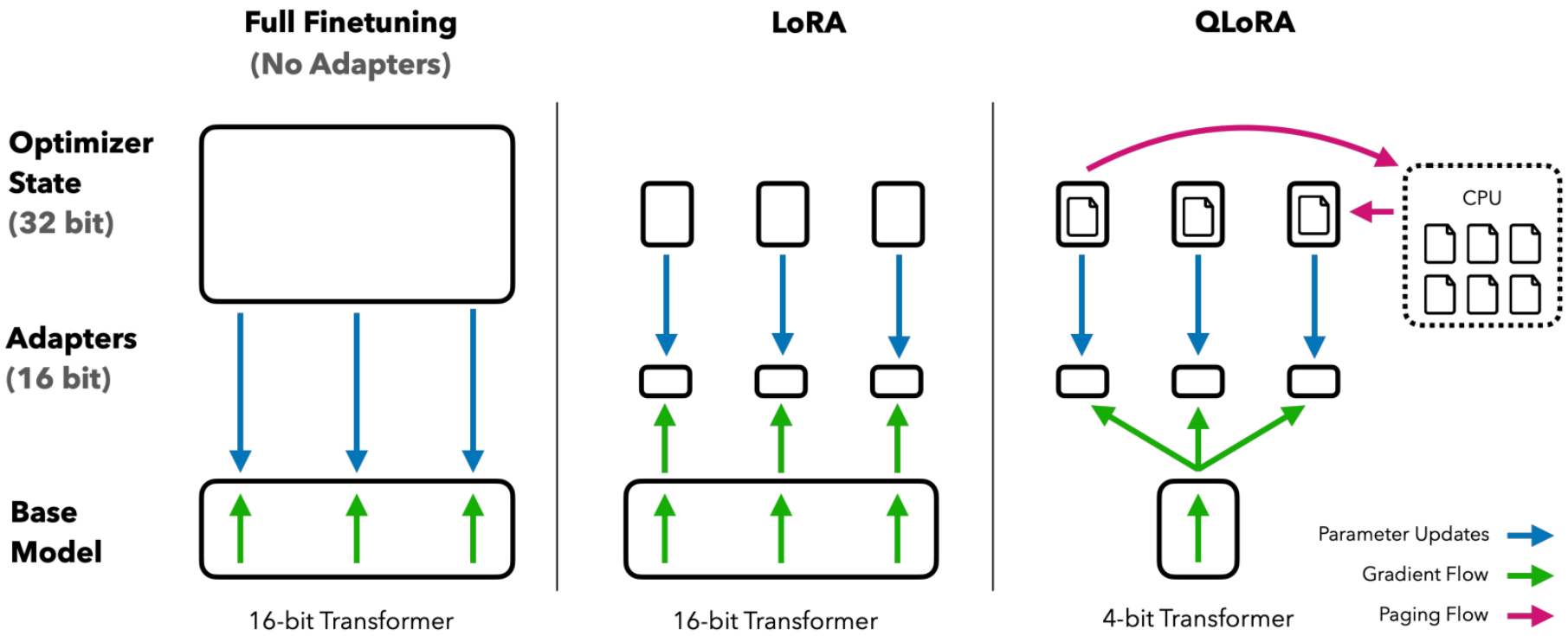
	# of Trainable Parameters = 18M						
Weight Type	W_q	W_k	W_v	W_o	W_q, W_k	W_q, W_v	W_q, W_k, W_v, W_o
Rank r	8	8	8	8	4	4	2
WikiSQL ($\pm$0.5%)	70.4	70.0	73.0	73.2	71.4	**73.7**	**73.7**
MultiNLI ($\pm$0.1%)	91.0	90.8	91.0	91.3	91.3	91.3	**91.7**

What is the optimal rank r for LoRA?

	Weight Type	$r = 1$	$r = 2$	$r = 4$	$r = 8$	$r = 64$
WikiSQL($\pm$0.5%)	W_q	68.8	69.6	70.5	70.4	70.0
	W_q, W_v	73.4	73.3	73.7	73.8	73.5
	W_q, W_k, W_v, W_o	74.1	73.7	74.0	74.0	73.9
MultiNLI ($\pm$0.1%)	W_q	90.7	90.9	91.1	90.7	90.7
	W_q, W_v	91.3	91.4	91.3	91.6	91.4
	W_q, W_k, W_v, W_o	91.2	91.7	91.7	91.5	91.4

Source: https://arxiv.org/pdf/2305.14314

From LoRA to QLoRA



- Quantizes the transformer model to 4-bit precision
- Uses paged optimizer to handle memory spikes
- 4-bit NormalFloat (NF4)
  - A new data type that is information theoretically optimal for normally distributed weights

Outline

- Statistical and Neural Language Models
- Pre-trained Language Models
- Large Language Models
- Parameter Efficient Tuning
- Prompt Engineering

Zero-shot Prompts – Elements of Prompts

A prompt is composed with the following components:

- **Instructions** - a specific task or instruction you want the model to perform
- **Context** - external information or additional context that can steer the model to better responses
- **Input data** - the input or question that we are interested to find a response for
- **Output indicator** - the type or format of the output.

Classify the text into neutral, negative or positive.

Text: I think the vacation is okay.

Sentiment:

Source: https://www.promptingguide.ai/

Few-shot Prompts

Prompt

A "**whatpu**" is a small, **furry animal native to Tanzania**. An example of a sentence that uses the word whatpu is:
We were traveling in Africa and we saw these **very cute whatpus**.
To do a "farduddle" means to jump up and down really fast.
An example of a sentence that uses the word farduddle is:

Output

When we won the game, we all started to **farduddle in celebration**.

Source: https://www.promptingguide.ai/

Chain-of-Thought Prompting

Standard Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ✗

Chain-of-Thought Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

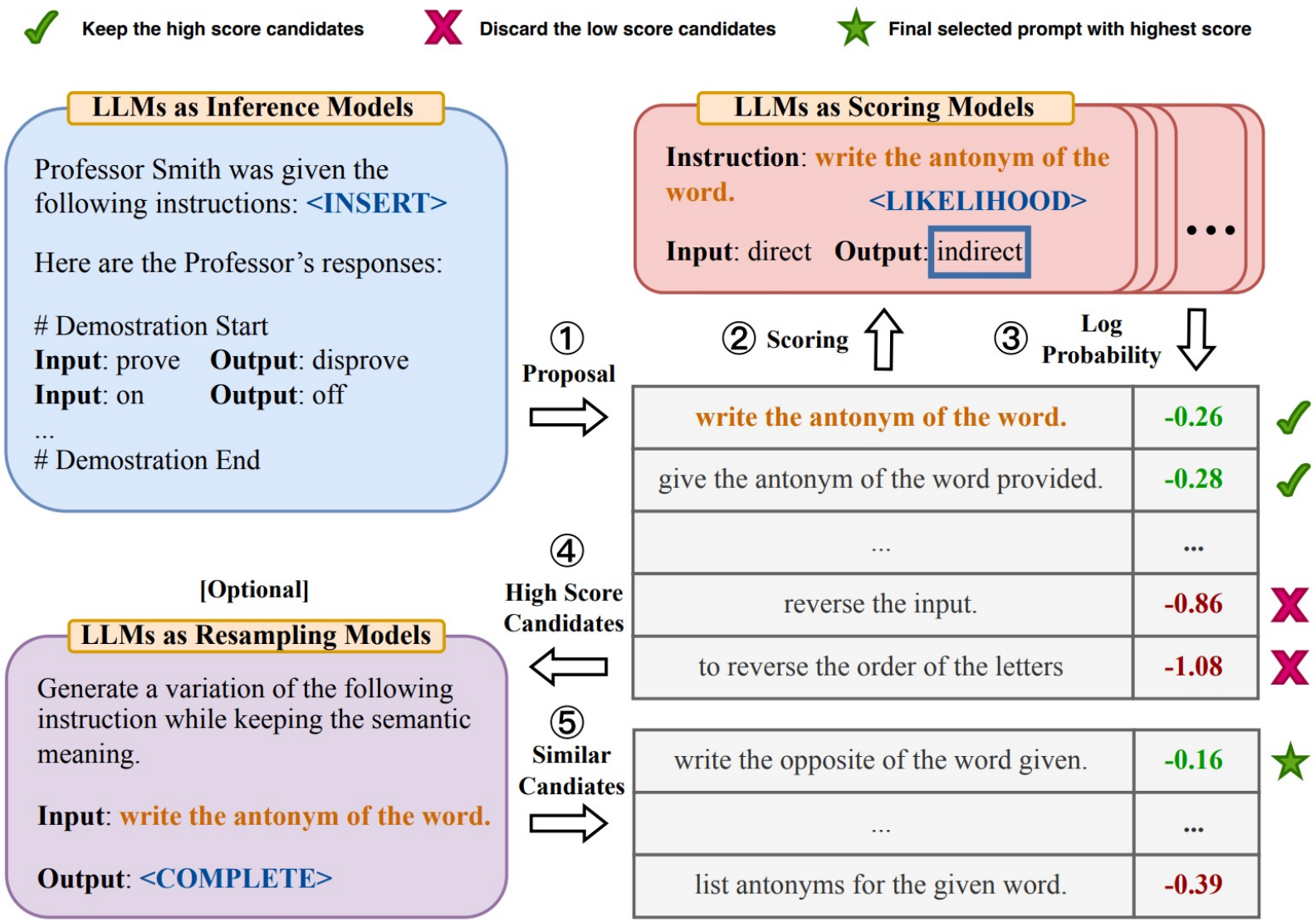
Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9. ✓

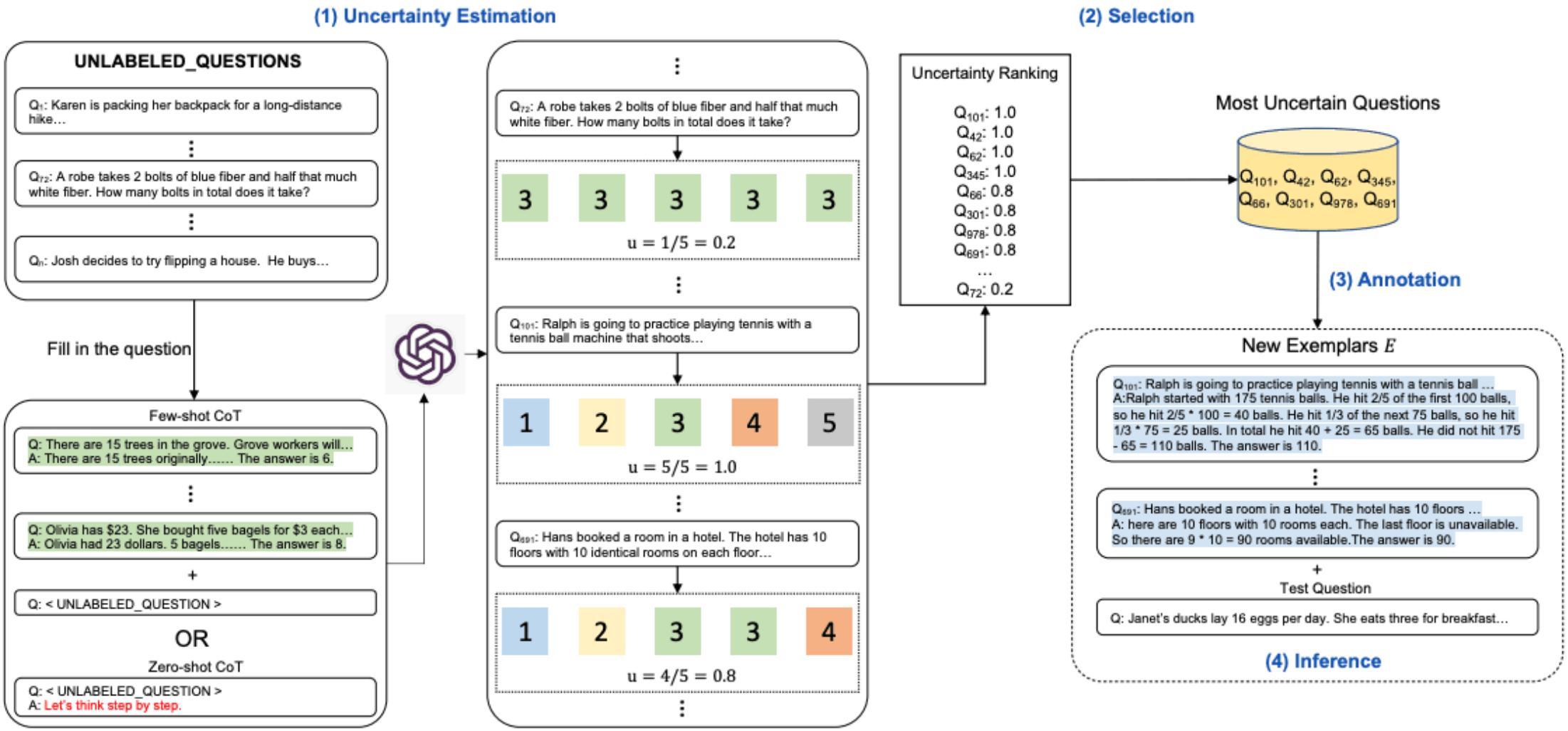
Source: https://www.promptingguide.ai/

Automatic Prompt Engineer



Source: https://arxiv.org/pdf/2211.01910

Active Prompting with Chain-of-Thought



Source: https://aclanthology.org/2024.acl-long.73.pdf

Summary

- Statistical and Neural Language Models
- Pre-trained Language Models
- Large Language Models
- Parameter Efficient Tuning
- Prompt Engineering